ICIIBMS 2020. Track 2. Artificial Intelligent, Robotics, and Human-Computer Interaction, Okinawa, Japan, Nov.18-20, 2020

Neural architecture search and weight adjustment by means of Ant Colony Optimization

Eito Suda

Graduated School of Information Science and Technology
the University of Tokyo
Tokyo, Japan

Hitoshi Iba

Graduated School of Information Science and Technology
the University of Tokyo
Tokyo, Japan

2020 5th International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS) | 978-1-7281-8022-9/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICIIBMS50712.2020.9336395

***Abstract*–In recent years, many different areas of research have utilized neural networks (NNs). Many have investigated weights in NNs as well as structural optimization of NNs via neural architecture search. In this paper, we apply weight training during neural architecture search by Ant Colony Optimization(ACO) to two problems from OpenAI Gym and one problem from pybullet-gym controlled by NNs. We also compare the timing of when the weight training is performed, which is before the architecture search, during architecture search and after architecture search. It was found that performing architecture search by ACO and weight training simultaneously is effective for increasing the score of NNs and that by performing weight training before or at the same time as architecture search, the score was increased statistically significantly for all problems compared with fully-connected NN and the score by performing weight training after architecture search was increased statistically significantly for only one problem from pybullet-gym.**

Keywords–NeuroEvolution, Ant Colony Optimization, Swarm Intelligence, Metaheuristics, Evolutionary Computation

1 Introduction

In recent years, many different areas of research have utilized neural networks (NNs). NNs achieve high levels of accuracy through weight-training methods such as backpropagation[1] and have yielded outstanding results in areas of research such as forecasting and control. In addition, many have investigated weights in NNs as well as structural optimization of NNs via neural architecture search. One representative example of this is research into NeuroEvolution of Augmenting Topologies (NEAT)[2], which showed that NN structure optimization is effective for increasing performance. Although NEAT and Hyper-NEAT[3] studied the structure optimization of NN via evolutionary computing, metaheuristic methods have also been used [4][5][6][7][8][9]. The present research also attempts to perform NN architecture search by using ant colony optimization (ACO)[10], which is one such metaheuristic method.

978-1-7281-8022-9/20/$31.00 ©2020 IEEE

Since the usefulness of NN structure optimization was demonstrated through NEAT[2] research, a wide variety of research has been conducted on neuroevolution, and structure optimization using metaheuristics has also been investigated, as noted above. Although neuroevolution using ant colony optimization has been studied for recurrent neural networks (RNN)[5], Long short-term memory(LSTM)[6] and Convolutional Neural Network(CNN) [7], including the selection of inputs for the NN[11], there has been little research on using ant colony optimization for architecture searches for other types of NNs. Therefore, the aim of this research was to increase the accuracy of the NN and demonstrate the effectiveness of performing weight training and evolution simultaneously during ant colony optimization of an architectural search on a pre-trained NN. Another aim was to clarify the optimal ordering of the architecture search and weight training when solving the game OpenAI Gym[12] and pybullet-gym [13] by using an NN, by comparing the timing when the weight training is performed.

The rest of this paper is organized as follows. In Section 2, we describe related work. We intoduce the details of our method in Section 3. Three examples are used to examine the effectiveness of our method. Their experimental results are shown in Section 4. Then, we discuss the results in Section 5 and give some conclusion in Section 6.

2 Related research

2.1 Ant Colony Optimization

Ant colony optimization[10] is a metaheuristic method that was developed based on the behavior of ants in the natural world. Ant colony optimization is suitable for finding optimal solutions to problems involving graphs such as Travelling Salesman Problem.

In ant colony optimization, virtual ants probabilistically select paths according to the amount of pheromone between nodes in the path, and an optimal solution is obtained by walking the nodes. Ants do not have a direct means of communication, but create the solution by indirect communication via pheromones. Ants leave pheromones on the paths they follow, and because

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 15:29:52 UTC from IEEE Xplore. Restrictions apply.

the pheromones decrease by evaporation over time, the largest amount of pheromones remain on the path with the shortest distance (best fitness). Repetition of these steps creates a mechanism by which the most pheromones are left on the best path, thereby generating the solution with the best fitness.

2.2 $ACO_{\mathbb{R}}$

$ACO_{\mathbb{R}}$[14] is an NN weight-training method based on ant colony optimization as described in Section 2.1. In $ACO_{\mathbb{R}}$, an N-dimensional vector (weight vector) is assumed for the solution, and R candidates of the weight vector are stored in an archive. R weight vectors are first created randomly in the initial stage, and then the fitness of each weight vector is calculated and stored in descending order of fitness. One weight from among the archived weights is selected by the probability in the following equation:

$$Pr(select\ s_a) = \frac{\omega_a}{\sum_{r=1}^{R} \omega_r}. \tag{1}$$

Taking ω_r as an arbitrary constant q gives the following equation:

$$\omega_r = g(a; 1, qR), \tag{2}$$

where g(y;μ,σ) is the probability density function of a normal distribution of the mean μ and standard deviation σ, given by the following equation:

$$g(y; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}. \tag{3}$$

Taking s_a as the weight selected by Eq. (1) above and s_i as the newly created weight, the jth value s_i $s_{i,j}$ is a value sampled by the probability density function for a normal distribution of the mean $s_{a,j}$ (the jth component of s_a) and standard deviation $\sigma_{a,j}$ calculated by Eq. (4):

$$\sigma_{a,j} = \xi \sum_{r=1}^{R} \frac{|s_{a,j} - s_{r,j}|}{R-1}, \tag{4}$$

where ξ in Eq. (4) is an arbitrary constant.

The fitness of the m weight vectors created in a single loop is calculated, then R weight vectors with the highest fitness from among R archived weight vectors and m newly created weight vectors are left in the archive, and a single loop is finished.

Once the above operation has been performed with no changes to the best weights in the archive for a predetermined number of loops, or once the specified number of loops has been performed, the algorithm finishes with the topmost weight in the archive as the output.

2.3 ANN-Miner

ANN-Miner[15] is an algorithm that performs structure optimization based on ant colony optimization. There are two types of pheromones in this algorithm for each connection, one for including connections in the network and one for not including connections in the network. The method for generating a solution in ANN-Miner starts from the state where initially none of the connections are connected in the NN. The method of selecting connections is to randomly choose whether or not to include a connection in the network according to a probability calculated by Eq. (5):

$$p(D_c^a) = \frac{\tau(D_c^a)}{\tau(D_c^{True}) + \tau(D_c^{False})}, \tag{5}$$

where D_c^{True} indicates that connection c is included in the network, and D_c^{False} indicates that connection c is not included in the network. Therefore, $\tau(D_c^{True})$ is a pheromone related to including a connection and $\tau(D_c^{False})$ is a pheromone related to not including a connection. Furthermore, the parameter a in Eq. (5) is either True or False. If the D_c^a chosen probabilistically like this is D_c^{True}, then connection c is added to the NN to form a network, and the above operation is performed for all connections until the finished NN is finally output.

The fitness of the network formed by the above operation is measured. The measurement is performed for the pre-determined colony_size times, and the pheromones are updated by using the best fitness. The method for updating pheromones is to add pheromones according to Eq. (6) and then evaporate pheromones according to Eq. (7):

$$\tau(D_c^a) = \tau(D_c^a) + \tau(D_c^a) \times Q, c \in C, D_c^{True} \in NN_{tbest}. \tag{6}$$

$$\tau(D_c^a) = \frac{\tau(D_c^a)}{\tau(D_c^{True}) + \tau(D_c^{False})}, \tag{7}$$

where NN_{tbest} is the NN that gives the best fitness among the colony_size repetitions. The above operation is continued until either the pre-determined maximum number of iterations is reached or the best structure does not change for a pre-determined number of iterations and is regarded as converged. The NN with the best fitness when this process ends is ultimately trained, and the trained NN becomes the output.

In ANN-Miner, pheromones are updated but ants are not used for creating the topology solution. In addition, weights of the neural network are not necessarily trained simultaneously. In our proposed method, we let ants walk the network topology and select the connections by means of pheromones. Network weights are trained at the same time as pruning. The purpose of this is to avoid falling into local solution when the search space is changed drastically.

3 Proposed Methods

3.1 Overview of neuroevolution using ACO

By treating the NN as a graph from the input layer toward the output layer, the graph can be optimized by ant colony optimization. The method employed in the present research is to apply an algorithm to the pre-trained NN by using $ACO_{\mathbb{R}}$, as described in Section 2.2, to reduce the size of the NN. The pseudocode for the algorithm is shown in **Algorithm 1**.

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 15:29:52 UTC from IEEE Xplore. Restrictions apply.

The main procedure after initialization is to have ants walk the network for the pre-determined number of iterations or until convergence. A single connection between the layers from the input layer to the output layer is chosen for each ant by the ants in the input layer walking to the output layer along the nodes in each layer. By having multiple ants do this, multiple connections between each layer are chosen by the ants, and a new NN is formed by connecting the chosen connections. The score of that NN is then measured, and if the score of the NN is the best so far, the pheromones are updated, and the algorithm is repeated. Furthermore, if the conditions for weight training are satisfied, weight training of the NN is newly performed using $ACO_{\mathbb{R}}$. However, only the weights of the connections that are used in the best structure at that time are trained.

3.2 Choosing connections

The method by which the ant chooses a connection in the ninth line of **Algorithm 1** is that the ant chooses the next node to traverse depending on the pheromone value that exists in each connection between the layers. More specifically, the connection is chosen randomly according to the probability given by Eq. (8), taking n_i as the node where the ant is currently located, $\tau(n_i, n_j)$ as the pheromone value between the two nodes n_i and n_j, and S as the set of all nodes that can be reached from n_i:

$$p(select(n_i, n_j)) = \frac{\tau(n_i, n_j)}{\sum_{n_k \in S}(\tau(n_i, n_k))} \tag{8}$$

Connections are added between the nodes selected in this way and the nodes where the ant is currently located, and finally these connections are output. In the present research, the set S of all nodes that can be reached from n_i is all nodes in the following layer.

3.3 Evaluation and updating pheromones

A partially connected NN is formed by connecting the connections selected by the ants in this way, and the score of this NN is measured. After this, the pheromones are updated by using Eq. (9) for connection c in the formed NN, but only if it has the best score so far. Note that Q is a value based on the score, and needs adjustment for each problem. Furthermore, evaporation of pheromones is then reproduced by using Eq. (10) with e as the evaporation rate. Evaporation is performed every time regardless of the score value. $\tau(c)$ is the pheromone on connection c. Updating of the pheromone of connection c chosen by the ants using Eq. (9) is performed only when the score is the best so far, to prevent the pheromones of connections included in low-score networks from increasing and reducing the probability of connections included in the best-so-far network from being chosen for the next network structure if a series of low-score networks are created consecutively, which can occur because a structure with a higher score than before is not always created due to the randomness inherent in the structure generation.

$$\tau(c) = \tau(c) + Q. \tag{9}$$

$$\tau(c) = \tau(c) * e. \tag{10}$$

Algorithm 1 Neuroevolution using ant colony optimization

```
1:  Begin
2:  NN_bsf = φ      //NN which performed the best so far.
3:  Q_bsf = −100     //score which is the best so far.
4:  t = 1;
5:  InitializePheromone()
6:  while t < max_iterations and not Convergence do
7:      NN_i = φ     //NN which will be created in this iteration.
8:      for j = 1 ... colony_size do
9:          connection = ant_j.SelectPath()     //See Eq. (8)
10:         NN_t.append(connection)
11:     end for
12:     Q_t = Evaluation(NN_t)
13:     if Q_t > Q_bsf then
14:         NN_bsf = NN_t
15:         Q_bsf = Q_t
16:         UpdatePheromone(NN_bsf, Q_bsf)  //See Eq. (6),(7)
17:     end if
18:     if learning_condition then
19:         train weights
20:     end if
21:     t = t + 1
22: end while
23: return NN_bsf
24: End
```

3.4 Training the weights

If the following two conditions are satisfied, the NN weights are updated (see lines 18-20 of **Algorithm 1**).The first is that the number of times the ant has already walked exceeds a pre-determined number of times, and the second is that the best NN structure has not been updated for a pre-determined number of iterations.

The first condition is imposed because during the initial stages of the algorithm, even if the best structure does not change for some number of iterations, the structure can subsequently change quite drastically, and connections learned in the initial stages might not be included in the ultimate output result, rendering the weight training useless.

The second condition is imposed to decide the timing of weight training under the prerequisite that the first condition is satisfied. The idea is that in cases where it is difficult to increase the evaluation value of the NN by only architecture search by ants, changes are made to the architecture search by ants by performing weight training to change the state of the NN.

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 15:29:52 UTC from IEEE Xplore. Restrictions apply.

4 Experiments

4.1 Problem

The experiments in this work were performed using two problems from the OpenAI Gym and one problem from pybullet-gym[13], namely, Pendulum and MountainCarContinuous from OpenAI gym and InvertedDoublePendulum from pybullet-gym.

MountainCarContinuous is a problem in which momentum is created by applying force to a car in the left and right directions such that it reaches the peak of the right Mountain (position of the yellow flag) in Fig. 1. Although the end conditions for this problem are only that the peak of the right Mountain is reached and that the speed of the car when it reaches the peak exceeds some pre-determined value, in the present work we added an end condition for when the number of movements reaches a predefined value. In OpenAI Gym, rewards are set for each problem. Although the original problem was set to give a reward of +100 when the right Mountain peak is reached, in the present work we instead set the reward by adding a penalty of −20 in cases where the number of movements reaches a fixed number of times and in the case where the car does not reach the right Mountain peak. In this problem, The total value of the reward is set as the score.

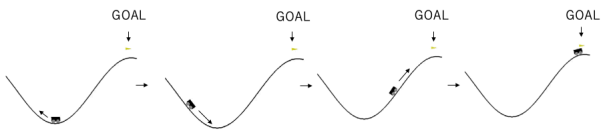


Figure 1: MountainCarContinuous

In the Pendulum problem, a red bar such as that shown in Fig. 2 must be to stand upright by applying torque to the center. When this problem was actually used in the present research, the score was calculated as the total value of the reward set by OpenAI Gym divided by the maximum number of steps (200). Furthermore, an end condition was imposed, and if some number of steps is finished without the conditions being satisfied, a penalty of −20 is added to the value after dividing by the maximum number of steps. The end condition is that a state in which the bar angle is ±5 degrees or less and the angular velocity is 1.0 or less continues for 30 steps in a row. The reason for the large penalties is not only to make it easier to escape from local solutions, but also so that the score when the end condition in MountainCarContinuous fails gives the similar value. Furthermore, the reason for dividing by the maximum number of steps is to make it easier to set the parameters related to updating the pheromone values in the algorithms in this research where the scores are similar between the problems, and to match up the orders of the scores between MountainCarContinuous and Pendulum.

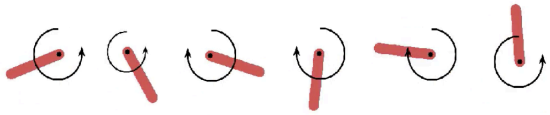


Figure 2: Pendulum

InvertedDoublePendulum has two joints and two poles and one cart shown in Fig. 3. The poles are upright when it starts and the aim of this game is to prevent the poles from falling over by moving the cart back and forth. The end condition is when the poles fall down or the number of movements reaches a fixed number of times. In pybullet-gym, rewards are set for each problem as well as OpenAI Gym, and using the reward given by pybullet-gym the score of this game is calculated by following equation:

$$score = -100/(sum\ of\ rewards). \tag{11}$$

The reason why we didn't simply set the score of this problem to sum of rewards is because we wanted to set the score of InvertedDoublePendulum similar to those of the two problems of OpenAI Gym in order to apply algorithms easily. The scores of the two problem of OpenAI gym are always negative value and the closer to 0 the score is, the better it is considered. And the score calculated by Eq. (11) has the same characteristic as those of MountainCarContinuous and Pendulum. The numerator of Eq. (11) is just to make it easier to set the parameters related to updating the pheromone values in the algorithms.

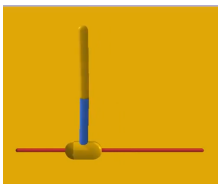


Figure 3: InvertedDoublePendulum

4.2 Experimental settings

This section describes the experiments performed in this work. To compare the fitness of the two problems described in Section 4.1, the scores from five networks were compared: a fully connected NN; an NN generated by the proposed procedure; an NN that performs architecture search using ACO with the learning_condition only on the 18th line of **Algorithm 1** always set to False so that weight training is not performed among the proposed methods for an NN then weight trainings are performed together at the end; a network in which all weight trainings are performed together first in an NN in which only architecture search by ACO is performed; and an NN generated by ANN-Miner. Because the scores for the two problems performed in this work had large variance, the NN provided at the start was shared among all conditions to make the comparison of scores fair. Note that the NN used in this work was a 4-layer NN with two intermediate layers. This NN had randomly initialized weights with weight training performed 50

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 15:29:52 UTC from IEEE Xplore. Restrictions apply.

Table 1: Summury of conditions

Condition	timing of training weights	pruning method
Condition 1	not prune	not prune
Condition 2	while pruning	ACO
Condition 3	after pruning	ACO
Condition 4	before pruning	ACO
Condition 5	after pruning	ANN-Miner

times using $ACO_{\mathbb{R}}$. We now give a detailed description of each condition and summarized the conditions in Table 1.

Condition 1 is a fully connected NN that is output by performing weight training the same number of times as that performed by the proposed method on the initially provided NN, because weight training is performed in the proposed method in Condition 2.

Condition 2 is the NN generated by the proposed method and is the NN output obtained by applying the proposed method to the initially provided NN.

Condition 3 is the NN output from performing architecture search by ACO on the provided NN and then performing weight training all together. Because of this, weight training is not performed in the middle of the architecture search. The total number of times that training is performed is the same as that in the proposed method.

Whereas weight training is performed after architecture search in Condition 3, in Condition 4 weight training is performed on the provided NN by $ACO_{\mathbb{R}}$ and then architecture search is performed by ACO. The number of times weight training is performed is the same as in Condition 3; here, too, weight training is not performed during the architecture search.

Condition 5 applies the NN provided by ANN-Miner as described in Section 2.3.

In terms of the detailed settings of the proposed method, although Section 3 stated that weight training is not performed in the initial stages of the algorithm, in the present work this specifically means that weight training is not performed for a quarter of the maximum number of episodes, with weight training performed as appropriate thereafter. Furthermore, the number of times the NN structure formed by the ants does not change in the settings of Condition 2, which performs weight training, was set to 1/10 of the maximum number of episodes. $ACO_{\mathbb{R}}$ is used for weight training, and the weight training is repeated 10 times when the conditions for performing weight training are satisfied. Because of this, once the conditions have been satisfied, weight training is performed by 10 iterations of $ACO_{\mathbb{R}}$, and architecture search by ants is performed again.

The parameters used in the experiments are summarized in Tables 2 and 3. Table 2 shows the parameter list for $ACO_{\mathbb{R}}$ and Table 3 shows the parameter list for the proposed method. Eq. (12) is used for Q, which is a value based on the score of the output NN in Eq. (9):

$$Q = \frac{r}{|Q_{bsf}|}. \tag{12}$$

Note that although Q_{bsf} is the score of the best individual, because the pheromones are added only when the best individual is updated, this is the score of the individual when these variables were updated. The variable r is a parameter that is specific to each problem. Furthermore, the parameters for the architecture search by ACO in Conditions 3 and 4, in which weight training is performed all together before or after architecture search, are the same as the proposed method in Condition 2. The parameters for ANN-Miner are as shown in Table 4, and settings such as the execution time were set according to the paper for ANN-Miner[15].

Table 2: List of parameters of $ACO_{\mathbb{R}}$

Parameter description	Value
Number of solutions produced by each loop	10
Number of times for convergence to end the algorithm in $ACO_{\mathbb{R}}$	10
q in Eq. (2) in $ACO_{\mathbb{R}}$	0.05
Number of solutions to archive in $ACO_{\mathbb{R}}$	50
ξ in Eq. (4) in $ACO_{\mathbb{R}}$	0.85

Table 3: List of parameters in the proposed method

Description	Value
Maximum number of iterations of architecture search	2000 times
Evaporation rate	0.99
Number of ants per loop	Number of input nodes*25
Number of times best individual does not change treated as convergence for ending the algorithm	1000 times
r (Pendulum) in Eq. (12)	0.75
r (MountainCarContinuous) in Eq. (12)	1.5
r (InvertedDoublePendulum) in Eq. (12)	0.5

Table 4: List of parameters in ANN-Miner

Parameter	Parameter description	Value
max_iterations(ANN-Miner)	Maximum number of iterations of ANN-Miner	50
colony_size	Number of solutions created per iteration	10
conv_iterations	Number of times the best solution does not change for regarding as convergence	10

4.3 Experimental results

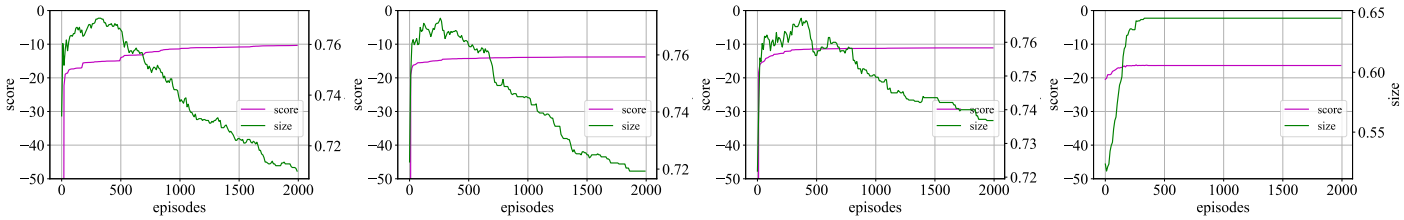
Table 5 compares the scores of the NN output by each condition for the above-mentioned three problems. The values written in the table are the mean andstandard error of 100 sets of data. The score is the result from testing the output NN 200 times while varying the initial conditions, and the size is the proportion of connections that are connected with 'fully connected' taken as 1.0. Furthermore, the CPU time is the time, including the weight training, until the algorithm ends.

Next, we explain the results of performing the t-test. Because each of the conditions in the present work were applied to a common NN, the paired t-test was performed with a significance level of 5%. Because only weight training is performed in Condition 1 without performing architecture search, the size and the CPU

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 15:29:52 UTC from IEEE Xplore. Restrictions apply.

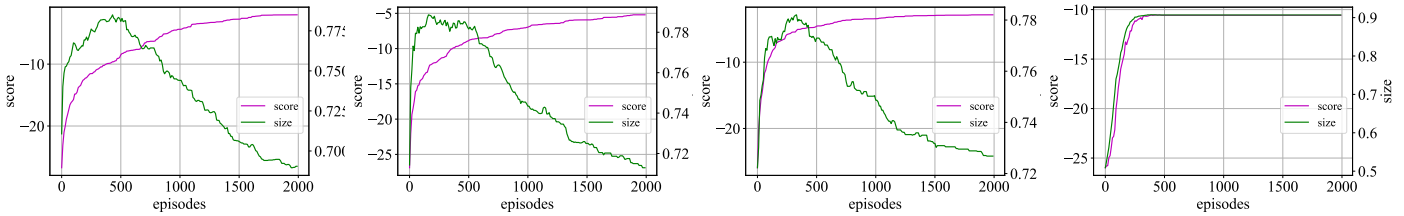
Table 5: Comparison results

	MountainCarContinuous			Pendulum			InvertedDoublePendulum		
	Score	Size	CPU time[s]	Score	Size	CPU time[s]	Score	Size	CPU time[s]
Condition1	-12.7±0.71	1.0	178±5.9	-6.03±0.57	1.00	523±13	-0.10±0.0024	1.00	851±26
Condition2	**-11.6±0.71**	**0.71±0.0063**	493±15	**-4.58±0.50**	**0.69±0.0084**	861±15	**-0.096±0.0031**	**0.62±0.0059**	**1420±35**
Condition3	-12.3±0.73	0.73±0.0063	**479±13**	-5.72±0.58	0.74±0.0087	**844±14**	**-0.094±0.0025**	**0.61±0.0059**	**1412±31**
Condition4	**-11.6±0.72**	0.75±0.0056	**473±13**	**-4.72±0.52**	0.74±0.0064	**832±14**	**-0.096±0.0028**	**0.60±0.0055**	1470±33
Condition5	-12.8±0.73	**0.67±0.018**	561±20	-5.69±0.58	0.91±0.011	1102±22	-0.10±0.0030	0.90±0.0060	1168±32



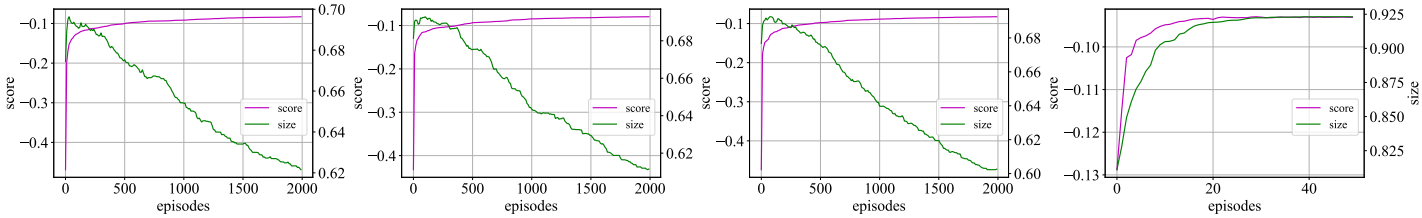
(a) Condition2 (b) Condition3 (c) Condition4 (d) Condition5

Figure 4: Changes of size and score of the best NN along with episodes for MountainCarContinuous



(a) Condition2 (b) Condition3 (c) Condition4 (d) Condition5

Figure 5: Changes of size and score of the best NN along with episodes for Pendulum



(a) Condition2 (b) Condition3 (c) Condition4 (d) Condition5

Figure 6: Changes of size and score of the best NN along with episodes for InvertedDoublePendulum

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 15:29:52 UTC from IEEE Xplore. Restrictions apply.

time were statistically significantly higher than those of other conditions for all problem. And a significance level was set to 5%.

First, we describe the statistically significant results among MountainCarContinuous scores shown in table 5. Although the scores of Conditions 2 and 4 were significantly reduced compared with the other conditions, no significant differences were found between the other scores. In terms of size, Condition 1 was naturally significantly larger than the other conditions. For Conditions 2 and 5, the p value was 0.058, which is slightly higher than 0.05, and thus no significant difference was found; however, significant differences were found between the other conditions. For CPU time, Condition 5 took significantly longer than Conditions 2, 3, and 4, and Condition 2 took significantly longer than Conditions 3 and 4. No significant difference was found between Conditions 3 and 4.

Next, for the Pendulum scores, Conditions 2 and 4 were found to have significantly lower scores compared with the other condition; no significant differences were found between the other conditions, the same as for MountainCarContinuous. In terms of size, no significant difference was found between Conditions 3 and 4; however, significant differences were found between the other conditions. Finally, for CPU time, a significant difference was found only between Conditions 3 and 4, and no significant differences were found between the other conditions.

For the InvertedDoublePendulum scores, Conditions 2, 3 and 4 were found to have significantly lower scores compared with Condition 1; no significant differences were found between the other conditions. In terms of size, no significant difference was found among Conditions 2, 3 and 4 ; however, significant differences were found between the other conditions. Finally, for CPU time, no significant difference was found between Conditions 2 and 3, and significant differences were found among the other conditions.

Figs. 4, 5 and 6 show how the score and the size of the best NN changed along with episodes for MountainCarContinuous, Pendulum and InvertedDoublePendulum respectively. In Figs. 4 and 5, when the pruning method was ACO (conditions 2,3 and 4), the size became larger once then the size became smaller regardless of the timing of training weights. In Fig 6 the size became larger first but the trend mentioned above was not remarkable compared with Figs 4 and 5.

5 Discussion

In terms of scores, there was no significant difference in the scores between Conditions 3 and 5, which perform weight training after NN architecture search, and the fully connected NN (Condition 1) in Pendulum and MountainCarContinuous. In contrast, Condition 2, in which weight training was performed during architecture search, and Condition 4, in which weight training was performed before architecture search, had significantly better scores compared with the fully connected NN. One possible reason for this is that because the initially provided NN fell into a local solution, even though architecture search was performed at that stage, it was not possible to escape from the local solution by architecture search alone and the appropriate combination of connections could not be chosen. However, in Condition 2, where weight training was performed while performing architecture search, because weight training was performed when it became difficult to increase the score by architecture search, it was possible to escape from the local solution. Furthermore, in Condition 4, where weight training was performed first, appropriate connections were chosen because the architecture search was performed after escaping from the local solution to some degree by weight training. Furthermore, because the score varied greatly depending on whether or not the current problem was solved, the local solution was easily fallen into when the conditions for solving the problem were not met. Because of this, if a combination of connections that can satisfy the conditions for solving the problem is not found by the architecture search in Conditions 3, 4, and 5, it converges immediately and the architecture search is meaningless. However, because weight optimization is performed first in Condition 4, a combination of connections that can satisfy the conditions for solving the problem is easy to find, and the score is better than Conditions 3 and 5. In Condition 2, even if a combination of connections that can satisfy the conditions for solving the problem cannot be found by architecture search, because weight training is performed, it does not fall into a local solution and gives good scores. In InvertedDoublePendulum, there were no statistically significance among Conditions 2, 3 and 4. The scores of these conditions were significantly higher than that of Condition 1. That seems to be because the number of inputs of this problem is 9 and those of Pendulum and MountainCarContinuous is 3 and 2 respectively so the NNs for InvertedDoublePendulum were more complex than those of the other problems. This leads to well performing easily by pruning connections of NNs.

In terms of CPU time, whereas the decision to include a connection in the output NN is made by ANN-Miner for each connection, because the ants choose only the next node to traverse in ACO, ANN-Miner takes a longer time to perform the NN architecture search.

In terms of changes of the size and the scores of the best NN along with episodes shown in Figs. 4, 5 and 6, sizes were not converged in some conditions. That is because the best NN was updated only when the score became larger in the algorithms so if there was still room to reduce the size, the best NN wasn't updated when the score didn't become larger. In Figs. 4 and 5, when the pruning method was ACO (conditions 2,3 and 4), the size became larger once then the size became smaller regardless of the timing of training weights. That means ants create relatively large NN in order to increase the score first then find a conbination of connections of smaller NN whose score is slightly higher than larger NNs. We didn't include the size in calculation of score so it was possible for ants to create large NN if the large NN had good score but the size actually became smaller and smaller in all problems when the pruning method was ACO. That means smaller NN can have better score than larger NN when the proper combination of connections is found in our experiments and ACO

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 15:29:52 UTC from IEEE Xplore. Restrictions apply.

can find it.

6 Conclusion

This research showed that performing architecture search by ACO and weight training simultaneously is effective for increasing the score of NNs. Furthermore, when using NNs to solve two problems from OpenAI Gym, it was found that the score was increased significantly by performing weight training before or at the same time as architecture search compared with NNs in which only weight training is performed or weight training is performed after architecture search. When using InvertedDoublePendulum from pybullet-gym, architecture search led higher preformance regardless of the order of weight training and architecture search by ACO.

The problems used in this work were limited to those in OpenAI Gym and pybullet-gym and were simple problems with relatively few inputs; therefore it will be necessary to apply the methodology to more difficult problems in the future and comfirm the effectiveness of neuroevolution with ACO.

References

[1] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[2] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[3] Jason Gauci and Kenneth O Stanley. A case study on the critical role of geometric regularity in machine learning. In *AAAI*, pages 628–633, 2008.

[4] Akira Hara, Jun-ichi Kushida, Koji Kitao, and Tetsuyuki Takahama. Neuroevolution by particle swarm optimization with adaptive input selection for controlling platform-game agent. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2504–2509. IEEE, 2013.

[5] Travis Desell, Sophine Clachar, James Higgins, and Brandon Wild. Evolving deep recurrent neural networks using ant colony optimization. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 86–98. Springer, 2015.

[6] AbdElRahman ElSaid, Brandon Wild, Fatima El Jamiy, James Higgins, and Travis Desell. Optimizing lstm rnns using aco to predict turbine engine vibration. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 21–22, 2017.

[7] Edvinas Byla and Wei Pang. Deepswarm: Optimising convolutional neural networks using swarm intelligence. In *UK Workshop on Computational Intelligence*, pages 119–130. Springer, 2019.

[8] Bin Wang, Yanan Sun, Bing Xue, and Mengjie Zhang. Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

[9] Bin Wang, Yanan Sun, Bing Xue, and Mengjie Zhang. Evolving deep neural networks by multi-objective particle swarm optimization for image classification. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 490–498, 2019.

[10] Marco Dorigo and Mauro Birattari. Ant colony optimization. encyclopedia of machine learning, 2010.

[11] Rahul Karthik Sivagaminathan and Sreeram Ramakrishnan. A hybrid approach for feature subset selection using neural networks and ant colony optimization. *Expert systems with applications*, 33(1):49–60, 2007.

[12] Openai gym. https://gym.openai.com/.

[13] pybullet-gym. https://github.com/benelot/pybullet-gym.

[14] Krzysztof Socha and Marco Dorigo. Ant colony optimization for continuous domains. *European journal of operational research*, 185(3):1155–1173, 2008.

[15] Khalid Salama and Ashraf M Abdelbar. A novel ant colony algorithm for building neural network topologies. In *International Conference on Swarm Intelligence*, pages 1–12. Springer, 2014.

Authorized licensed use limited to: Hebei University of Technology. Downloaded on February 02,2023 at 15:29:52 UTC from IEEE Xplore. Restrictions apply.