Journal of Engineering and Technology for Industrial Applications

ITEGAM-JETIA

Manaus, v.9 n.42, p. 11-15. Jul/Aug, 2023

DOI: https://doi.org/10.5935/jetia.v9i42.867

ISSN ONLINE: 2447-0228

RESEARCH ARTICLE OPEN ACCESS

AN EVALUATIVE ANALYSIS OF PARTICLE SWARM OPTIMIZATION FOR REINFORCEMENT LEARNING IN PENDULUM TASK

Hidehiko Okada*[1]

[1] Department of Intelligent Systems, Kyoto Sangyo University, Kyoto, Japan.

[1] http://orcid.org/0000-0001-7092-4979

Email: *hidehiko@cc.kyoto-su.ac.jp

ARTICLE INFO

Article History
Received: July 01th, 2023
Revised: August 20th, 2023
Accepted: August 28th, 2023
Published: August 31th, 2023

Keywords:
Particle swarm optimization,
Swarm intelligence,
Metaheuristics,
Neural network,
Reinforcement learning.

ABSTRACT

Applying swarm intelligence algorithms to reinforcement learning of neural networks is practical because they do not rely on gradients. Particle swarm optimization (PSO) is a representatives of swarm algorithms. In this paper, the author experimentally evaluates the effectiveness of PSO in the reinforcement learning of multilayer perceptrons (MLPs), using a pendulum control task. Experimental results demonstrated the successful training of an MLP with 8 hidden units, enabling rapid uprighting of the pendulum. Notably, it was found that increasing the population size rather than the number of iterations allowed PSO to discover better solutions. In PSO, increasing the population size promotes global exploration in the early stages, while increasing the number of iterations enhances local exploitation in the later stages. Based on the results of this experiment, it is evident that in this learning task, early-stage global exploration is more important.

I. INTRODUCTION

In supervised learning, neural networks can be optimized using gradient-based methods with labeled training data. This involves computing the difference between neural network's outputs and their respective target values, and then adjusting connection weights and unit biases through backpropagation of errors. However, reinforcement learning tasks require the use of gradient-free training algorithms since labeled training data are not available. Applying swarm intelligence algorithms [1,2] to reinforcement learning of neural networks is practical because they do not rely on gradients. On the other hand, Q-learning [3,4] is a popular reinforcement learning method that selects subsequent actions based on the reward r(t) for action a(t) in state s(t) at each time step t. Unlike Q-learning, swarm algorithms do not require the calculation of r(t) at every step, but instead, evaluate the reward after the completion of an episode. This feature of swarm algorithms relieves the practitioner from the burden of designing appropriate rewards for every combination of states and actions.

Particle swarm optimization (PSO) [5,6], ant colony optimization (ACO) [7,8], and artificial bee colony (ABC) [9,10] are representative swarm algorithms. However, to effectively use these algorithms for training neural networks, it is essential to select appropriate variations and design their hyperparameters carefully, as they have a significant impact on performance. In this paper, the author experimentally evaluates the effectiveness of PSO in the reinforcement learning of multilayer perceptrons, using a pendulum control task.

II. PENDULUM TASK

As a reinforcement learning task, this study utilizes the pendulum task available in the OpenAI Gym[1,2]. The goal of the task is to maintain the pendulum in an upright position by applying torque. The system is depicted in Fig. 1, which shows a screenshot of the task, with the round arrow indicating the direction and magnitude of the torque applied by the controller. The study aims to provide insights into the performance of PSO on this task.

1. https://www.gymlibrary.dev/environments/classic_control/pendulum/
2. https://github.com/openai/gym/blob/master/gym/envs/classic_control/pendulum.py

The author modified the system such that the task starts with the pendulum in a position opposite to the desired outcome, as shown in Fig. 2(a). The objective is to manipulate the pendulum to reach and maintain the state depicted in Fig. 2(b). Additionally, the author adjusts the system to begin the control task with zero angular velocity for the pendulum.

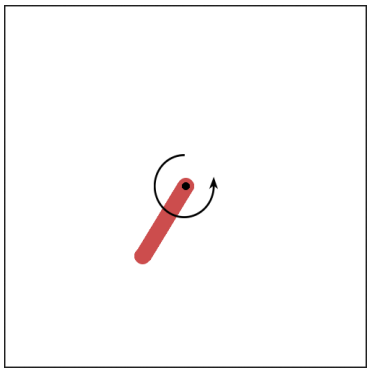


Figure 1: Pendulum system.
Source: OpenAI Gym, (2023).
https://www.gymlibrary.dev/_images/pendulum.gif

Figure 2: Initial and goal states.
Source: Author, (2023).

An episode in the simulation consists of 200 time steps, during which the controller observes the current state and determines the corresponding action. The state is characterized by three values: the cosine and sine of the angle (θ), and the angular velocity, which are within the ranges of -1.0 to 1.0 and -8.0 to 8.0, respectively. The action taken by the controller is the torque applied to the pendulum, within the range of -2.0 to 2.0. The constant torque of 2.0 (or -2.0) is not sufficient to bring the pendulum from its initial position to the goal position: the controller must actively swing the pendulum, leveraging gravity to increase the angular velocity and allow it to overcome the obstacle.

In this study, the author defines the fitness of a controller as follows:

$$\text{Fitness} = \frac{1}{200}\sum_{t=1}^{200}(1 - \text{Error}(t)), \tag{1}$$

$$\text{Error}(t) = \frac{|\theta(t)|}{\pi}. \tag{2}$$

θ(t) denotes the angular at time step t. Initially, the error is calculated as Error(t)=|±π|/π=1, indicating that the pendulum is in a position opposite to the desired goal state. As the pendulum moves towards the goal state, the error decreases. At the goal state, the error is 0/π=0, indicating that the pendulum is upright.

The fitness score rewards the controller more for achieving the desired goal state more quickly and maintaining it longer, i.e., a higher fitness score is obtained when the error is lower for more time steps.

III. MULTILAYER PERCEPTRON

This study adopts a multilayer perceptron (MLP) [11] as the pendulum controller. The MLP is a three-layered feedforward neural network. The topology is illustrated in Fig. 3, while the feedforward computations are shown in (3)-(7).

Input layer:

$$out_i^{(1)} = x_i, i = 1,2,\dots,N \tag{3}$$

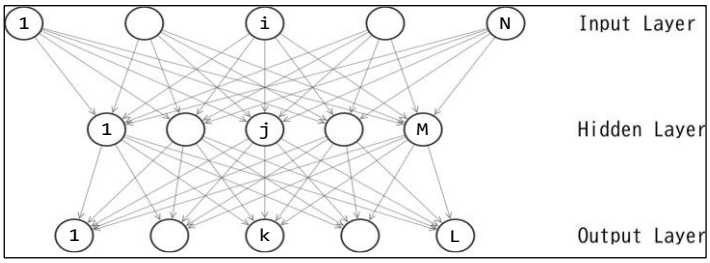


Figure 3: Topology of the MLP.
Source: Author, (2023).

Hidden layer:

$$in_j^{(2)} = \theta_j^{(2)} + \sum_i w_{i,j}^{(2)}\, out_i^{(1)}, j = 1,2,\dots,M \tag{4}$$

$$out_j^{(2)} = h(in_j^{(2)}), j = 1,2,\dots,M \tag{5}$$

Output layer:

$$in_k^{(3)} = \theta_k^{(3)} + \sum_j w_{j,k}^{(3)}\, out_j^{(2)}, k = 1,2,\dots,L \tag{6}$$

$$out_k^{(3)} = h(in_k^{(3)}), k = 1,2,\dots,L \tag{7}$$

The activation function denoted as h() is the hyperbolic tangent function whose shape is illustrated in Fig. 4. This activation function is a widely used in neural networks due to its ability to produce a smooth non-linear output that ranges from -1.0 to 1.0.

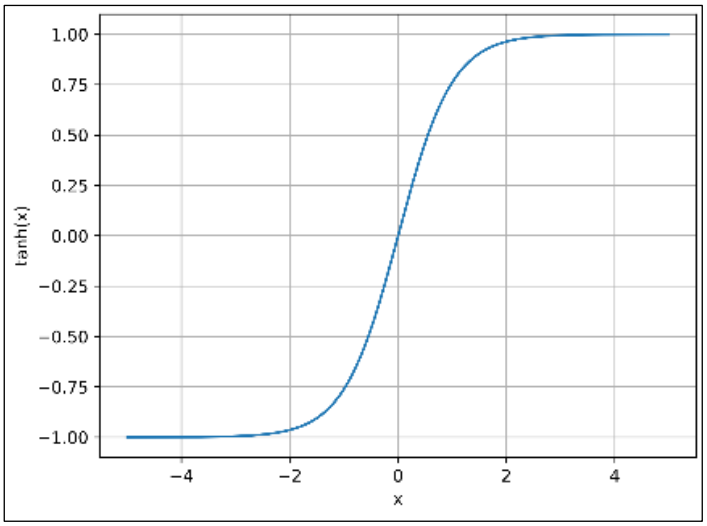


Figure 4: Hyperbolic tangent function.
Source: Author, (2023).

The MLP plays the role of a policy function where the action at time t is a function of the observation at time t, i.e., action(t) = F(observation(t)). The input layer of the MLP comprises three units that receive the values of cos(θ), sin(θ), and the angular velocity. To ensure that the input values are within the range of [-1.0, 1.0], the angular velocity is normalized by dividing it by 8.0. The output layer of the MLP consists of one unit, which outputs the torque applied to the pendulum. To ensure that the torque falls within the range of [-2.0, 2.0], the output value is scaled by multiplying it by 2.0.

IV. TRAINING OF MLPS USING PSO

The MLP illustrated in Fig. 3 comprises M + L units and NM + ML connections, giving a total of D = M + L + NM + ML parameters. To train the MLP, the author formulates the problem as the optimization of a D-dimensional real-valued vector, $\boldsymbol{x} = (x_1, x_2, \ldots, x_{\mathrm{D}})$, where each x_i corresponds to one of the D parameters in the MLP. The feedforward computation, as described in (3)-(7), involves applying the values of $\boldsymbol{x}$ to their corresponding connection weights or unit biases.

In this study, PSO is applied to optimize the D-dimensional vector $\boldsymbol{x}$. PSO represents one of the swarm intelligence algorithms, which are characterized by being population-based stochastic search algorithms. PSO utilize $\boldsymbol{x}$ as a particle position in the D-dimensional search space. Fig. 5 shows the process of training neural networks by PSO.

Step1: Initialization
Step2: Evaluation
Step3: Conditional Termination
Step4: Updates of Pbests and Gbest
Step5: Updates of Particle Velocities
Step6: Updates of Particle Positions
Step7: Goto Step2

Figure 5: The process of particle swarm optimization.
Source: Author, (2023).

In Step 1, vectors $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^{\mathrm{S}}$ are initialized randomly, where S represents the swarm size (the number of particles in the swarm). $\boldsymbol{x}^s$ denotes the position vector of the s-th particle in the D-dimensional search space, i.e., $\boldsymbol{x}^s = (x_1^s, x_2^s, \ldots, x_{\mathrm{D}}^s), s = 1,2, \ldots, \mathrm{S}$. The swarm size is predetermined. In Step 2, the fitness of each particle is evaluated using (1). In Step 3, the loop of the swarm process is terminated when a specific termination condition is satisfied. In this study, the loop is terminated when the loop counter reaches a predetermined value. In Step 4, the personal best (Pbest) of each particle and the global best (Gbest) in the swarm are updated according to their fitness scores. The Pbest of a particle represents the position vector that has achieved the highest fitness score up to the current iteration for that specific particle. On the other hand, the Gbest represents the position vector with the highest fitness score among all the Pbests within the population. Let us denote each Pbest as $\boldsymbol{p}^s$ and the Gbest as $\boldsymbol{g}$, respectively. In Step 5, the velocity of each particle is updated. Let $v^s = (v_1^s, v_1^s, \ldots, v_{\mathrm{D}}^s)$ represents the velocity for the s-th particle. The velocity v^s is updated by (8).

$$\boldsymbol{v}^s \leftarrow w\boldsymbol{v}^s + c_p r_p(\boldsymbol{p}^s - \boldsymbol{x}^s) + c_g r_g(\boldsymbol{g} - \boldsymbol{x}^s) \quad (8)$$

w denotes the inertia weight, while c_p and c_g are coefficients. Additionally, r_p and r_g are uniformly distributed random values within the interval [0,1]. In Step 6, each particle moves in the search space according to its velocity. The position vector $\boldsymbol{x}^s$ is updated by (9).

$$\boldsymbol{x}^s \leftarrow \boldsymbol{x}^s + \boldsymbol{v}^s \quad (9)$$

V. EXPERIMENT

The MLP's capability to model nonlinear functions is influenced by the number of hidden units. Optimizing a smaller MLP using swarm algorithms is facilitated by a reduced number of variables (the shorter length of position vector $\boldsymbol{x}$). However, this reduction in hidden units may impede the MLP's capability to effectively control the pendulum. Conversely, a larger MLP is more capable of successfully controlling the pendulum, but optimizing it becomes more challenging due to the longer position vector $\boldsymbol{x}$. Moreover, implementing a larger MLP requires additional computational resources. Therefore, striking a balance between these trade-offs is essential for determining the optimal number of hidden units for the given task. This study explores three different configurations of hidden units: 8, 16, and 32. PSO hyperparameter values were determined through empirical analyses, as illustrated in Table 1. The number of iterations was set to 500 or 100, corresponding to swarm sizes of 10 and 50, respectively. Consequently, the total number of fitness evaluations remained constant at 50,000 (equal to the product of iteration and swarm size). It is important to choose an appropriate search space because the values in $\boldsymbol{x}^s$ are utilized as connection weights or unit biases in the neural network. The range should neither be excessively large nor small. In this experiment, the search space is $[-10.0, 10.0]^{\mathrm{D}}$. The position vectors $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^{\mathrm{S}}$ are randomly initialized within the space, and the velocities $\boldsymbol{v}^1, \boldsymbol{v}^2, \ldots, \boldsymbol{v}^{\mathrm{S}}$ are initially zero vectors.

Table 1: PSO hyperparameters.

Hyperparameter	**(a)**	**(b)**
Swarm size	10	50
Iteration	500	100
Fitness Evaluations	50,000	50,000
Inertia weight w	0.9	0.9
Pbest coefficient c_p	1.0	1.0
Gbest coefficient c_g	1.0	1.0

Source: Author, (2023).

An MLP with 8, 16, or 32 hidden units was trained 11 times independently. Table 2 displays the best, worst, average, and median fitness scores achieved by the trained MLPs among the 11 trials. Each of the two hyperparameter configurations (a) and (b) was applied.

Table 2: Fitness Scores among 11 Runs.

		M=8	**M=16**	**M=32**
(a)	**Best**	0.810	0.831	0.832
	Worst	0.571	0.565	0.577
	Average	0.675	0.727	0.702
	Median	0.641	0.811	0.687
(b)	**Best**	0.832	0.833	0.833
	Worst	0.807	0.623	0.583
	Average	0.823	0.789	0.783
	Median	0.825	0.827	0.830

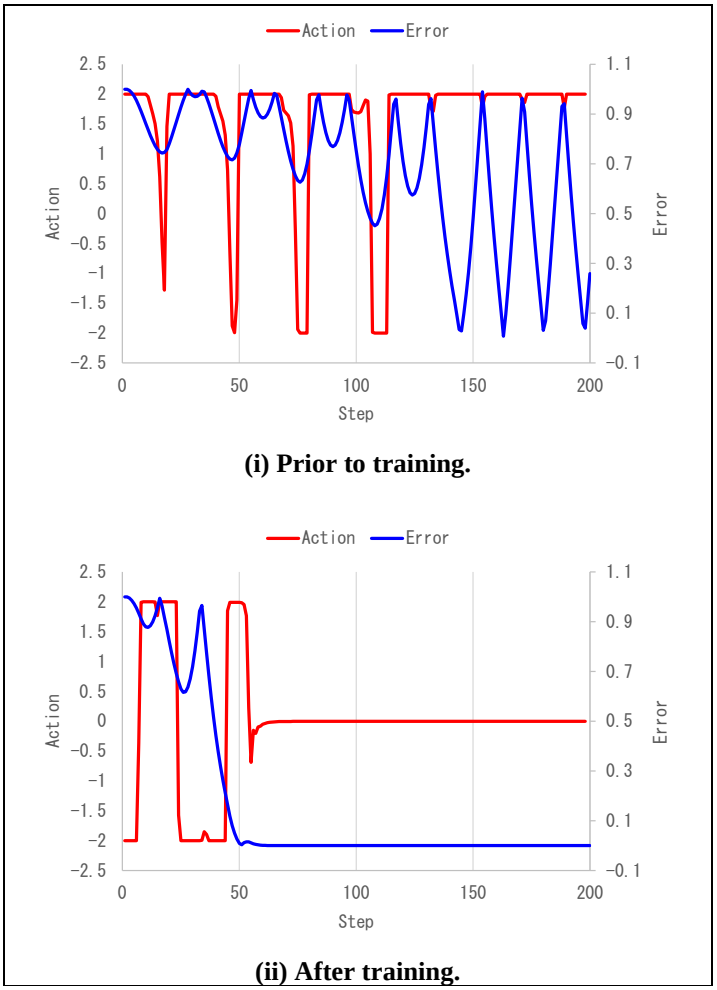
Source: Author, (2023).

Comparing the scores in Table 2 between configurations (a) and (b), it is observed that the values obtained using configuration (b) are higher than those obtained using configuration (a). This result indicates that configuration (b) is better than configuration (a). Wilcoxon signed rank test revealed that this difference is statistically significant (p=1.52e-5). Therefore, in this study, it is evident that increasing the swarm size rather than the number of iterations allowed PSO to discover better solutions. In PSO, increasing the swarm size promotes global exploration in the early stages, while increasing the number of iterations enhances local exploitation in the later stages. Based on the results of this experiment, it is evident that in this learning task, early-stage global exploration is more important.

Next, comparing the fitness scores obtained using configuration (a) among the three variations of M (the number of hidden units), it is observed that even for the smallest size, M=8, the scores are not inferior to those of M=16 or M=32. In fact, the average and worst values across the 11 trials indicate that M=8 is the most desirable. Increasing hidden units would typically enhance the MLP's nonlinear modeling capability and improve the performance of pendulum control. However, it can be seen that increasing hidden units to 16 and 32 does not improve the control performance and instead leads to a decrease in the learning performance through PSO. Wilcoxon rank sum test revealed that the difference between M=8 and M=16 (or 32) is not statistically significant (p=0.55 and p=0.42 respectively). As the number of hidden units increases, the dimensionality of the search space also increases, resulting in the increased difficulty in global exploration. Therefore, in the learning task of this study, the swarm size of 50 particles is sufficient for M=8 but insufficient for M=16 and M=32, indicating that the global exploration was not adequate in those cases.

Fig. 6 presents the learning curves of the best, median, and worst runs among the 11 trials, where M=8 and the configuration is (b). These learning curves indicate a slower progression of fitness scores within the ranges of [0.4, 0.5] and [0.6, 0.7]. Consequently, attaining a fitness score of 0.4 is relatively straightforward for PSO in training MLPs, while challenges arise in achieving higher scores for improved pendulum control. Remarkably, even in the most unfavorable trial out of the 11 conducted, PSO successfully trained the MLPs to reach a score of 0.807 (as shown in Table 2), demonstrating the robustness of PSO in effectively discovering desirable solutions.

Figure 6: Learning curves.
Source: Author, (2023).



(i) Prior to training.

(ii) After training.

Figure 7: Actions and errors by the MLP.
Source: Author, (2023).

Fig. 7(i) illustrates the actions and errors of the MLP in the 200 steps prior to training, while Fig. 7(ii) displays the corresponding actions and errors after training. In this scenario, the MLP employed 8 hidden units, and the configuration (b) was utilized. Fig. 7(i) reveals that the MLP prior to training outputs significant variations in torque values ranging from 2.0 to -2.0 during the initial and mid-stages of the 200 steps, indicating an attempt to lift the pendulum. However, from the mid-stage to the end, the torque value remains approximately constant at 2.0, leading to pendulum rotation and substantial fluctuations in error. In contrast, Fig. 7(ii) reveals that the MLP after training successfully switches the polarity of torque appropriately within the first 50 steps, lifting the pendulum upward and rapidly reducing the error to nearly zero. Furthermore, it maintains the pendulum in the upward position with zero error by setting the torque value to 0 for the remaining steps, exerting no unnecessary force on the pendulum. Supplementary videos are provided which demonstrate the pendulum controlled by the MLPs[3,4].

VI. CONCLUSIONS

In this study, the neural network controller for the pendulum task was trained using Particle Swarm Optimization. The results demonstrated the successful training of an MLP with 8 hidden units, enabling rapid uprighting of the pendulum. Notably, it was found that a larger swarm size yielded greater effectiveness compared to increasing the number of iterations. In future work,

3. https://youtu.be/g7sqUhZnru4
4. https://youtu.be/o49by3OwT58

the author plans to evaluate additional evolutionary/swarm algorithms by implementing them on the same task as conducted in this study.

VII. AUTHOR'S CONTRIBUTION

Conceptualization: Hidehiko Okada.
Methodology: Hidehiko Okada.
Investigation: Hidehiko Okada.
Discussion of results: Hidehiko Okada.
Writing – Original Draft: Hidehiko Okada.
Writing – Review and Editing: Hidehiko Okada.
Resources: Hidehiko Okada.
Supervision: Hidehiko Okada.
Approval of the final text: Hidehiko Okada.

VIII. ACKNOWLEDGMENTS

The author conducted this study as an official researcher of Kyoto Sangyo University.

IX. REFERENCES

[1] R. C. Eberhart, Y. Shi, and J. Kennedy, *Swarm Intelligence*, Morgan Kaufmann, 2001.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford Academic, 2020.

[3] C.J.C.H. Watkins, and P. Dayan, "Q-learning," Machine Learning, vol. 8, no. 3, pp. 279-292, 1992.

[4] R.S. Sutton, and A.G. Barto, *Reinforcement Learning: An Introduction (2nd ed.)*, MIT Press, 2018.

[5] J. Kennedy, and R. Eberhart, "Particle swarm optimization," IEEE Int. Conf. on Neural Networks, vol. iv, pp. 1942-1948, 1995.

[6] P. Riccardo, J. Kennedy, and T. Blackwell, "Particle swarm optimization: an overview," Swarm Intelligence, vol. 1, pp. 33-57, 2007.

[7] M. Dorigo, and T. Stützle, *Ant Colony Optimization*, MIT Press, 2004.

[8] M. Dorigo, M. Birattari, and T. Stützle, "Ant colony optimization," IEEE Computational Intelligence Magazine, vol. 1, no. 4, pp. 28-39, 2006.

[9] D. Karaboga, and B. Basturk, "Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems," Advances in Soft Computing: Foundations of Fuzzy Logic and Soft Computing, LNCS, vol. 4529, pp. 789-798, Springer, 2007.

[10] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, "A comprehensive survey: artificial bee colony (ABC) algorithm and applications," Artificial Intelligence Review, vol. 42, pp. 21–57, 2014.

[11] P. K. Sankar, and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," IEEE Trans. on Neural Networks, vol. 3, no. 5, pp. 683-697, 1992.