Neural Networks with {-1,1} or {-1,0,1} Weights Playing Atari Galaxian (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 evolutionary training of neural networks (NNs) with binary, ternary, and real-valued connection weights using Evolution Strategy (ES). The NNs are trained to control the Atari Galaxian game, with the objective of maximizing the game score. A three-layer feedforward neural network is employed, in which both connection weights and unit biases are represented using one of three parameter domains: {-1, 1}, {-1, 0, 1}, or the continuous interval [-1.0, 1.0]. The experiments examine the effects of ES hyperparameter configurations, connection-weight representations, and the number of hidden units on game-playing performance. Two ES configurations with the same total number of fitness evaluations are compared. The results show that the configuration using a larger offspring population and fewer generations significantly outperforms the configuration using a smaller offspring population and more generations for all three weight representations. Under the better-performing configuration, NNs with binary and ternary weights generally achieve higher game scores than those with real-valued weights, although the differences are not statistically significant. Furthermore, for ternary-weight NNs, a network with four hidden units significantly outperforms one with a single hidden unit and achieves performance comparable to networks with 8, 16, and 32 hidden units, providing a favorable trade-off between model size and game-playing performance.

Keywords: evolutionary algorithm, evolution strategy, neuroevolution, Atari game, binary neural network, ternary neural network.

1. Introduction

Neural networks (NNs) have been widely used in a variety of tasks, including classification, prediction, and control. In particular, neural networks can serve as controllers for computer games by mapping game states to actions. Reinforcement learning provides a framework for training such controllers through interactions with an environment, in which the objective is to maximize the cumulative reward. However, conventional gradient-based training methods are not always straightforward to apply to problems in which the objective is defined by the final performance of a controller rather than by a differentiable loss function.

Evolutionary algorithms provide an alternative approach to neural network training by optimizing network parameters through population-based search. Instead of calculating gradients, these algorithms evaluate candidate networks according to their performance and iteratively improve the population through evolutionary operations. Evolution Strategy (ES) is one such algorithm and can be applied to the optimization of neural network parameters using the game score as the fitness function.

The representation of neural network parameters is an important consideration in evolutionary training. In conventional neural networks, connection weights are typically represented by real-valued parameters. However, restricting connection weights to a small set of discrete values may offer practical advantages, including reduced model storage requirements and potentially simpler implementations. In particular, binary connection weights, which take values from {-1,1}, and ternary connection weights, which take values from {-1,0,1}, provide alternatives to real-valued connection weights. Nevertheless, the effect of these different representations on evolutionary training performance requires further investigation. Although real-valued

weights provide a larger parameter search space, this larger space may also make it more difficult for an evolutionary algorithm to identify high-performing solutions within a limited evaluation budget.

Another important consideration is the configuration of the evolutionary algorithm and the architecture of the neural network. For a fixed number of fitness evaluations, different combinations of offspring population size and number of generations may lead to different optimization outcomes. Similarly, increasing the number of hidden units increases the number of trainable parameters and the complexity of the search space. Consequently, a larger neural network does not necessarily achieve better performance when its parameters are optimized by an evolutionary algorithm. Understanding the relationships among parameter representation, evolutionary search, and network size is therefore important for developing effective evolutionary training methods.

This study investigates the evolutionary training of neural networks with binary, ternary, and real-valued connection weights using ES. The networks are trained to control the Atari Galaxian game, with the objective of maximizing the game score. The game state, represented by the contents of the Atari system's RAM, is provided as the network input, and the network selects an action based on its output. A three-layer feedforward neural network, or multilayer perceptron (MLP), is used as the controller. Both connection weights and unit biases are represented using the same parameter domain, allowing the effects of different parameter representations to be examined under a common network architecture.

The experiments address four main questions. First, two ES configurations with the same total number of fitness evaluations but different combinations of offspring population size and number of generations are compared to investigate their effects on optimization performance. Second, the game-playing performance of networks with binary, ternary, and real-valued connection weights is compared to examine how the parameter representation affects the effectiveness of evolutionary training. Third, the effect of the number of hidden units is investigated to assess the trade-off between network size and game-playing performance. Finally, learning curves are analyzed to characterize the progress of evolutionary optimization and the variation in solution quality across independent runs.

The experimental results provide insights into the selection of ES hyperparameters, connection-weight representations, and network architectures for evolutionary neural network training. In particular, the findings help clarify whether restricting connection weights to discrete values can be advantageous when ES is used to optimize neural network controllers under a fixed evaluation budget.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the Atari Galaxian environment and the evaluation procedure. Section 4 presents the neural network architecture and the parameter representations considered in this study. Section 5 explains the ES-based training procedure. Section 6 describes the experimental configurations, including the network architectures and ES hyperparameters. Section 7 presents and discusses the experimental results, including statistical comparisons and learning-curve analyses. Finally, Section 8 concludes the paper and outlines directions for future research.

2. Related Work

Evolutionary algorithms have been widely investigated as derivative-free optimization methods for neural network training. In particular, Evolution Strategy (ES) optimizes candidate solutions through population-based search and selection, without requiring gradient information [1], [2]. Salimans et al. demonstrated that

ES served as a scalable alternative to conventional reinforcement learning methods and reported its effectiveness on benchmark tasks, including Atari games [3]. These studies indicate that evolutionary optimization is a viable approach to training neural network controllers when candidate solutions can be evaluated directly through their performance in an environment.

Another relevant research direction concerns neural networks with discrete-valued connection weights. Conventional neural networks generally use real-valued weights, whereas binary and ternary neural networks restrict weights to a small set of discrete values. Courbariaux et al. proposed BinaryConnect, which uses binary weights during forward and backward propagation while retaining higher-precision parameters for gradient accumulation [4]. Hubara et al. subsequently introduced binarized neural networks in which both weights and activations are binary during inference, demonstrating the potential of binary representations to reduce memory requirements and computational costs [5]. For ternary representations, Liu et al. investigated ternary weight networks with weights restricted to {-1, 0, 1} and discussed their potential for reducing model size and computational requirements [6]. These studies establish the relevance of discrete weight representations, although their training methods and evaluation objectives differ from the evolutionary optimization approach considered in the present study.

More directly related to this work are our previous studies of neural network controllers trained using ES to play Atari Space Invaders. In the first study, binary connection weights in {-1, 1} were investigated, and the results indicated that binary-weight networks could achieve game-playing performance comparable to that of networks with 64-bit real-valued weights for the task examined [7]. The study also reported that increasing the number of offspring per generation was more effective than increasing the number of generations under the tested evaluation budget. In a subsequent study, ternary connection weights in {-1, 0, 1} were examined and compared with binary weights [8]. Although ternary-weight networks achieved higher average scores in some comparisons, the observed differences between binary and ternary representations were not statistically significant. The subsequent study also reported that a larger offspring population combined with fewer generations improved ES performance.

The present study extends this line of research in several respects. First, it compares three parameter representations—binary weights in {-1, 1}, ternary weights in {-1, 0, 1}, and real-valued weights in [-1.0, 1.0]—within a common neural network architecture and training framework. Second, it evaluates these representations on Atari Galaxian rather than Space Invaders, thereby examining whether the observations from the previous studies also hold in a different game environment. Third, it investigates the effects of ES configurations under a fixed total number of fitness evaluations and examines the influence of the number of hidden units on game-playing performance. By combining these comparisons with statistical tests and learning-curve analyses, this study provides a systematic evaluation of how parameter representation, search configuration, and network size affect ES-based neural network training.

3. Atari Galaxian

The experiments use the Atari *Galaxian* environment provided by the Arcade Learning Environment (ALE) and accessed through the Gymnasium framework[1]. A neural network is trained using an evolutionary algorithm to maximize the total episode reward, which corresponds to the game score. Figure 1 shows a screenshot of the game.

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

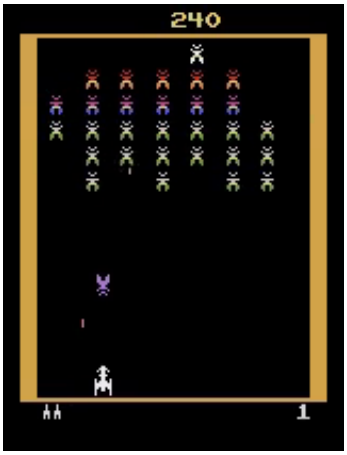


Figure 1. Screenshot of Atari *Galaxian*.

Galaxian is a classic arcade game in which the player controls a spacecraft and attempts to destroy enemy formations while avoiding hostile attacks. In this study, the neural network receives the game's RAM state as input and selects an action at each time step. The environment was initialized using the identifier `ALE/Galaxian-ram-v5`, with the frame-skip parameter set to 1. Consequently, the environment was configured to advance by one game frame per environment step, without the additional frame skipping. At the beginning of each evaluation, the environment was reset using a fixed seed of 0. The observation returned by the environment consists of the contents of the Atari system's RAM. Each observation element is first divided by 255 and then linearly transformed to the interval [-1.0, 1.0]. Specifically, the normalized input vector $\boldsymbol{x}$ is calculated from the original observation vector $\boldsymbol{o}$ as $\boldsymbol{x} = 2(\boldsymbol{o}/255) - 1$. The resulting vector is passed to the neural network, whose output is computed by a forward-propagation procedure. The action is selected by applying the argmax operation to the network's output vector; that is, the action corresponding to the output unit with the largest activation is selected. The selected action is then executed in the game environment. The actions are (1) no operation (NOOP), (2) fire, (3) move right, (4) move left, (5) move right while firing, or (6) move left while firing.

The evaluation continues until the episode terminates. At each time step, the reward returned by the environment is accumulated to obtain the total episode reward R which is the same as the final game score. The game terminates if the controller loses all available lives.

4. Neural Network Architecture and Parameter Representations

The neural network employed in this study has a three-layer feedforward architecture, commonly referred to as a multilayer perceptron (MLP), as illustrated in Figure 2. Three different sets of values are considered for both the connection weights and the unit biases: binary values {-1, 1}, ternary values {-1, 0, 1}, and real-valued parameters within the closed interval [-1.0, 1.0]. The hyperbolic tangent (tanh) function is used as the activation function for all units in the hidden and output layers. Consequently, the output of each output-layer unit is a real number in the open interval (-1.0, 1.0) regardless of whether the connection weights and biases are binary, ternary, or real-valued.

The MLP serves as the policy function: action(t) = F(observation(t)). The `ALE/Galaxian-ram-v5` environment returns 128 RAM values in each observation. Accordingly, the input layer contains 128 units (N=128 in Figure 2). Each raw observation value o_i, which lies in the interval [0,255], is normalized to the

interval [-1.0,1.0] according to $x_i = 2(o_i/255)-1$.

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.

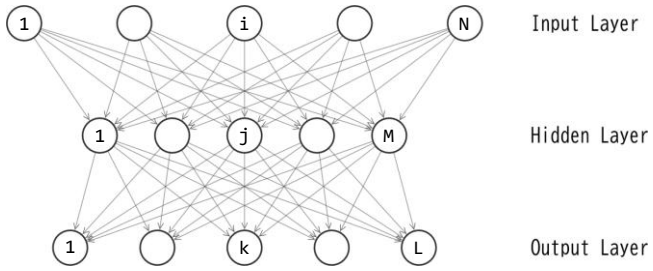


Figure 2. Topology of the MLP.

5. Evolution Strategy for Neural Network Training

The three-layer 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, ternary or real-valued variable, i.e., $x_i \in \{-1, 1\}$, $x_i \in \{-1, 0, 1\}$ or $-1.0 \leq x_i \leq 1.0$. By applying the value of each element in $\mathbf{x}$ to its corresponding connection weight or unit bias, the feedforward calculations can be processed.

The D-dimensional vector $\mathbf{x}$ is optimized using Evolution Strategy [1]. 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 phenotype of $\mathbf{x}$. Figure 3 illustrates the ES process. The process is the same as that in the previous studies [7,8]. In Step 1, vectors $\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^C$ are randomly initialized where C denotes the number of offspring. 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 plays a game. The fitness of $\boldsymbol{y}^c$ is defined as the game score obtained by the MLP. 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. A greater value of pm promotes exploration more. The mutation method in Step 5-4 differs depending on the type of connection weights. When the connection weights are binary or ternary, each of $y_1^c, y_2^c, \ldots, y_D^c$ is mutated under the probability pm. A greater value of pm promotes exploration

more. When the connection weights are real-valued, $\boldsymbol{y}^c$ is perturbed according to Eqs. (1)-(3), where s is a hyperparameter called the *step size*, and *rand* denotes a uniformly distributed random number sampled from the interval [-1.0, 1.0]. A larger value of s promotes exploration of the search space.

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

Figure 4. Reproduction process in Evolution Strategy.

$$\mathrm{y}_\mathrm{d}^\mathrm{c} \leftarrow \mathrm{y}_\mathrm{d}^\mathrm{c} + \mathrm{s} * \mathrm{rand} \tag{1}$$

$$\mathrm{if}\, \mathrm{y}_\mathrm{d}^\mathrm{c} < -1.0, \mathrm{y}_\mathrm{d}^\mathrm{c} \leftarrow -1.0 \tag{2}$$

$$\mathrm{if}\, 1.0 < \mathrm{y}_\mathrm{d}^\mathrm{c}, \mathrm{y}_\mathrm{d}^\mathrm{c} \leftarrow 1.0 \tag{3}$$

6. Experiments

6.1 Neural Network

This study employs six configurations for the number of hidden units; M ∈ {1, 2, 4, 8, 16, 32}.

6.2 Evolution Strategy

Table 1 presents ES hyperparameter configurations. 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 = 10,000 (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). The mutation probability *pm* is adopted when the genotype values are binary or ternary, while the step size s is adopted when the genotype is real-valued.

Table 1. ES hyperparameter configurations.

Hyperparameters	(a)	(b)
Number of offspring (C)	10	50
Generations (G)	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) for binary/ternary genotypes	0.01	0.01
Step size (s) for real-valued genotypes	0.25	0.25

7. Results and Discussion

Table 2 reports statistical summaries of the game scores obtained after evolutionary learning. A higher game score indicates better game-playing performance by the trained neural network (NN). For example, the row corresponding to M=1 in Table 2(i)(a) shows the results of 11 independent evolutionary learning trials in which the connection weights were binary {-1, 1}, the ES configuration specified in Table 1(a) was used, and the number of hidden units was set to 1. Across these 11 trials, the best game score was 3450, the worst was 540, the mean was 2450.9, and the median was 2630. The values in the remaining rows of Table 2 can be interpreted in the same way.

A comparison of configurations (a) and (b) in Table 2(i) shows that configuration (b) yields higher values than configuration (a). A Wilcoxon signed-rank test was performed using the 24 data points reported for configuration (a) in Table 2(i) and the 24 corresponding data points for configuration (b) in Table 2(i). The resulting p-value was 7.00e-4, indicating that configuration (b) significantly outperforms configuration (a) for the ES algorithm. Similar results were observed in Tables 2(ii) and 2(iii), where configuration (b) also yielded higher values than configuration (a). The corresponding Wilcoxon signed-rank tests yielded p-values of 3.25e-4 for Table 2(ii) and 1.57e-3 for Table 2(iii). Thus, configuration (b) significantly outperformed configuration (a) in all three cases of the binary weights, the ternary weights, and the real-valued weights.

These results are consistent with those of our previous study [7, 8], indicating that the optimization performance of ES improves when a larger number of offspring is prioritized over a larger number of generations. This improvement may be attributed to the ability of a larger offspring population to compensate for a potential weakness of ES. Although ES is effective at local search, it may be less effective at global exploration. Configuration (a) uses a larger number of generations than configuration (b), which may favor local search, whereas configuration (b) uses a larger number of offspring, which may promote global exploration. Consequently, ES with configuration (a) may exhibit stronger local search capabilities at the expense of global exploration, whereas configuration (b) may provide a better balance between local search and global exploration. This improved balance is considered a possible factor contributing to the enhanced search performance of ES.

Next, the game scores obtained using the three types of connection weights—binary, ternary, and real-valued weights—are compared. Since configuration (b) was found to outperform configuration (a), the following analysis focuses exclusively on the results for configuration (b). A comparison of the values in Tables 2(i)(b), 2(ii)(b), and 2(iii)(b) shows that, in general, the values in Table 2(iii)(b) are smaller than those in Tables 2(i)(b) and 2(ii)(b). A Wilcoxon signed-rank test was performed using the 24 data points reported in Table 2(i)(b) and the 24 data points reported in Table 2(iii)(b), yielding a p-value of 0.076. Thus, no statistically significant difference was found between binary and real-valued connection weights. Nevertheless, the p-

value is substantially below 0.5, suggesting that binary connection weights contributed better than real-valued connection weights. Similarly, a Wilcoxon signed-rank test was performed using the 24 data points reported in Table 2(ii)(b) and the 24 data points reported in Table 2(iii)(b). The resulting p-value was again 0.076. Thus, no statistically significant difference was found between ternary and real-valued connection weights. Nevertheless, the p-value is substantially below 0.5, suggesting that ternary connection weights contributed better than real-valued connection weights.

Table 2. Summary statistics of game scores over 11 independent runs.

(i) binary weights

	M	Best	Worst	Mean	Median
(a)	**1**	3450	540	2450.9	2630
	2	3800	2240	2960.9	2830
	4	4210	2670	3276.4	3190
	8	4230	2820	3286.4	3190
	16	3920	3040	3389.1	3330
	32	3410	2670	3100.0	3100
(b)	**1**	4960	940	2995.5	3020
	2	4210	2380	3175.5	3250
	4	4290	2510	3387.3	3310
	8	3970	2830	3352.7	3410
	16	4120	2830	3462.7	3290
	32	4050	2860	3299.1	3140

(ii) ternary weights

	M	Best	Worst	Mean	Median
(a)	**1**	3160	940	2439.1	2750
	2	3860	2510	3205.5	3270
	4	3890	2400	3099.1	3190
	8	3790	2720	3170.0	3140
	16	3570	2660	3181.8	3220
	32	3910	1950	3092.7	2970
(b)	**1**	3780	1500	2860.0	2800
	2	4650	1530	3120.0	2910
	4	4320	2800	3494.5	3480
	8	4140	2870	3393.6	3310
	16	4190	3020	3504.5	3520
	32	3970	3010	3460.9	3470

(iii) real-valued weights

	M	Best	Worst	Mean	Median
(a)	**1**	3310	940	1878.2	1470
	2	3690	940	2576.4	2660
	4	4570	1540	2688.2	2820
	8	3580	1010	2803.6	2980
	16	3650	1860	2987.3	3000
	32	3880	1990	2997.3	2970
(b)	**1**	4210	1060	3171.8	3260
	2	4100	2850	3370.0	3370
	4	3920	2580	3343.6	3280
	8	3780	2180	3107.3	3170
	16	3870	2810	3202.7	3050
	32	3460	3050	3248.2	3280

Using binary or ternary connection weights instead of real-valued connection weights can reduce the amount of storage required to record trained NN models and the amount of main memory required to run the models. In general, however, real-valued connection weights offer greater potential for achieving higher performance than binary or ternary connection weights. In the present experiments, real-valued connection weights could take any value in the interval [-1.0, 1.0], which includes all three values, -1, 0, and 1, available to binary and ternary connection weights. Consequently, every optimal solution in the optimization search space for binary connection weights, denoted by V_1, is necessarily contained in the optimization search space for real-valued connection weights, denoted by V_3. Similarly, every optimal solution in the optimization search space for ternary connection weights, denoted by V_2, is necessarily contained in V_3. Furthermore, since V_3 contains solutions that do not exist in either V_1 or V_2, it may contain solutions that are superior to the optimal solutions in V_1 and V_2 respectively. Therefore, the best performance achievable by an NN with real-valued connection

weights is at least as high as that achievable by an NN with binary or ternary connection weights. Despite this theoretical advantage, the NNs trained in the experiments achieved higher performance with binary or ternary connection weights than with real-valued connection weights. This result may be explained by the substantially larger number of solutions in V_3 than in V_1 or V_2, which makes it more difficult for the ES algorithm to find good solutions in V_3 than in V_1 or V_2. Let D denote the dimensionality of the solution space, which is common to V_1, V_2, and V_3. The binary and ternary representations define search spaces containing 2^D and 3^D parameter vectors, respectively. In contrast, the real-valued representation allows a much larger set of parameter vectors within the interval ([-1.0, 1.0]). The larger search space may make optimization more difficult, although further experiments are needed to establish whether this explains the observed results.

Next, game scores are compared across different numbers of hidden units. A smaller number of hidden units is desirable because it reduces the NN model size, thereby decreasing memory requirements and computational costs. In general, however, increasing the number of hidden units is expected to improve the performance of the trained NN. The ternary-weight NNs were trained using ES under configuration (b) in 11 independent trials. Consequently, 11 data points were obtained for each number of hidden units M. A Wilcoxon rank-sum test was performed using the 11 data points obtained for M=4 and those obtained for M=1, yielding a p-value of 4.28e-3. Thus, the NN with M=4 significantly outperformed the NN with M=1. Similarly, a Wilcoxon rank-sum test comparing the 11 data points for M=4 with those for M=2 yielded a p-value of 0.079. Although no statistically significant difference was found between M=4 and M=2, the p-value was substantially below 0.5, suggesting that M=4 is better than M=2. In contrast, Wilcoxon rank-sum tests comparing the 11 data points for M=8 with those for M=4, M=16, and M=32 yielded p-values of 0.65, 0.59, and 0.54, respectively. These results indicate that the NN with M=4 was competitive compared to those with M=8, M=16, or M=32. Taken together, these results indicate that M=4 provides the best trade-off between NN size and performance. The absence of performance improvements for M=8, M=16, and M=32 may be attributed to the increased difficulty of the optimization problem as the number of hidden units increases. Specifically, the total number of connection weights D increases with M as described above (i.e., D=135M+6). Consequently, the search space for ES, which contains 3^D possible solutions, also grows exponentially, making it more difficult for ES to discover better solutions within the search space. The results of the present experiments suggest that, when ES is used for evolutionary training of ternary-weight NNs, increasing the model size can substantially increase the difficulty of the search process and does not necessarily improve the performance of the trained NNs.

Figures 5-7 show the learning curves obtained using the ternary, binary, and real-valued connection weights respectively, where the number of NN hidden units was 4 and the ES hyperparameters were (b) in Table 1. The horizontal axis represents the number of solution evaluations on a logarithmic scale, whereas the vertical axis represents the game score. The three curves in each figure correspond to the best, median, and worst trials among 11 independent runs.

Figure 5 presents the learning curves obtained using the ternary connection weights. In Fig. 5, the fitness generally improves as the number of evaluations increases. However, the improvement is not continuous; instead, periods of stagnation are followed by substantial increases in fitness. This behavior is observed in all three representative trials, although the timing and magnitude of the improvements differ. The median trial exhibits a rapid increase in fitness during the early stages of the search, reaching approximately 2,200 within only a few evaluations. The fitness then remains relatively stable for a considerable number of evaluations before increasing to approximately 3,100. Further improvements occur during the later stages, and the final fitness reaches approximately 3,500. These results indicate that a relatively high fitness can be obtained at an early stage, while continued exploration can still lead to further improvements. In contrast, the best trial initially exhibits a more gradual improvement. Its fitness reaches approximately 1,800 after around 100 evaluations and subsequently increases in several stages. Notably, substantial improvements continue to

occur even after several thousand evaluations. The final fitness reaches approximately 4,300, which is the highest value among the three representative trials. This result suggests that a relatively slow improvement during the early stages does not necessarily prevent an evolutionary algorithm from finding a high-quality solution through continued exploration. The worst trial also exhibits an overall increase in fitness, although its improvement is more limited than that of the other two trials. After an initial improvement, the fitness remains relatively stable for an extended period. It subsequently increases to approximately 2,300 and undergoes further gradual improvements, eventually reaching approximately 2,800. The final fitness values of the best, median, and worst trials are approximately 4,300, 3,500, and 2,800, respectively. The difference between the best and worst trials indicates that the final solution quality can vary substantially across independent runs. Moreover, the distinct improvement patterns suggest that the search trajectory depends on the particular run. Overall, the results demonstrate that ES can progressively improve neural network performance when ternary connection weights are used. Nevertheless, the observed stagnation periods and differences among the three trials indicate that the effectiveness of the search cannot be adequately characterized by examining only the initial improvement or a single trial.

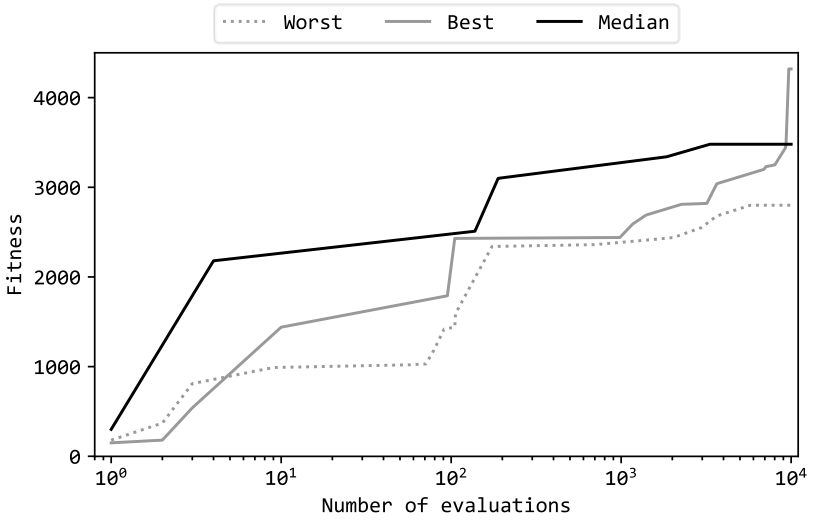


Figure 5. Learning curves of MLP with {-1,0,1} weights and 4 hidden units.

Figure 6 presents the learning curves obtained using the binary connection weights. The trajectories differ substantially among the best, worst and median trials, particularly in the timing of major improvements and the final fitness values. In the median trial, the fitness initially increases gradually and reaches approximately 1,200 after several dozen evaluations. A substantial improvement then occurs at around 100 evaluations, increasing the fitness to approximately 2,600. Following this increase, the fitness improves more gradually, eventually reaching approximately 3,300. This trajectory indicates that a major improvement can occur during the intermediate stage of the search, followed by further, smaller improvements. The best trial exhibits a different pattern. Its fitness remains relatively low during the early stages, reaching approximately 800 after several dozen evaluations. It subsequently increases in several stages, with particularly notable improvements after approximately 1,000 evaluations and again during the final stage of the search. The fitness eventually reaches approximately 4,300. This is the highest final fitness among the three representative trials. The late-stage improvement in the best trial is particularly noteworthy. Despite its relatively modest fitness during the early stages, the trial ultimately achieves a substantially higher fitness than the median and worst trials. This observation reinforces the importance of examining the entire learning trajectory rather than evaluating search performance solely on the basis of early-stage results. The worst trial exhibits a more gradual improvement over most of the search process. Its fitness increases progressively, reaching approximately

2,200 during the later stages. Although a further increase is observed near the end of the search, the final fitness remains considerably lower than that of the best trial.

The final fitness values of the best, median, and worst trials were approximately 4,300, 3,400, and 2,500, respectively. The difference between the best and worst trials indicates substantial variation in the final solution quality across independent runs. Overall, these results demonstrate that the evolutionary algorithm can identify high-fitness neural network configurations even when the connection weights are restricted to two discrete values. However, the substantial differences among the representative trials suggest that the resulting performance depends on the search trajectory. The improvements observed during the later stages also indicate that terminating the search solely because of limited early-stage progress may prevent the discovery of better solutions.

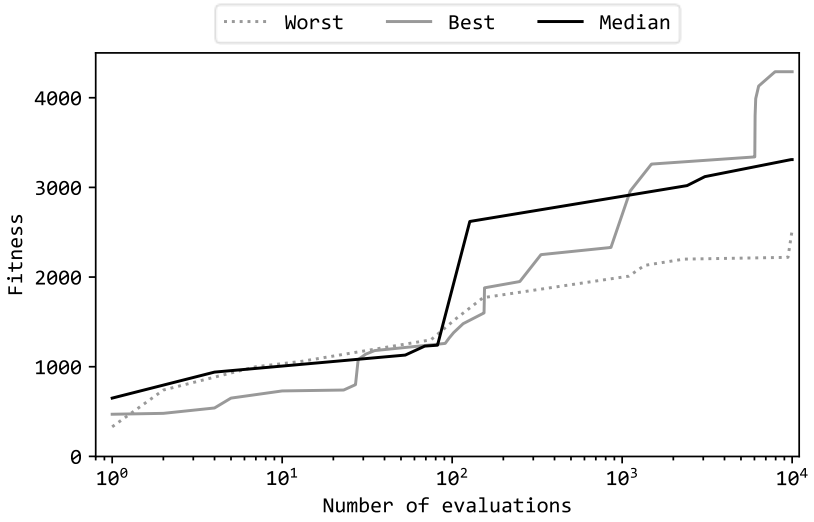


Figure 6. Learning curves of MLP with {-1,1} weights and 4 hidden units.

Figure 7 shows the learning curves obtained using the real-valued connection weights and four hidden units. As observed in the experiments with discrete weights, the fitness generally increases with the number of solution evaluations. Nevertheless, the learning curves exhibit distinct periods of stagnation and substantial improvement, and the timing of these changes varies among the three trials. The median trial initially achieves a fitness of approximately 900 and remains at a relatively stable level for a short period. A substantial increase subsequently occurs, raising the fitness to approximately 1,900 after several dozen evaluations. The fitness then remains nearly unchanged for an extended interval. Further improvements occur at approximately 700 evaluations and again during the later stages of the search. The final fitness reaches approximately 3,300. The best trial exhibits a different learning trajectory. Its fitness initially increases from approximately 400 to around 1,000 and subsequently improves more gradually. A substantial increase occurs after approximately 100 evaluations, followed by further improvements. Another marked increase is observed after approximately 2,000 evaluations, and the fitness continues to rise during the final stages. The final fitness reaches approximately 3,900, the highest value among the three representative trials. These observations suggest that substantial improvements can occur at different stages of the search, even when real-valued connection weights are used. In particular, the best trial continues to improve considerably after the initial stages, indicating that the search may benefit from additional evaluations beyond the point at which early progress becomes limited. The worst trial exhibits a more gradual improvement during the initial and intermediate stages. Its fitness increases substantially at around 600 evaluations and continues to improve, eventually reaching approximately 2,600. However, the fitness subsequently remains nearly unchanged,

indicating that further evaluations yield little additional improvement in this trial. The final fitness values of the best, median, and worst trials are approximately 3,900, 3,300, and 2,600, respectively. These differences demonstrate that the final solution quality varies across independent runs, even when the connection weights are represented by real-valued parameters. Overall, the results show that the evolutionary algorithm can obtain high-fitness solutions using real-valued connection weights within the interval [-1.0,1.0]. However, the differences in the learning trajectories indicate that the search behavior varies across trials. Furthermore, the stagnation periods observed in all three curves suggest that the ability to adjust connection weights continuously does not necessarily result in continuous improvements in fitness.

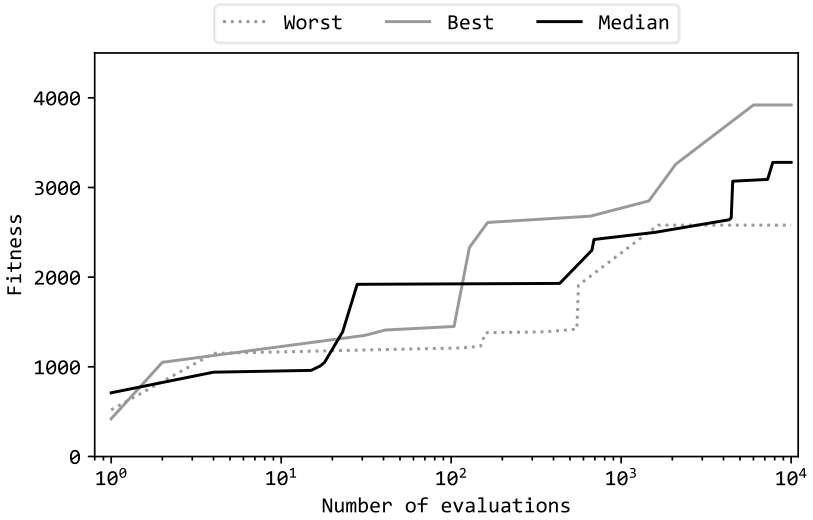


Figure 7. Learning curves of MLP with [-1.0, 1.0] weights and 4 hidden units.

A common feature is that fitness generally increases with the number of solution evaluations. This indicates that the evolutionary algorithm can progressively identify better neural network configurations under all three weight representations. However, the improvement is not continuous. In all three experiments, periods of stagnation are followed by substantial increases in fitness. These changes occur at different evaluation counts across the representative trials, suggesting that the search trajectory depends on the particular run. Another common characteristic is the difference between the best, median, and worst trials. In each experiment, the best trial achieves a higher final fitness than the worst trial, indicating that the final solution quality varies across independent runs. The observed differences also show that the fitness achieved during the early stages does not necessarily predict the final performance. In particular, some trials exhibit substantial improvements after hundreds or thousands of evaluations. These observations suggest that continuing the search beyond the early stages can be beneficial, even when the fitness has temporarily stagnated. Nevertheless, the learning curves alone do not establish that additional evaluations will always improve the solution quality. Overall, the experiments share two important characteristics: fitness generally improves as the search progresses, and the improvement proceeds through a combination of stagnation and substantial increases rather than through a smooth, continuous process.

Although all three experiments exhibit stagnation followed by improvement, the timing and magnitude of these changes differ. In the ternary-weight experiment, the median trial achieves a high fitness relatively early, reaching approximately 2,200 after only a few evaluations. The best trial, in contrast, improves more gradually during the early stages but eventually reaches approximately 4,300. A further increase near the end of the search contributes to its highest final fitness. In the binary-weight experiment, the median trial exhibits a particularly large increase at around 100 evaluations, reaching approximately 2,600. The best trial

undergoes substantial improvements at later stages and ultimately reaches approximately 4,300. The worst trial, however, reaches a final fitness of only approximately 2,200. The gap between the best and worst trials is therefore larger than that observed in the other two experiments. In the real-valued-weight experiment, the best trial reaches approximately 3,900, while the median and worst trials reach approximately 3,300 and 2,600, respectively. The best trial continues to improve during the later stages of the search, whereas the worst trial exhibits an extended period of stagnation after reaching approximately 2,600. The differences in these trajectories suggest that the weight representation may influence the way ES explores the solution space.

Supplementary Videos[2,3] show gameplay before and after training, respectively, for an MLP with ternary parameters and four hidden units.

8. Conclusion

This study investigated the evolutionary training of neural networks with binary, ternary, and real-valued connection weights using Evolution Strategy. The NNs were trained to control the Atari Galaxian game, with the objective of maximizing the game score. The experimental results were analyzed in terms of ES hyperparameter configurations, connection-weight representations, the number of hidden units, and the characteristics of the learning curves.

First, two ES hyperparameter configurations were compared under the same total number of fitness evaluations. The experimental results demonstrated that configuration (b), which used a larger offspring population and fewer generations, significantly outperformed configuration (a) for all three types of connection weights. These findings suggest that allocating more evaluations to a larger offspring population can improve the optimization performance of ES, potentially by achieving a better balance between local search and global exploration.

Second, the performance of NNs with binary, ternary, and real-valued connection weights was compared using configuration (b). Although no statistically significant differences were found between the real-valued and binary or ternary representations, the observed results generally favored binary and ternary connection weights. This finding is noteworthy because real-valued weights provide a larger search space that includes the solutions available to both binary and ternary weights and may therefore offer greater potential for achieving high performance. Nevertheless, the experimental results suggest that the larger search space associated with real-valued weights may make optimization more difficult for ES. Furthermore, binary and ternary weights offer the practical advantage of reducing the storage requirements of trained NN models.

Third, the effect of the number of hidden units was investigated using NNs with ternary connection weights. The results showed that an NN with four hidden units significantly outperformed one with a single hidden unit, while its performance was competitive with those of NNs containing 8, 16, or 32 hidden units. Thus, four hidden units provided a favorable trade-off between model size and game-playing performance under the experimental conditions. These results also indicate that increasing the number of hidden units does not necessarily improve performance when ES is used for training, possibly because the resulting increase in the dimensionality of the search space makes optimization more difficult.

[2] https://youtu.be/NKJ3G8reTYs ({-1,0,1} weights, 4 hidden units, before the training)
[3] https://youtu.be/7SwQdEV95-8 ({-1,0,1} weights, 4 hidden units, after the training)

Finally, an analysis of the learning curves revealed common characteristics across all three connection-weight representations. Although fitness generally increased as the number of solution evaluations increased, the improvement was not continuous. Instead, periods of stagnation were often followed by substantial improvements, and the timing and magnitude of these improvements varied across independent runs. In particular, some trials achieved substantial improvements during the later stages of training despite relatively limited progress in the early stages. These observations highlight the importance of examining the entire learning process and suggest that premature termination based solely on limited early progress may prevent the discovery of better solutions.

In summary, this study demonstrated that ES could train NNs with binary, ternary, and real-valued connection weights to control Atari Galaxian. Under the experimental conditions, the ES configuration with a larger offspring population achieved significantly better performance than the configuration with more generations, while binary and ternary connection weights generally yielded higher game scores than real-valued weights. Moreover, an NN with four hidden units provided a favorable balance between model size and performance when ternary weights were used. Under the experimental conditions considered in this study, the results suggest that the choice of parameter representation, ES configuration, and network size can affect the performance of evolutionary neural network training.

Future work will investigate whether these findings generalize to other Atari games and different experimental conditions. Further research will also examine alternative ES configurations and search strategies to improve optimization performance, particularly for NNs with real-valued connection weights. Such investigations may help clarify the relationship between the representation of connection weights, the complexity of the search space, and the performance of evolutionary training.

References

[1] H.-P. Schwefel, “Evolution strategies: A family of nonlinear optimization techniques based on imitating some principles of organic evolution,” Annals of Operations Research, vol. 1, pp. 165–167, 1984. doi: 10.1007/BF01876146.

[2] H.-G. Beyer and H.-P. Schwefel, “Evolution strategies—A comprehensive introduction,” Natural Computing, vol. 1, pp. 3–52, 2002. doi: 10.1023/A:1015059928466.

[3] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, “Evolution strategies as a scalable alternative to reinforcement learning,” arXiv:1703.03864, 2017.

[4] M. Courbariaux, Y. Bengio, and J.-P. David, “BinaryConnect: Training deep neural networks with binary weights during propagations,” Advances in Neural Information Processing Systems (NeurIPS), vol. 28, 2015, pp. 3123–3131.

[5] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” Advances in Neural Information Processing Systems (NeurIPS), vol. 29, 2016, pp. 4107–4115.

[6] B. Liu, F. Li, X. Wang, B. Zhang, and J. Yan, “Ternary weight networks,” Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2023, doi: 10.1109/ICASSP49357.2023.10094626.

[7] H. Okada, “Binary neural networks playing Atari Space Invaders (1) trained by evolution strategy,” viXra.org, Aug. 2025. https://vixra.org/pdf/2508.0109v2.pdf.

[8] H. Okada, “Neural networks with {-1,0,1} weights playing Atari Space Invaders (1) trained by evolution strategy,” viXra.org, Feb. 2026. https://vixra.org/pdf/2602.0030v1.pdf.