International Journal of
Science and Engineering Investigations vol. 7, issue 80, September 2018
ISSN: 2251-8843

Comparison of ES and GA Applied to Neuroevolution of Autoencoders

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

Abstract- In this paper, the author compares two evolutionary algorithms, an evolution strategy (ES) and a genetic algorithm (GA), for evolutionary training of autoencoders. 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 evolutionary algorithms so that the autoencoder can precisely reproduce its input data. The author previously reported an experimental result by a GA [1]. This paper reports an experimental result by an ES, and compares it with the previously reported one to investigate which is better and why. The same dataset of handwritten digits is used in these two experiments. Results show that the ES evolves autoencoders better than the GA does. Because ESs are known as better at fine-tuning solutions in the later generations, the result indicates that evolutionary algorithms which are better at fine-grained search in the later phase of their optimization process suit more for neuroevolution.

Keywords- *Neural Network, Evolutionary Algorithm, Optimization*

I. INTRODUCTION

Deep neural networks and their learning algorithms have been actively researched recently [2-15]. A stacked autoencoder is a kind of the deep neural network, where an autoencoder is a kind of layered feed forward neural networks [2,8,9]. An autoencoder can be trained by the well-known back propagation (BP) algorithm [16], but the training of neural networks by the BP algorithm are 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 [17,18]. 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 [19-24]. Therefore, we can expect that evolutionary algorithms contribute well to the training of autoencoders (and thus stacked autoencoders).

The author previously reported an experimental result by the GA [1]. Several kinds of evolutionary algorithms other than GAs 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 [25]. Thus, various evolutionary methods should be compared for investigating which method suits better and why. As an alternative to the GA, the author adopts an evolution strategy (ES) [26] in this paper. This paper reports an experimental result by the ES, and compares it with the previously reported one. The same dataset of handwritten digits is used in these two experiments.

II. AUTOENCODER

An autoencoder [2,8,9] is a layered feed forward neural network where the number of units in the output layer is the same as the number of units 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 values are not exactly be the same as the input values. An autoencoder is trained to make the error between the input and output values smaller.

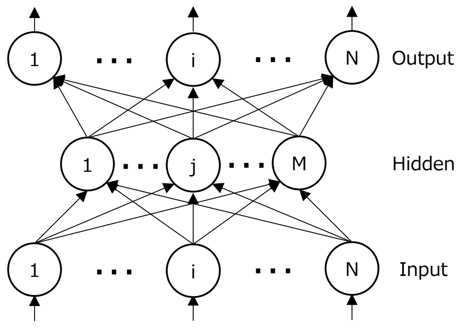


Figure 1. Topology of an autoencoder in this research.

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:

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

Hidden layer:

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

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

Output layer:

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

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

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.

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

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

In (6), $\boldsymbol{X}$ is the set of training data. Each training data ($\boldsymbol{x}_d$ 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 inputs ($x_{d,i}, i = 1,2,\dots,N$). 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.

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

$$e = \frac{1}{D}\sum_{d=1}^{D} e_d \qquad (9)$$

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 Evolution Strategy

Instead of the BP, an ES is adopted as the training method of the autoencoder in this research. Both of the ES and the GA are population-based stochastic search algorithms, whereas the BP is a gradient-based single-point search algorithm. Because of this difference, the ES and the GA are better than the BP in searching solutions globally. It was reported that an evolutionary algorithm could optimize neural networks better than the BP did [19-24]. Optimization of neural networks by means of evolutionary algorithms is called neuroevolution [17,18]. 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. These parameters consist of a $2MN + M + N$ dimensional real vector and the vector is treated as the genotype in an evolutionary algorithm. The phenotype in the algorithm is the autoencoder in Fig.1. Evolutionary operators are applied to the $2MN + M + N$ real values to optimize them 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 evolutionary process of the ES is as follows in this paper:

Step1: Initialization
Step2: Evaluation
Step3: Conditional Termination
Step4: Selection
Step5: Perturbation
Step5: Goto Step2

In Step1, genotype values ($2MN + M + N$ real values) are initialized with random numbers for each of λ individuals where λ denotes the offspring population size. 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 individuals (those with new genotype values) are evaluated. In this research, the fitness is based on the error in (9). An individual with a smaller error fits better (and thus ranked higher). In Step3, the loop of evolutionary process is finished if a given termination condition is met. In this research, the loop is finished if the number of generations reaches to a given limit. In Step4, the better μ individuals are selected as parents in the next step, where μ denotes the parent population size. The value of μ is also given. In this research, the (μ + λ)-ES is adopted. In Step5, λ offspring are produced by using the μ parents; new genotype values for an offspring are determined by applying the perturbation operator to the genotype values of a parent. Let us denote a parent genotype instance as $\boldsymbol{p} = \{p_1, p_2, \ldots, p_L\}$ and its offspring genotype instance as $\boldsymbol{c} = \{c_1, c_2, \ldots, c_L\}$, where $L = 2MN + M + N$. The parent $\boldsymbol{p}$ is randomly sampled from the μ parent population. c_i is determined by (10),

$$c_i = p_i + 0.1 * r \qquad (10)$$

where r is a random number sampled from the normal distribution.

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. [27]

For each of the 10 digits (0,1,…,9), 20 data are randomly extracted from the data file `optdigits.tra`. Thus, the total number of the sampled data is 10 × 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 × 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.

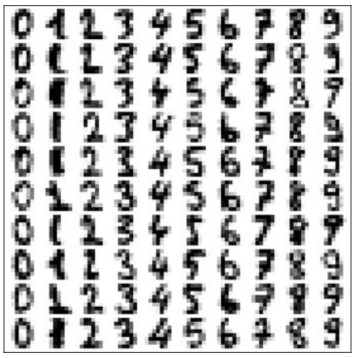


Figure 2. Training data in this experiment.

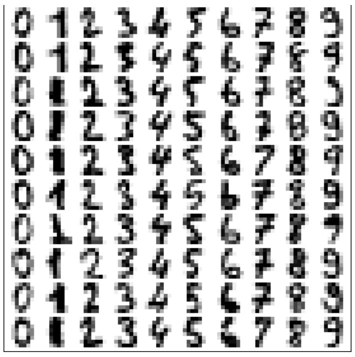


Figure 3. Test data in this experiment.

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 10%, 20%, …, 90% of 64 (i.e., 7, 13, …, 58). For example, an autoencoder with 32 (50% of 64) hidden units has 64×32+32×64 weights and 32+64 biases in total. Thus, the genotype is a 4192 dimensional real vector for the "64→32→64" autoencoder.

Parameter values of the ES and the GA are experimentally set as follows:

Both of ES and GA:

- Limit of genotype values: within the interval [-5.0, 5.0].
- Initial genotype values: randomly sampled from the standard normal distribution.
- Limit of generations: 10,000.

ES:

- Population size: μ =10, λ=100.

GA[1]:

- Population size: 100.
- Number of elite individuals: 2.
- Tournament size for the selection of parents: 10.
- α for the blend crossover: 0.5.
- Mutation probability: $1/L$, where L is the genotype length.

The values of error *e* in (9) were observed 10 times for each setting. Fig. 4 and Table 1 show the best (smallest) and the average of the 10 error values. Fig. 4 and Table 1 revealed that, for both of ES and GA, the error was smaller as the number of hidden units was larger. This is simply because an autoencoder with more hidden units is usually able to fit to training data more. Besides, the average errors were smaller for ES than those for GA. Thus, the ES could train autoencoders better than the GA could.

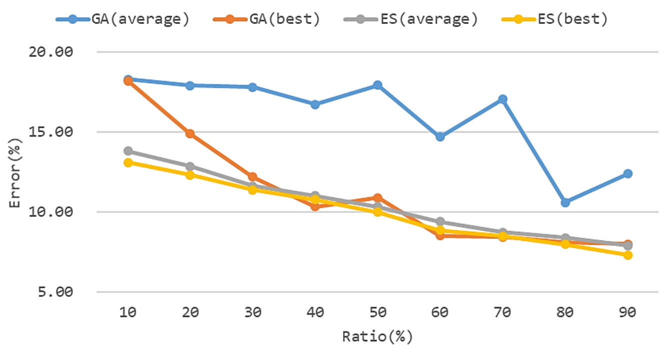


Figure 4. Ratio of hidden units and training errors.

If the trained autoencoders overfit to the training data (Fig. 2), the error on the training data becomes small but the error on the test data (Fig. 3) becomes much larger. Table 2 shows the error on the test data, where the test data are input to the trained autoencoders with the smallest training error among the 10 runs. Figs. 5-8 show the outputs by the trained autoencoders. These figures illustrate the errors on the training/test data. For example, the difference between the two figures in Fig. 5 shows the error on the training data. These results are for the "64→58→64" autoencoders. Table 2 reveals that the test errors are larger than the training errors, but the differences are small. Indeed, Figs. 6 and 8 reveal that the trained autoencoders reconstruct the test input data well. Thus, both the ES and the GA did not let the autoencoders overfit to the training data. In addition, the test errors are also smaller for the ES as the training errors are.

TABLE I. RATIO OF HIDDEN UNITS AND TRAINING ERRORS.

	GA		ES	
Ratio	average	best	average	best
10%	18.27	18.20	**13.80**	**13.10**
20%	17.89	14.87	**12.86**	**12.31**
30%	17.78	12.18	**11.63**	**11.38**
40%	16.73	**10.32**	**11.01**	10.76
50%	17.92	10.90	**10.34**	**9.99**
60%	14.68	**8.53**	**9.41**	8.87
70%	17.02	**8.44**	**8.74**	8.50
80%	10.59	8.10	**8.40**	**7.98**
90%	12.38	8.00	**7.90**	**7.33**

If the trained autoencoders overfit to the training data (Fig. 2), the error on the training data becomes small but the error on the test data (Fig. 3) becomes much larger. Table 2 shows the error on the test data, where the test data are input to the trained
autoencoders with the smallest training error among the 10 runs. Figs. 5-8 show the outputs by the trained autoencoders. These figures illustrate the errors on the training/test data. For example, the difference between the two figures in Fig. 5 shows the error on the training data. These results are for the "64→58→64" autoencoders. Table 2 reveals that the test errors are larger than the training errors, but the differences are small. Indeed, Figs. 6 and 8 reveal that the trained autoencoders reconstruct the test input data well. Thus, both the ES and the GA did not let the autoencoders overfit to the training data. In addition, the test errors are also smaller for the ES as the training errors are.

TABLE II. TRAINING ERRORS AND TEST ERRORS.

	Ratio	10%	50%	90%
GA_	Train	18.20	10.90	8.00
	Test	18.61	12.40	9.30
ES_	Train	13.10	9.99	7.33
	Test	**14.34**	**10.99**	**8.07**

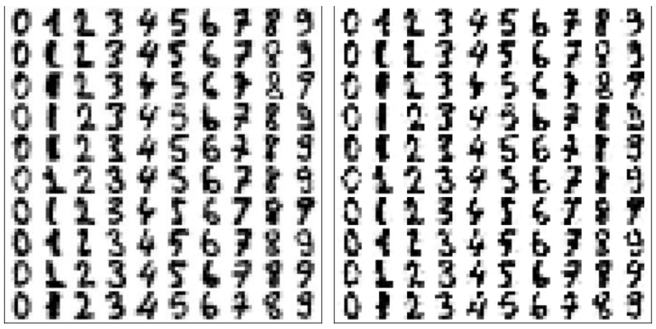

Input (Fig. 2) Output

Figure 5. Output from the best "64→58→64" autoencoder trained by GA.

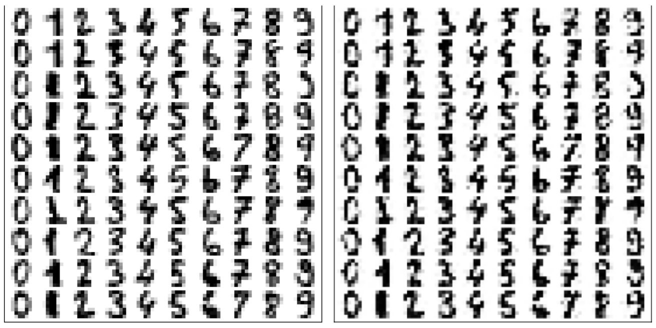
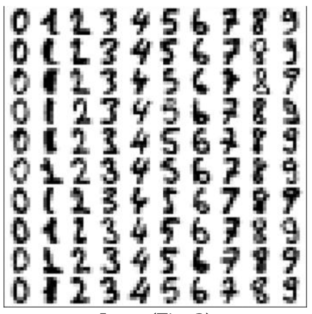

Input (Fig. 3) Output

Figure 6. Output from the same autoencoder as for Fig. 5.

These results reveals that the ES contributes better than the GA to the evolutionary training of autoencoders in this research. Because ESs are known as better at fine-tuning solutions in the later generations, the result indicates that evolutionary algorithms which are better at fine-grained search in the later phase of their optimization process suit more for neuroevolution. The author will evaluate such algorithms and compare them to the ES.

V. CONCLUSION AND FUTURE WORK

The author adopted an ES and a GA to the evolutionary training of an autoencoder. The experimental results with the data of handwritten digits showed that the ES was better than the GA. The results suggest that evolutionary algorithms which are better at fine-grained search in the later phase of their optimization process suit more for neuroevolution. Evolutionary algorithms other than ESs and GAs have been proposed. Swarm intelligence algorithms can also be adopted to the stochastic training of deep neural networks, e.g., particle swarm optimizations and artificial bee colony algorithms. The author will evaluate, compare and improve the capabilities of these evolutionary/swarm algorithms in the training of deep neural networks.


Input (Fig. 2)

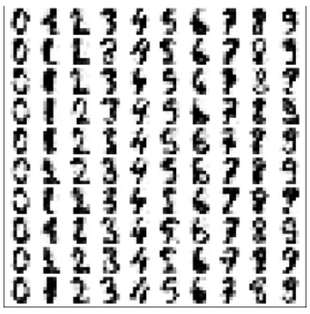
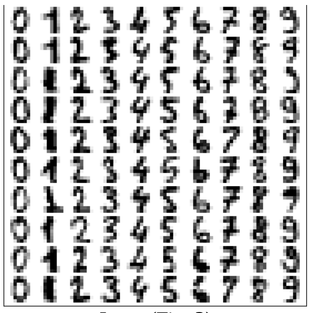

Output

Figure 7. Output from the best “64→58→64” autoencoder trained by ES.


Input (Fig. 3)

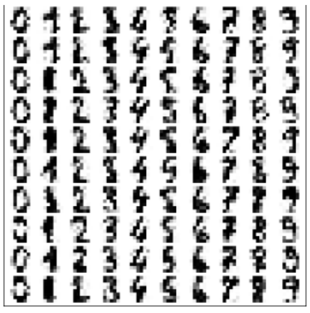

Output

Figure 8. Output from the same autoencoder as for Fig. 7.

References

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

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

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

[4] Boureau, Y. L., & Cun, Y. L. (2008). Sparse feature learning for deep belief networks. In Advances in neural information processing systems (pp. 1185-1192).

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

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

[7] 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.

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

[9] 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.

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

[11] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[12] 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.

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

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

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

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

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

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

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

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

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

[26] Beyer, H-G., & Schwefel, H-P. (2002). Evolution strategies - a comprehensive introduction, Natural Computing, 1(1) 3-52.

[27] https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits