International Journal of
Science and Engineering Investigations vol. 9, issue 97, February 2020
Received on February 4, 2020 ISSN: 2251-8843

Evolutionary Training of Autoencoders by Particle Swarm Optimization

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

Abstract- In this paper, the author experimentally evaluates the ability of Particle Swarm Optimization (PSO) algorithm in evolutionary training of autoencoders. PSO is a representative instance of swarm intelligence algorithms. An autoencoder is a component of a deep neural network known as a stacked autoencoder. Optimization of neural networks by means of evolutionary algorithms is called neuroevolution. Weights and biases in an autoencoder are optimized by PSO so that the autoencoder can precisely reconstruct its input data. A dataset of handwritten digits is used in the experiment. The result showed that PSO could evolve autoencoders that reconstructed the training and test data well. The result is then compared with previous experimental results by Evolution Strategy (ES) and Genetic Algorithm (GA), in order to investigate whether PSO is better than those two evolutionary algorithms on the task. The comparison revealed that PSO could train autoencoders significantly better than GA could, but no significant difference was observed between PSO and ES. Both PSO and ES are good at exploitation, while GA is good at exploration. Thus, this result of comparison indicates that exploitation contributes more to neuroevolution than exploration does.

Keywords- *Neural Network, Swarm Intelligence, Optimization*

I. INTRODUCTION

Deep neural networks and their learning algorithms have been actively researched recently [1-14]. A stacked autoencoder is a kind of the deep neural network, where an autoencoder is a kind of layered feedforward neural networks [1,7,8]. An autoencoder can be trained by the well-known backpropagation (BP) algorithm [15], but the training of neural networks by the BP algorithm is likely to get stuck in an undesirable local minimum because the algorithm is based on a gradient decent method. Besides, several methods are proposed for training neural networks by using evolutionary algorithms, known as neuroevolution and evolutionary neural networks [16,17]. An advantage of evolutionary algorithms over the BP in training neural networks is that evolutionary algorithms can globally search solutions well and thus the trained neural networks are less likely to get stuck in an undesirable local minimum [18-23]. Therefore, we can expect that evolutionary algorithms contribute well to the training of autoencoders (and thus stacked autoencoders).

The author previously reported experimental results by Genetic Algorithm (GA) [24] and Evolution Strategy (ES) [25]. Several algorithms other than GA and ES can also be applied to the training, and it is known as the "no free lunch" theorem that no single search method is able to find a better solution than any other method does [26]. Thus, various evolutionary methods should be compared for investigating which method suits better and why. As an alternative, Particle Swarm Optimization (PSO) [27] is adopted in this paper. This paper reports an experimental result by PSO, and compares it with the previously reported results by GA and ES. The same dataset of handwritten digits is consistently used in these experiments.

II. AUTOENCODER

An autoencoder [1,7,8] is a layered feedforward neural network where the number of units in the output layer is the same as that in the input layer. An autoencoder is trained to output the same values as input values, in other words, to reproduce their input data. Fig. 1 shows the topology of an autoencoder adopted in this research. It has a single hidden layer. Usually, the number of hidden units is smaller than those of input (output) layer: N dimensional input real vectors are encoded (compressed) to $M(< N)$ dimensional real vectors between the input and hidden layers, and the M dimensional real vectors are decoded (decompressed) to the N dimensional real vectors between the hidden and output layers. Note that the compression/decompression process is not lossless but lossy so that the output vectors are not exactly be the same as the input vectors. An autoencoder is trained to make the error between the input and output vectors smaller.

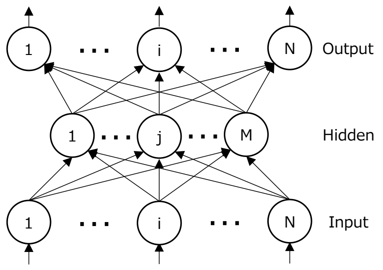

Figure 1. Topology of an autoencoder in this research.

The feedforward calculations in this autoencoder are the same as those in the traditional three layered perceptron. The following equations (1)-(5) show the calculations.

Input layer:

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

Hidden layer:

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

$$out_j^{(2)} = f(in_j^{(2)}), j = 1,2,\dots,M \tag{3}$$

Output layer:

$$in_i^{(3)} = \theta_i^{(3)} + \sum_j w_{j,i}^{(3)} out_j^{(2)}, i = 1,2,\dots,N \tag{4}$$

$$out_i^{(3)} = f(in_i^{(3)}), i = 1,2,\dots,N \tag{5}$$

The symbols in (1)-(5) denote as follows:

x_i	Input value to i-th input unit.
$out_i^{(1)}$	Output value from i-th input unit.
$in_j^{(2)}$	Input value to j-th hidden unit.
$w_{i,j}^{(2)}$	Weight value from i-th input unit to j-th hidden unit.
$\theta_j^{(2)}$	Bias value of j-th hidden unit.
$out_j^{(2)}$	Output value from j-th hidden unit.
$in_i^{(3)}$	Intput value to i-th output unit.
$w_{j,i}^{(3)}$	Weight value from j-th hidden unit to i-th output unit.
$\theta_i^{(3)}$	Bias value of i-th output unit.
$out_i^{(3)}$	Output value from i-th output unit.

$f()$ is a unit activation function, where the sigmoidal one is adopted in this research: $f(x) = 1/(1 + e^{-x})$.

Suppose the training data are N dimensional real vectors and the number of the data is D.

$$\boldsymbol{X} = \{\boldsymbol{x}_d\}, d = 1,2,\dots,D \tag{6}$$

$$\boldsymbol{x}_d = (x_{d,1}, x_{d,2}, \dots, x_{d,N}) \tag{7}$$

In (6), $\boldsymbol{X}$ is the set of training data. Each $\boldsymbol{x}_d$ in (7) is the N-dimensional real vector. An autoencoder is trained (i.e., values of $w_{i,j}^{(2)}$, $\theta_j^{(2)}$, $w_{j,i}^{(3)}$, and $\theta_i^{(3)}$ are optimized) so that its output values ($out_i^{(3)}, i = 1,2,\dots,N$) become closer to its input values ($x_{d,i}, i = 1,2,\dots,N$). In other words, the input value $x_{d,i}$ is the target for the output value $out_i^{(3)}$. Thus, the error between $x_{d,i}$ and $out_i^{(3)}$ becomes smaller by optimizing the value of weights and biases.

$$e_d = \frac{1}{N}\sum_{i=1}^{N}(out_i^{(3)} - x_{d,i})^2 \tag{8}$$

$$e = \frac{1}{D}\sum_{d=1}^{D} e_d \tag{9}$$

e_d in (8) denotes the error for $\boldsymbol{x}_d$ ($0\% \le e_d \le 100\%$), and e in (9) denotes the average error over the entire training data $\boldsymbol{X}$ ($0\% \le e \le 100\%$).

III. Evolutionary Training by Particle Swarm Optimization

Instead of the BP, PSO is adopted as a training method of the autoencoder in this research. PSO is a representative instance of swarm intelligence algorithms. Both of swarm intelligence algorithms and evolutionary algorithms are population-based stochastic search algorithms, whereas the BP is a gradient-based single-point search algorithm. Because of this difference, swarm/evolutionary algorithms are better than the BP in searching solutions globally. It was reported that evolutionary algorithms could optimize neural networks better than the BP did [18-23].

Optimization of neural networks by means of evolutionary algorithms is called neuroevolution [16,17]. There are two types of neuroevolution methods: (A) the topology of a neural network (e.g., the number of hidden layers, the number of units in each hidden layer) is fixed and the weights are optimized, or (B) both of the topology and the weights are optimized. In this paper, the author adopts the former method. The autoencoder with the topology shown in Fig. 1 includes $2MN$ $(= MN + MN)$ weights and $M + N$ biases. Thus, the autoencoder includes $2MN + M + N$ parameters in total. Let us denote $S = 2MN + M + N$. These parameters forms an S dimensional real vector, $\boldsymbol{y} = (y_1, y_2, \dots, y_S)$, and the vector is treated as the genotype in an evolutionary/swarm algorithm. The phenotype in the algorithm is the autoencoder in Fig. 1. Evolutionary/swarm operators are applied to optimize $\boldsymbol{y}$ so that the error becomes smaller. The error value e in (9) is calculated with the training data and the output values of an autoencoder.

The process of training neural networks by PSO is as follows in this paper:

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

In Step1, $\boldsymbol{y}^1, \boldsymbol{y}^2, \dots, \boldsymbol{y}^\lambda$ are initialized as random values, where λ denotes the population size. $\boldsymbol{y}^k$ denotes the genotype vector of the k-th particle in the population, i.e., $\boldsymbol{y}^k = (y_1^k, y_2^k, \dots, y_S^k), k = 1,2,\dots,\lambda$. The value of λ is given. The domain range of each genotype value should be neither too large nor too small in this research because the value is used as a weight or bias value in a neural network. In Step2, fitness of new particles (those with new genotype values) are evaluated. In this research, the fitness is based on the error in (9). A particle with a smaller error fits better. In Step3, the loop of swarm process is finished if a given termination condition is met. In this research, the loop is finished if the loop counter reaches to a given number. In Step4, the personal best (*Pbest*) of each particle and the global best (*Gbest*) among the population are updated based on the fitness scores. The *Pbest* of a particle is the genotype vector with the best fitness score found so far by the particle, and the *Gbest* is the best genotype vector among the λ Pbests. Let us denote each *Pbest* and the *Gbest* as $\boldsymbol{p}^k$ and $\boldsymbol{g}$ respectively. In Step5, the velocity is updated for each particle. Let us denote the S dimensional velocity for the k-th particle as $\boldsymbol{v}^k = (v_1^k, v_2^k, \dots, v_S^k)$. Each $\boldsymbol{v}^k$ is updated as:

$$\boldsymbol{v}^k \leftarrow \alpha \boldsymbol{v}^k + c_1 r_1 (\boldsymbol{p}^k - \boldsymbol{y}^k) + c_2 r_2 (\boldsymbol{g} - \boldsymbol{y}^k), \qquad (10)$$

where α is the inertia weight (0.9 in this research), c_1 and c_2 are constants (both 1.0 in this research), r_1 and r_2 are uniformly random values within the interval $[0,1]$. Each value of v_j^k in the velocity $\boldsymbol{v}^k$ is restricted within the interval $[-0.05, 0.05]$ in this research, in order to avoid premature convergence. In Step6, each particle moves in the search space at the velocity. $\boldsymbol{y}^k$ is updated as:

$$\boldsymbol{y}^k \leftarrow \boldsymbol{y}^k + \boldsymbol{v}^k \qquad (11)$$

IV. Experiment

This section reports an experimental study in which a dataset of handwritten digits is used as training data. The dataset is the Optical Recognition of Handwritten Digits Data Set which is available in the UC Irvine Machine Learning Repository.[https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits]

For each of the 10 digits (0,1,…,9), 20 samples are randomly extracted from the data file `optdigits.tra`. Thus, the total number of the sampled data is $10 \times 20 = 200$. A half of the 200 data is used as the training data, and the remaining half is used as the test data. Each data consists of $8 \times 8 = 64$ pixels and a pixel is valued with either of 0,1,…,16 (0: white, 16: black). In this experiment, the pixel values are normalized to a real value within the interval $[0.0, 1.0]$ by dividing the values by 16.0. Figs. 2 and 3 visually show the training and test data respectively.

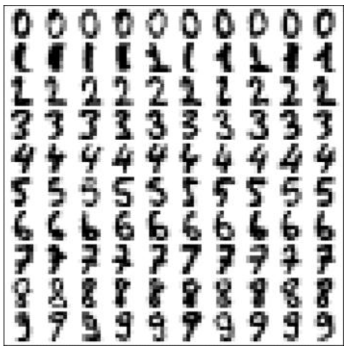
Figure 2. Training data in this experiment.

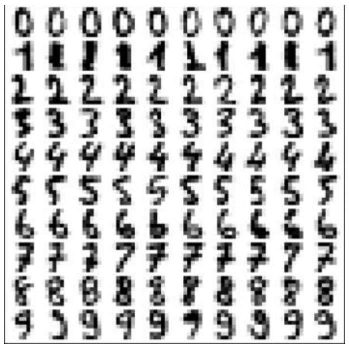
Figure 3. Test data in this experiment.

The numbers of units in the input and output layers are 64, because each training data consists of 64 values. The number of hidden units is set to 32, i.e., 50% of the input/output units. Thus, in this experiment, an autoencoder has 64×32+32×64 weights and 32+64 biases in total, and the genotype is a 4192 dimensional real vector.

Parameter values of PSO are experimentally set as follows:

- Population size λ: 100.
- Number of loops: 10,000.
- Number of evaluations: 100×10,000 = 1,000,000
- Genotype values: within the interval $[-5.0, 5.0]$.
- Initial genotype values: randomly sampled from the standard normal distribution $N(0,1)$.

The values of error *e* in (9) were observed for 10 runs. Table 1(a) shows the best (smallest), the worst (largest), and the average of the 10 training error values. Fig. 4 shows the outputs by the trained autoencoder with the best error of 7.87%. The trained autoencoder reconstructs each of the 100 input digits in Fig. 2 to the corresponding output digit in Fig. 4 (the error between the corresponding input/output digits is 7.87% per pixel in average).

If the trained autoencoders overfit to the training data, the training error becomes small but the test error becomes much larger. Table 1(b) shows the test errors, where the test data (Fig. 3) are input to the 10 trained autoencoders. The best one of the 10 trained autoencoders reconstructs each of the 100 input digits in Fig. 3 to the corresponding output digit in Fig. 5, where the error between the corresponding input/output digits is 10.55% per pixel in average. Table 1 reveals that the test errors are larger than the training errors but the differences are small. Thus, PSO did not make autoencoders overfit to the training data.

TABLE I. Training and Test Erros by PSO(%).

	(a) training			(b) test		
	best	worst	average	best	worst	average
PSO	7.87	9.31	8.69	10.55	11.65	11.18

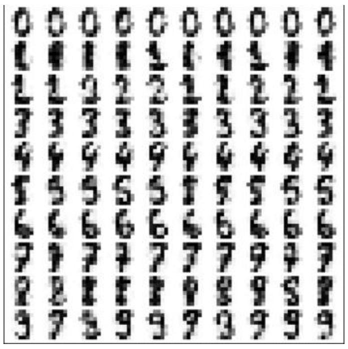
Figure 4. Output digits by the best autoencoder (reconstruction of input digits in Fig. 2)

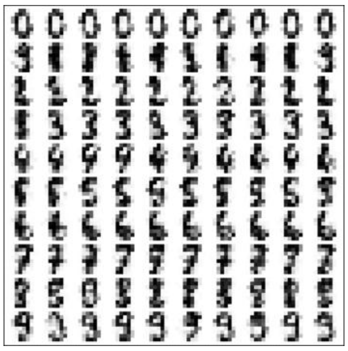
Figure 5. Output digits by the best autoencoder (reconstruction of input digits in Fig. 3)

The author next compares the result by PSO with those by other algorithms, ES and GA, in order to investigate whether PSO is better than those two evolutionary algorithms on this task. The author previously reported experimental results by ES and GA [24, 25], where the same training and test data were adopted. In the same manner described in this paper for PSO, ES and GA were applied to evolutionary training of autoencoders. The values of error e in (9) were observed for 10 runs with each algorithm. Table 2 shows the training and test errors in the same manner as in table 1. Firstly, all the error values in table 1(a) are smaller than the corresponding values in table 2(a). Secondly, all the error values in table 1(b) are also smaller than the corresponding values in table 2(b) except that the best error by PSO (10.55, in table 1(b)) is slightly larger than those by ES (10.43 in table 2(b)) and GA (10.54 in table 2(b)). The differences of training/test errors are statistically tested to confirm whether PSO could train significantly better than ES and GA. The Wilcoxon rank-sum test revealed that,

- The training errors by PSO were significantly smaller than those by GA (p=0.001045), but they were not significantly smaller than those by ES (p=0.2406), and
- The test errors by PSO were significantly smaller than those by GA (p=2.165×10^{-5}, but they were not significantly smaller than those by ES (p=0.602).

Thus, PSO could train significantly better than GA but not than ES.

TABLE II. TRAINING AND TEST ERROS BY ES AND GA(%).

	(a) training			(b) test		
	best	worst	average	best	worst	average
ES	8.26	10.50	9.07	10.43	12.50	11.23
GA	8.42	11.34	9.67	11.54	13.93	12.63

Fig. 6 illustrates the training error curves along with the progress of evolutionary training (the number of evaluations). Each curve shows errors in the best run among the 10 runs by each algorithm. Fig. 6 revealed that the errors by PSO were smaller than those by ES and GA in all of the early, middle and late stages of evolution. Although the 10 final errors by PSO were not significantly smaller than those by ES, in the early stage of evolution, the errors by PSO were much smaller than those by ES (in Fig. 6, the blue curve for PSO is much under than the gray curve for ES in the early stage). Thus, PSO could train more efficiently than ES. This result demonstrates the better exploration ability of PSO in the early stage.

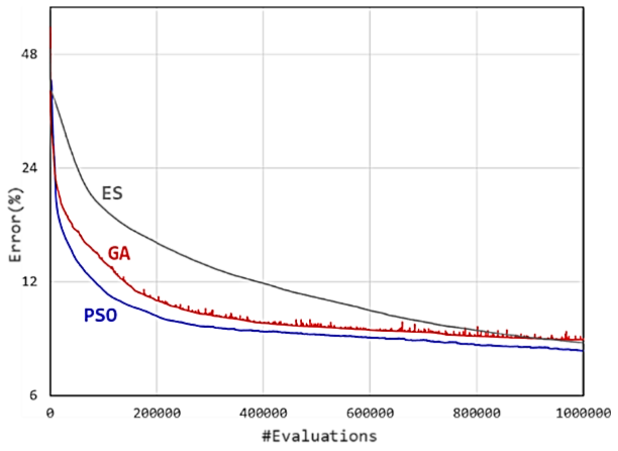

Figure 6. Training errors by PSO, ES and GA.

The experimental result in the previous paper [25] indicated that algorithms with better ability of exploitation in the later stage of optimization will suit more for the neuroevolution task. The result in this paper confirmed this: the reason why PSO contributed better than GA will be because PSO is good at exploitation in the later stage, as ES is. In Fig. 6, the training errors by PSO and ES did not stagnate but decreased gradually in the later stage, meaning that PSO and ES could continuously optimize autoencoders by their ability of fine-tuning.

V. CONCLUSION AND FUTURE WORK

The author adopted PSO to the evolutionary training of an autoencoder. The experimental result with the data of handwritten digits showed that PSO contributed significantly better than GA but not significantly better than ES. This result confirms that algorithms with better ability of exploitation in the later stage of their optimization suit more for neuroevolution. The author will further evaluate, compare and improve the abilities of other evolutionary/swarm algorithms in the training of deep neural networks.

REFERENCES

[1] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504-507.

[2] Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural Computation, 18(7), 1527-1554.

[3] Boureau, Y. L., & Cun, Y. L. (2008). Sparse feature learning for deep belief networks. Advances in Neural Information Processing Systems, 1185-1192.

[4] Sutskever, I., & Hinton, G. E. (2008). Deep, narrow sigmoid belief networks are universal approximators. Neural Computation, 20(11), 2629-2636.

[5] Bengio, Y. (2009). Learning deep architectures for AI. Foundations and Trends in Machine Learning, 2(1), 1-127.

[6] Larochelle, H., Bengio, Y., Louradour, J., & Lamblin, P. (2009). Exploring strategies for training deep neural networks. Journal of Machine Learning Research, 10(Jan), 1-40.

[7] Tan, C. C., & Eswaran, C. (2010). Autoencoder neural networks: a performance study based on image reconstruction, recognition and compression. Lambert Academic Publishing.

[8] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P. A. (2010). Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. Journal of Machine Learning Research, 11(Dec), 3371-3408.

[9] Salakhutdinov, R., & Hinton, G. (2012). An efficient learning procedure for deep Boltzmann machines. Neural Computation, 24(8), 1967-2006.

[10] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 1097-1105.

[11] Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: a review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(8), 1798-1828.

[12] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[13] Schmidhuber, J. (2015). Deep learning in neural networks: an overview. Neural Networks, 61, 85-117.

[14] Zhang, S., Choromanska, A. E., & LeCun, Y. (2015). Deep learning with elastic averaging SGD. Advances in Neural Information Processing Systems, 685-693.

[15] Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-538.

[16] Yao, X. (1999). Evolving artificial neural networks. Proceedings of the IEEE, 87(9), 1423-1447.

[17] Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. Evolutionary Intelligence, 1(1), 47-62.

[18] Montana, D. J., & Davis, L. (1989). Training feedforward neural networks using genetic algorithms. IJCAI, 89, 762-767.

[19] Sexton, R. S., Dorsey, R. E., & Johnson, J. D. (1998). Toward global optimization of neural networks: a comparison of the genetic algorithm and backpropagation. Decision Support Systems, 22(2), 171-185.

[20] Sexton, R. S., & Gupta, J. N. (2000). Comparative evaluation of genetic algorithm and backpropagation for training neural networks. Information Sciences, 129(1), 45-59.

[21] Örkcü, H. H., & Bal, H. (2011). Comparing performances of backpropagation and genetic algorithms in the data classification. Expert Systems with Applications, 38(4), 3703-3709.

[22] Joy, C. U. (2011). Comparing the performance of backpropagation algorithm and genetic algorithms in pattern recognition problems. International Journal of Computer Information Systems, 2(5), 7-12.

[23] Che, Z. G., Chiang, T. A., & Che, Z. H. (2011). Feed-forward neural networks training: a comparison between genetic algorithm and back-propagation learning algorithm. International Journal of Innovative Computing, Information and Control, 7(10), 5839-5850.

[24] Okada, H. (2017). Neuroevolution of autoencoders by genetic algorithm, International Journal of Science and Engineering Investigations (IJSEI), 6(65), 127-131.

[25] Okada, H. (2018). Comparison of ES and GA applied to neuroevolution of Autoencoders, International Journal of Science and Engineering Investigations (IJSEI), 7(80), 1-5.

[26] Wolpert, D.H., & Macready, W.G. (1997). No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1(1), 67-82.

[27] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization, IEEE International Conference on Neural Networks, IV, 1942-1948.

How to Cite this Article:

Okada, H. (2020) Evolutionary Training of Autoencoders by Particle Swarm Optimization. International Journal of Science and Engineering Investigations (IJSEI), 9(97), 29-33. http://www.ijsei.com/papers/ijsei-99720-05.pdf