Binary Neural Networks 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: This study investigates the performance of Genetic Algorithm for optimizing binary neural network controllers in the Atari *Space Invaders* task, extending prior work that applied Evolution Strategy to the same optimization problem. The network topology and the activation function are kept consistent with the earlier study to enable direct comparison between GA and ES. Two GA configurations were utilized while varying the number of hidden units and the bit precision of connection weights. Experimental results revealed that, for the number of hidden units of 1, 2, 4, and 8, the game scores achieved by 1-bit networks were not significantly lower than those of 64-bit networks, consistent with prior ES-based findings. Moreover, even a single hidden unit exhibited competitive performance, unlike in the ES case where performance degraded markedly. GA outperformed ES under the configuration emphasizing the number of generations, while ES performed better under the configuration emphasizing population size; the former difference was statistically significant ($p < .01$). These findings suggest that GA provides a viable alternative to ES for training binary neural network controllers in reinforcement learning tasks.

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

1. Introduction

The author has previously reported experimental results on evolutionary reinforcement learning for binary neural network controllers applied to the Pendulum task [1,2] the Acrobot task [3,4], and the Atari Space Invaders [5]. In the last study [5], Evolution Strategy [6,7] was adopted as the training algorithm. In this study, Genetic Algorithm [8-11] is applied to the same optimization problem in the last study, and the experimental performances are compared between GA and ES. The network topology and the activation function are kept consistent with the prior studies. This study compares two different GA configurations, as well as varying the number of hidden units and the number of bits used per connection weight.

2. Atari Space Invaders

To compare the performance of Genetic Algorithm with that of Evolution Strategy on the same problem, this study employs Atari Space Invaders available in the OpenAI Gym (and Gymnasium[1]) framework. Figure 1 shows an example of the game screen. Details on this task were described in [5]. The same method for scoring fitness of a neural network controller was employed again in this study.

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. Details on the MLP were also described in [5].

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

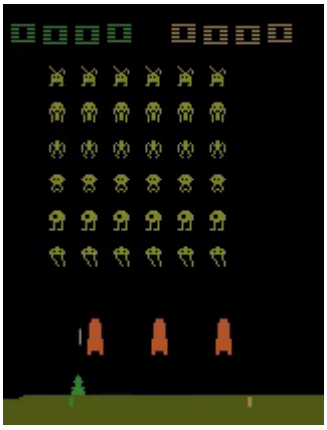


Figure 1. Screenshot of Atari Space Invaders [5].

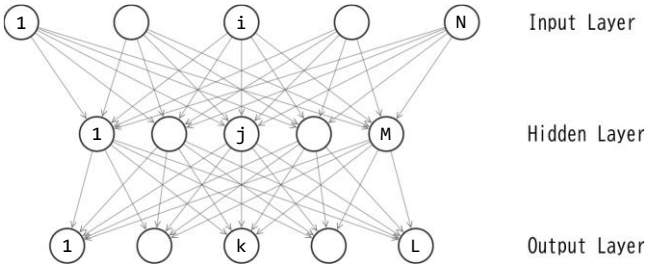


Figure 2. Topology of the MLP [5].

4. Training of Binary Neural Networks 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, 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 Genetic Algorithm [8-11]. 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 phonotype of $\mathbf{x}$. Figure 3 illustrates the GA process. The process is the same as that in the previous studies [2,4]. In Step 1, D-dimensional binary 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 $\mathbf{y}^p$ $(p = 1,2,\ldots,P)$ are applied to the MLP and the MLP plays the game. The fitness of $\mathbf{y}^p$ 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 offsprings 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, $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^E$, 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 $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^E$ 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 offsprings, 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 binary 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 bit-flipped under the mutation probability pm. A greater pm promotes explorative search 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

The neural network employed in this experiment consists of 128 input units, M hidden units (M ∈ {1, 2, 4, 8, 16}) and 6 output units respectively, which is consistent with the network in the previous study [5]. 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].

5.2 Genetic Algorithm

The total number of individuals evaluated in a single run of GA is given by P × 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 both of this study and the previous study [5], two different combinations are consistently configured such that P × 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 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 binary genotype vector.

Table 1. GA hyperparameter configurations.

Hyperparameters	(a)	(b)
Population size (P)	10	50
Generations	100	20
Fitness evaluations	10×100=1,000	50×20=1,000
Number of elites (E)	10×0.2=2	50×0.2=10
Mutation probability (pm)	1/D	1/D

The crossover method for reproducing offsprings in Figure 3 differs depending on whether the 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 the uniform crossover is applied in this study. On the other hand, when weights are represented with 64-bit precision, the genotype is a real-valued vector, and the blend crossover (BLX-α) [12] is applied. α is set to 0.5 in this study.

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 applied to the 20×2 data points presented in Table 2(i). Although no statistically significant difference was observed between configurations (a) and (b), the p-value of 0.14 suggests a tendency for (a) to perform better than (b). In contrast, in the previous experiments using ES, (b) outperformed (a) [5], yielding results opposite to those obtained with GA. This discrepancy is likely attributable to the differing search characteristics of the two algorithms. GA excels at global exploration but tends to be less effective in local search, whereas ES shows the opposite tendency—it performs well in local exploration but is less proficient in global exploration. Meanwhile, configuration (a) promotes local exploitation, whereas configuration (b) encourages global exploration. Therefore, in both GA and ES, the experimental configurations appear to have compensated for the respective weaknesses of each algorithm's search behavior, thereby improving the balance between global and local search and leading to enhanced performance.

Table 2. Fitness scores among 11 runs.

(i) 1bit

	M	Best	Worst	Average	Median
	1	1080	285	840.9	855
	2	1085	565	874.5	870
(a)	**4**	995	730	844.5	800
	8	1045	600	823.2	800
	16	1110	745	857.3	830
	1	945	745	812.7	800
	2	1075	685	828.2	800
(b)	**4**	870	780	810.9	800
	8	900	675	797.7	800
	16	1150	705	807.3	800

(ii) 64bit

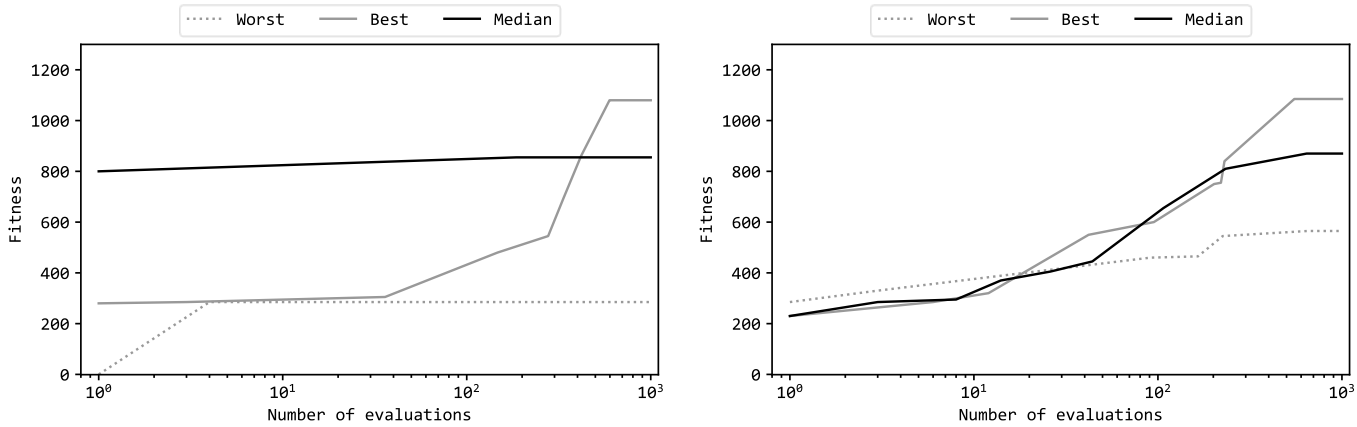
	M	Best	Worst	Average	Median
	1	1110	725	835.9	800
	2	955	720	832.7	845
(a)	**4**	995	765	851.8	820
	8	1055	675	855.5	800
	16	965	745	840.5	835
	1	1000	635	803.2	800
	2	955	800	844.5	800
(b)	**4**	1015	735	843.6	800
	8	1155	770	884.1	855
	16	965	725	845.0	855

The author next 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 (a), resulting in 11 game scores. The Wilcoxon rank sum test was applied to the 11×5 data points. The test revealed that there were no significant differences between

any pair of the two distinct values of M tested ($p > .05$). Even when the hidden layer contained only a single unit, its performance was not significantly inferior to those with 2, 4, 8, or 16 units. In contrast, when using ES, the case of M=1 performed significantly worse than M=2 [5]. A detailed comparison of the performance between GA and ES is presented in Section 6.

The author next 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 $p > .05$). This result is consistent with that in the previous study using ES [5]. These two studies revealed 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. Surprisingly, in this study using GA, the performance of the MLPs with 1-bit weights was found to be better than that with 64-bit weights when the number of hidden units was one or two ($p = .21$ and .16 for M=1 and M=2, respectively).

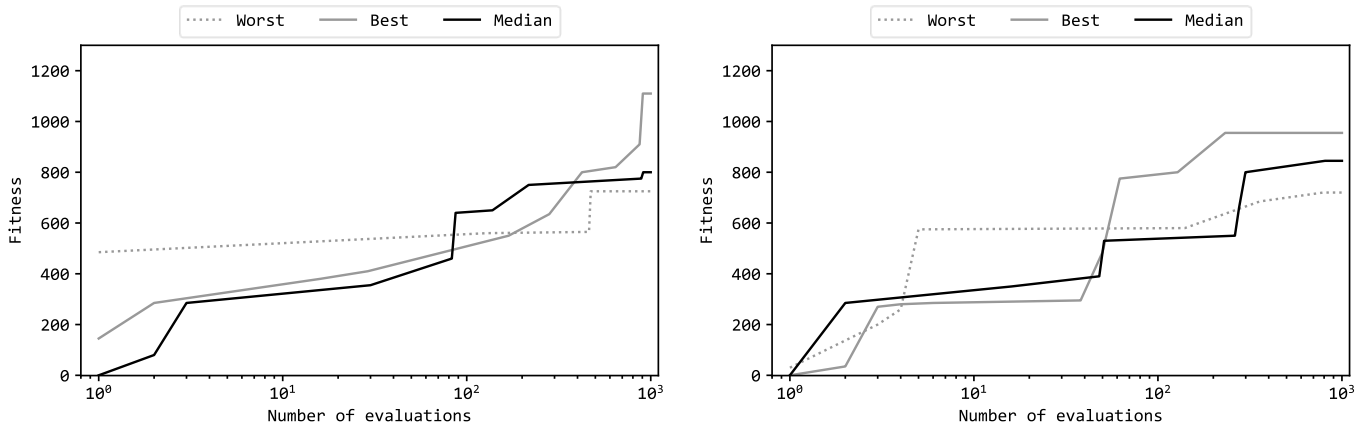
Figures 4 and 5 show the learning curves for the best, median, and worst trials out of the 11 trials conducted under the same condition. The GA configuration used in these experiments corresponds to (a) in Table 1. The number of hidden units M is one for Figure 4 and two for Figure 5. In both figures, (i) and (ii) correspond to the results obtained with 1-bit and 64-bit MLPs, respectively. As shown in Figure 4(ii), the median, best, and worst runs all start with initial scores of approximately 200. Similarly, in Figure 5(ii), the median, best, and worst runs all begin with initial scores of approximately 0. These results indicate that, in the case of 64-bit weights, the performance of randomly initialized MLPs exhibits little variation across the 11 runs. In contrast, Figure 4(i) shows that the initial scores differ substantially among the median, best, and worst runs. In particular, the median run starts with a score of 800 even before training, and its score shows little improvement thereafter. Likewise, in Figure 5(i), the initial scores vary considerably among the three runs. Although the worst run starts with a higher initial score than the best run, the final score of the best run surpasses that of the worst run. These observations suggest that, in the case of 1-bit weights, initial fitness scores vary largely and the solutions with relatively high initial scores may hinder the subsequent evolution of solutions. Since configuration (a) uses a smaller population size than configuration (b), the influence of the initial solutions is presumed to have been more pronounced. Supplementary videos[2,3] are provided which demonstrate the game screens played by the best MLP with 1-bit weights and a single hidden unit.



(i) 1bit (ii) 64bits

Figure 4. Learning curves of MLP with a single hidden unit (M=1).

[2] https://youtu.be/4DBowOmgJEA (before the training)
[3] https://youtu.be/T3JkrzIzET0 (after the training)



(i) 1bit (ii) 64bits

Figure 5. Learning curves of MLP with two hidden units (M=2).

6. Comparison of GA and ES

The game scores of the binary MLP trained using GA are shown in Table 2(i) in this paper, while the corresponding scores obtained using ES were presented in the previous paper [5]. Table 3 is a citation of the previous table.

Table 3. Fitness scores among 11 runs by Evolution Strategy [5].

	M	Best	Worst	Average	Median
	1	980	285	625.0	745
	2	955	285	690.0	805
(a)	**4**	905	480	733.6	780
	8	1075	800	909.1	895
	16	990	565	807.3	785
	1	1065	310	750.5	740
	2	1140	310	843.6	875
(b)	**4**	1245	800	886.4	825
	8	1125	750	878.2	870
	16	1015	765	879.1	875

To test the statistical significance, the Wilcoxon rank sum tests were applied to the data in Tables 2(i) and 3. The tests revealed the followings.

- The scores obtained by GA with for configuration (a) are larger than those by ES, and the difference was statistically significant ($p < .01$).
- The scores obtained by GA with for configuration (b) are smaller than those by ES, but the difference was not statistically significant ($p = .07$).
- The scores obtained by GA with for both of the two configurations are larger than those by ES, but the difference was not statistically significant ($p = .15$).

The test showed that GA generally outperformed ES with the performance difference being particularly pronounced when the configuration prioritizes the number of generations over the population size. However, when the configuration prioritizes the population size over the number of generations, ES performed slightly

better than GA. These findings indicate that making hyperparameter values appropriate to each evolutionary algorithm is crucial for fully exploiting the performance of the algorithm.

7. Conclusion

In this study, Genetic Algorithm was applied to the same reinforcement learning task as in the previous study [5] in which Evolution Strategy had been applied. The task was to optimize binary connection weights of neural network controllers for the Atari Space Invaders. The findings from this study are summarized as follows:

(1) For the number of hidden units M=1, 2, 4, and 8, the game scores achieved by the trained 1-bit MLPs were not significantly smaller than those achieved by the trained 64-bit MLPs. This result is consistent with that in the previous study [5].

(2) Even though the hidden layer contained only a single unit, its performance was not significantly inferior to those with 2, 4, 8, or 16 units. In contrast, when using ES, the case of M=1 performed significantly worse than M=2 [5].

(3) GA performed better than ES with configuration (a), while ES performed better than GA with configuration (b). Configurations (a) and (b) prioritized the number of generations and the population size respectively. The former difference was statistically significant ($p < .01$) but the latter difference was not statistically significant ($p > .05$).

Future directions of this study include comparing GA and ES with other evolutionary algorithms and evaluating performances across other reinforcement learning tasks.

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] Okada, H. (2025). Binary neural networks playing Atari space invaders (1) trained by evolution strategy. viXra.org. https://vixra.org/pdf/2508.0109v2.pdf

[6] 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.

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

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

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

[10] Mitchell, M. (1998). An introduction to genetic algorithms. MIT press.

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

[12] Eshelman, L.J. & Schaffer, J.D. (1993). Real-coded genetic algorithms and interval-schemata, Foundations of Genetic Algorithms, 2, 187–202.