Neural Networks with {-1,0,1} Weights Playing Atari Space Invaders (1) Trained by Evolution Strategy

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

Abstract: In prior work, discrete-weight neural networks trained via evolutionary algorithms have been investigated, demonstrating the feasibility of binary-weight models on reinforcement learning tasks including Atari *Space Invaders*. In this study, we extend this line of research by evaluating ternary-weight neural networks with weights in $\{-1, 0, 1\}$ and comparing their performance with binary-weight counterparts $\{-1, 1\}$. Using Evolution Strategy to train multilayer perceptron controllers for the Atari *Space Invaders* task, the author analyzes the effects of weight representation and evolutionary hyperparameters. Experimental results show that ternary-weight networks achieved higher average performance than binary-weight networks with identical architectures, although the difference was not statistically significant. Additionally, a larger population size combined with fewer generations was found to be more effective than smaller populations with longer training durations, consistent with prior findings. These results suggest that population size plays a critical role in compensating for the limited global search capability of ES.

Keywords: evolutionary algorithm, evolution strategy, 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, the author has 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 [1,2]. In those studies, the connection weights were binary values $\{-1,1\}$. In the present study, the author evaluates the performance of ternary weights $\{-1,0,1\}$ on the same task and compares the results with those obtained using binary weights.

2. Atari Space Invaders

In the experiments presented in this article, as well as in our previous studies [1,2], 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.

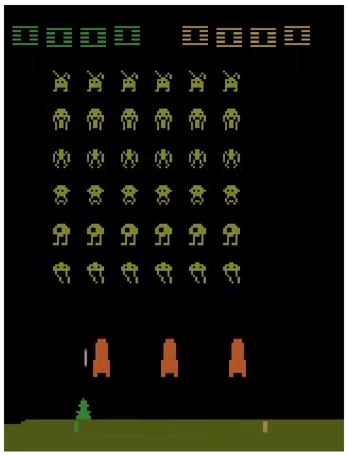

Figure 1. Screenshot of Atari *Space Invaders*.

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 210×160 pixels and three RGB color channels. While the raw images can be used directly, it is common practice to apply preprocessing techniques such as grayscale conversion and downsampling to reduce dimensionality and improve learning efficiency. In contrast, the RAM-based observation mode provides a snapshot of the internal RAM state of the Atari 2600. Each observation is a one-dimensional array of length 128, with each element represented as an 8-bit unsigned integer (uint8) ranging from 0 to 255. This array contains compact, low-level information about the game state, such as score, enemy positions, and projectile coordinates, directly encoded in memory. In this study, the author employs the RAM-based environment, `ALE/SpaceInvaders-ram-v5` 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 [1,2]. 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 perceptrons (MLP), as the controller to the game. Figure 2 illustrates the topology of the network. A single hidden layer is included, and the connection weights are 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 [3-6].

The MLP serves as the policy function: action(t) = F(observation(t)). Since the author utilized the `ALE/SpaceInvaders-ram-v5` environment, each observation yields 128 numerical values. To use these values as input to the neural network, the input layer is configured with 128 units (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.

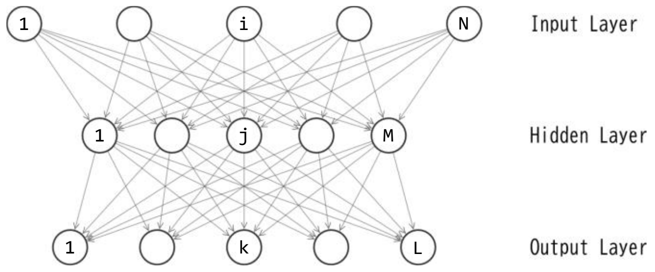

Figure 2. Topology of the MLP.

4. Training of Neural Networks with Ternary Connection Weights by Evolution Strategy

The three-layered perceptron, as depicted in Figure 2, includes M+L unit biases and NM+ML connection weights, resulting in a total of M+L+NM+ML parameters. Let D represent the quantity M+L+NM+ML. For this study, the author sets N=128 and L=6, leading to D=135M+6. The training of this perceptron is essentially an optimization of the D-dimensional binary vector. Let $\mathbf{x} = (\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_\mathrm{D})$ 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, $\mathrm{x}_\mathrm{i} \in \{-1,0,1\}$. 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 binary vector $\mathbf{x}$ is optimized using Evolution Strategy [7,8]. ES treats $\mathbf{x}$ as a chromosome (a genotype vector) and applies evolutionary operators to manipulate it. The fitness of $\mathbf{x}$ is the game score played by the MLP as the phonotype of $\mathbf{x}$. Figure 3 illustrates the ES process. The process is the same as that in the previous studies [1,3,5]. In Step 1, D-dimensional binary vectors $\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^C$ are randomly initialized where C denotes the number of offsprings. A larger value of C promotes exploration more. In Step 2, values in each vector $\boldsymbol{y}^c$ $(c = 1,2,\ldots,C)$ are applied to the MLP and the MLP play a game. The fitness of $\boldsymbol{y}^c$ is then evaluated with the game score. Let $f(\boldsymbol{y}^c)$ denote the fitness. In Step 3, the loop of evolutionary training is finished if a preset condition is satisfied. A simple example of the condition is the limit number of fitness evaluations. In Step 4, among the $P + C$ vectors in the parent population $(\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^P)$ and the offspring population $(\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^C)$, vectors with the top P fitness scores survive as the parents in the next reproduction, and the remaining vectors are deleted. P denotes the number of parents. A smaller value of P promotes exploitation more. Note that, for the first time of Step 4, the parent population is empty so that vectors with the top P fitness scores survive among the C vectors in the offspring population $(\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^C)$. In Step 5, new C offspring vectors are produced by applying the reproduction operator to the parent vectors $(\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^P)$ which are selected in the last Step 4. The new offspring vectors form the new offspring population $(\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^C)$. Figure 4 denotes the process of reproduction. In Step5-4, each of $y_1^c, y_2^c, \ldots, y_D^c$ is mutated under the probability pm. A greater value of pm promotes exploration more.

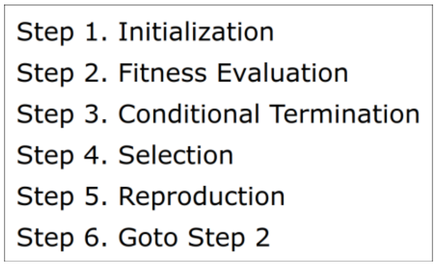
Step 1. Initialization
Step 2. Fitness Evaluation
Step 3. Conditional Termination
Step 4. Selection
Step 5. Reproduction
Step 6. Goto Step 2

Figure 3. Process of Evolution Strategy.

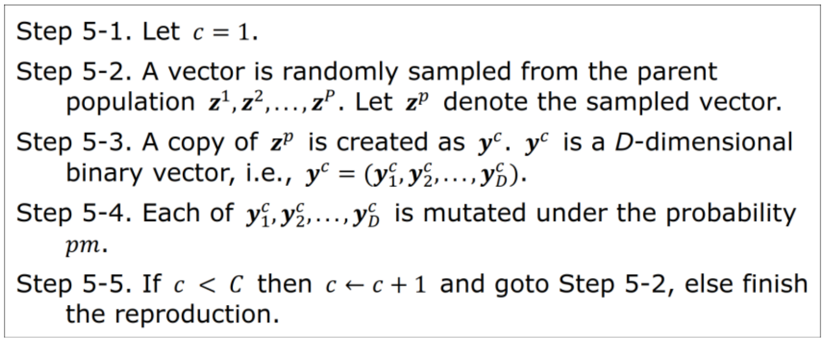
Step 5-1. Let $c = 1$.
Step 5-2. A vector is randomly sampled from the parent population $\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^P$. Let $\boldsymbol{z}^p$ denote the sampled vector.
Step 5-3. A copy of $\boldsymbol{z}^p$ is created as $\boldsymbol{y}^c$. $\boldsymbol{y}^c$ is a D-dimensional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

Figure 4. Reproduction process in Evolution Strategy.

5. Experiment

5.1 Neural Network

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

5.2 Evolution Strategy

The setup of ES is largely identical to those used in our previously reported experiments [1]; the primary difference lies in the number of generations. In our previous studies, the number of generations was 100 (20) when the population size was 10 (50). In contrast, in the experiments reported in this paper, these values are increased by a factor of ten, to 1,000 (200) when the population size is 10 (50). 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 set to 1%.

Table 1. ES hyperparameter configurations.

Hyperparameters	**(a)**	**(b)**
Number of offsprings (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)	0.01	0.01

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 no statistically significant difference was observed between configurations (a) and (b); however, the p-value of 0.06 indicated a trend in favor of configuration (b) over configuration (a). 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 revealed that configuration (b) outperformed configuration (a) with statistical significance (p=5.5e-3). Thus, for both ternary and binary weight representations, configuration (b) was found to outperform configuration (a). This result is consistent with our previous experiment [1]. Although evolutionary strategies (ES) are effective at local search, they are less suited to global exploration. Increasing the population size promotes global exploration; therefore, compared with configuration (a), configuration (b) is considered to better compensate for ES's limited global search capability, resulting in a more favorable balance between global and local search.

Next, the author compares 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 (a) was adopted; however, the p-value of 0.10 indicated a trend in favor of the $\{-1,0,1\}$ weights over the $\{-1,1\}$ weights. 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 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=0.55). 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.

Table 2. Fitness scores among 11 runs.

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

	M	Best	Worst	Average	Median
(a)	**1**	1310	285	841.4	985
	2	1310	800	1018.6	1005
	4	1570	690	1095.9	1125
	8	1410	920	1145.5	1095
	16	1355	930	1075.0	1030
	32	1210	910	1051.4	1085
(b)	**1**	1420	800	1009.1	950
	2	1515	935	1149.1	1100
	4	1425	930	1132.3	1105
	8	1285	945	1095.9	1055
	16	1255	975	1073.2	1035
	32	1250	950	1069.1	1045

(ii) {-1,1} weights

	M	Best	Worst	Average	Median
(a)	**1**	1185	285	835.9	885
	2	1310	765	1017.7	1045
	4	1420	945	1074.5	1030
	8	1830	565	1193.6	1120
	16	1230	885	1036.8	1020
	32	1230	930	1053.2	1045
(b)	**1**	1225	745	950.9	950
	2	1735	810	1070.5	980
	4	1425	905	1113.2	1055
	8	1385	915	1126.4	1155
	16	1615	995	1127.3	1035
	32	1230	1000	1108.2	1110

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 ES configuration is (b) in Table 1. The number of hidden units is M=2 for Figure 5 and M=16 for Figure 6 (the average scores were the largest for these numbers of hidden units). Note that the horizontal axis represents the number of evaluations on a logarithmic scale in these figures. As the number of evaluations increases, the solution improvement becomes more gradual. Moreover, in both figures, there are regions in which the fitness score remained stagnant for a certain numbers of evaluations and then increased again. Neural network training based on evolutionary algorithms does not exploit the gradient of the objective function, and the connection weights are updated randomly. Prolonged periods of stagnation in improvement indicate that random modifications of connection weights are unlikely to succeed, resulting in low learning efficiency. Application of more efficient learning algorithms therefore remains a direction for future work.

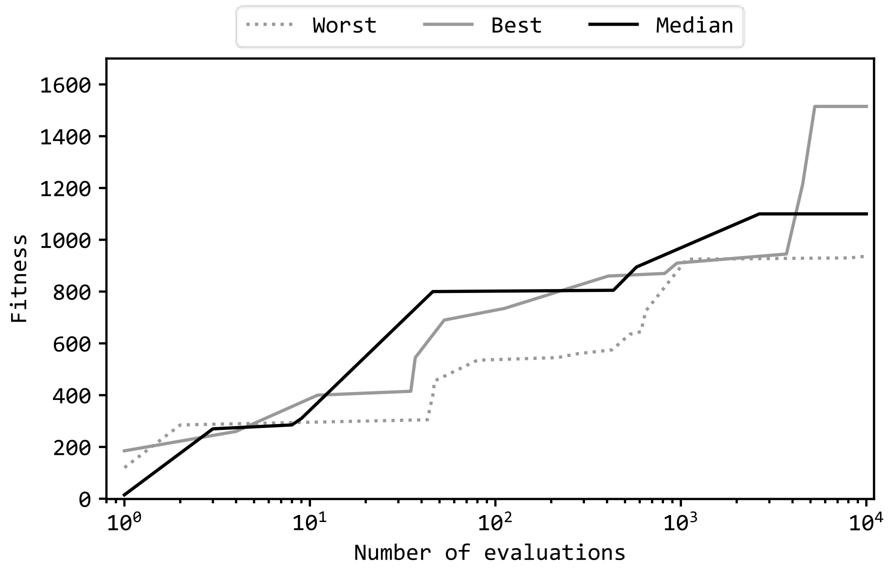

Figure 5. Learning curves of MLP with {-1,0,1} weights and two hidden units.

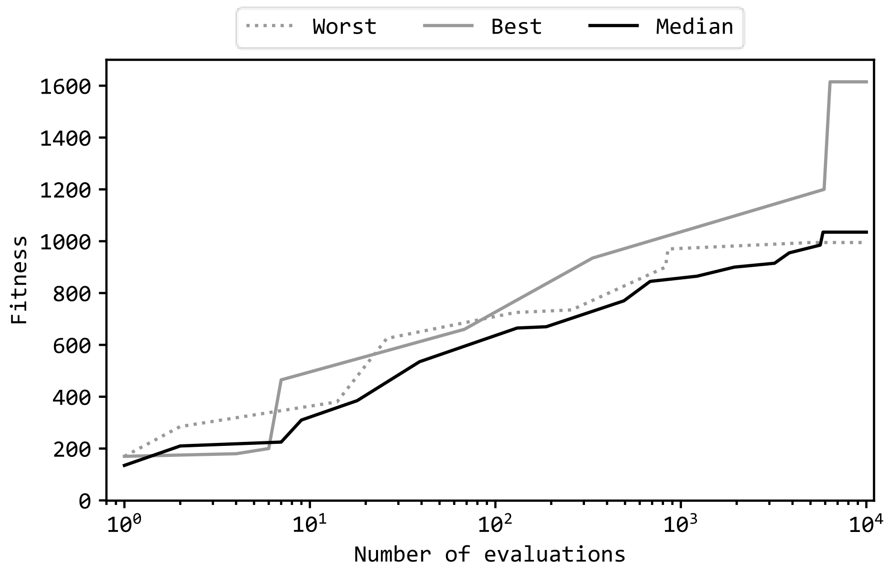

Figure 6. Learning curves of MLP with {-1,1} weights and 16 hidden units.

Supplementary videos[2-5] are provided which demonstrate the game screens played by the trained/untrained MLPs.

6. Conclusion

In this study, Evolution Strategy was applied to the reinforcement learning of neural network controllers for the Atari Space Invaders, where the connection weights in the neural network are ternary $\{-1,0,1\}$ or binary $\{-1,1\}$. The findings from this study are summarized as follows:

(1) The trained ternary MLPs achieved higher performance on the game than the trained binary MLPs with the same topologies; however, this performance difference was not statistically significant.

(2) A larger population (50) and a smaller number of generations (200) contributed more for ES to train the MLPs with ternary/binary connection weights than a smaller number of generations (10) and a larger number of generations (1,000). This result was consistent with our previous study [1]. The reason will be because a larger population compensates for ES's limited global search capability, resulting in a more favorable balance between global and local search.

Although no statistically significant difference in performance was observed between ternary-weight and binary-weight models, this result may be task-dependent and may also depend on the learning algorithm employed. Future work will therefore include comparisons using tasks other than Atari Space Invaders and learning algorithms other than ES.

[2] https://youtu.be/A60KQ17IAXY ({-1,0,1} weights, 2 hidden units, before the training)
[3] https://youtu.be/ANXQMiOFiis ({-1,0,1} weights, 2 hidden units, after the training)
[4] https://youtu.be/0y1n3QyzFao ({-1,1} weights, 16 hidden units, before the training)
[5] https://youtu.be/ZEz1Ft1FZjY ({-1,1} weights, 16 hidden units, after the training)

References

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

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

[3] 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

[4] 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

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

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

[7] 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.

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