Training Deep Autoencoder via VLC-Genetic Algorithm

Qazi Sami Ullah Khan(✉), Jianwu Li, and Shuyang Zhao

Beijing Key Laboratory of Intelligent Information Technology,
School of Computer Science and Technology, Beijing Institute of Technology,
Beijing, China
qazisamiullah170@yahoo.com

Abstract. Recently, both supervised and unsupervised deep learning techniques have accomplished notable results in various fields. However neural networks with back-propagation are liable to trapping at local minima. Genetic algorithms have been popular as a class of optimization techniques which are good at exploring a large and complex space in an intelligent way to find values close to the global optimum.

In this paper, a variable length chromosome genetic algorithm assisted deep autoencoder is proposed. Firstly, the training of autoencoder is done with the help of variable length chromosome genetic algorithm. Secondly, a classifier is used for the classification of encoded data and compare the classification accuracy with other state-of-the-art methods. The experimental results show that the proposed method achieves competitive results and produce sparser networks.

Keywords: Neural networks · Genetic algorithm · Variable length chromosome · Deep autoencoder

1 Introduction

Neural network researchers had wanted for eras to train deep multi-layer neural networks [1,2], which is inspired by the architectural depth of the human brain but before 2006 no successful attempts were reported. Positive experimental results were reported by researchers with usually one or two hidden layers but training deeper networks repeatedly returned poorer results. Hinton et al. introduced Deep Belief Networks [3], with an unsupervised learning algorithm Restricted Boltzmann Machine (RBM) [4] that greedily trains one layer at a time. Later, autoencoders based algorithms were proposed [5,6], actually taking advantage of the same principle using of unsupervised learning to train of middle layer at each level [7]. More recently, other algorithms for deep learning were proposed using neither autoencoder nor RBMs but using the same principle [8,9].

Since 2006, deep learning have been applied on various tasks such as dimensionality reduction [10], classification [5,11,12], modeling textures [13], collaborative filtering [14], regression [15], object segmentation [16], natural language processing [8,17] and information retrieval [18,19].

D. Liu et al. (Eds.): ICONIP 2017, Part II, LNCS 10635, pp. 13–22, 2017.
https://doi.org/10.1007/978-3-319-70096-0_2

In this paper we describe a variable length chromosome genetic algorithm (VLC-GA) for training deep autoencoder. We use VLC-GA for training deep autoencoder that not only succeeds in its task but outperforms backpropagation (the standard training algorithm) and another approach in [20] on MNIST handwritten digits dataset [21].

The rest of paper is organized as follow: In Sect. 2, related work is briefly reviewed. In Sect. 3, the proposed method is explained. In Sect. 4, experiments are presented. The contributions of this paper are concluded in Sect. 5.

2 Related Work

Neural networks (NNs) and genetic algorithms both have the ability to solve complex problem. The idea of combination of neural networks and genetic algorithms came up first in the late 80s, which is inspired from the nature. In real life, a successful person not only depends on his knowledge and expertise, which he gained through experience (the neural network training), but also depends on his inborn inheritance (set by the genetic algorithm) [22].

Since 1980, genetic algorithms have been effectively used for training neural networks. Genetic algorithms have been used as a replacement for the backpropagation algorithm, or in combination with backpropagation to increase the entire performance of the neural network [23]. A large number of problems have been examined by using various Genetic Algorithm Neural Networks (GANN) techniques, such as classification [24], face recognition [25], color recipe prediction [26], animates [27], etc.

In 2014, Omid E. David and Iddo Greental used a GA-assisted approach for training deep autoencoder which improved the performance of deep autoencoder and produced a sparser neural network [20]. In [28], Montana, David J., and Lawrence Davis used a different genetic algorithm for training feed forward networks which is not only prospers in its job but surpassed the standard training algorithm backpropagation on different datasets. In regards of genetic algorithm they showed a real world application of genetic algorithm to a big and difficult problem. They also show that adding domain specific information to genetic algorithm improves its performance. Philipp and Koehn [26] in their thesis, survey how genetic algorithms can be used to enhance the network topology, learning rate and initial weight of neural networks. They also inspect how various encoding strategies influence the combination of GANN. Besides this, many researchers used variable length genetic algorithm instead of constant length genetic algorithm for different problems [20,29–32].

3 Methodology

3.1 Deep Autoencoder

An autoencoder, also called auto-associator or diabolo, network is an artificial neural network. Autoencoder were first introduced in 1980s by Hinton and Parallel Distributing Processing (PDP) group by using input data as a teacher to

solve the problem of backpropagation without mentor [33]. Autoencoder is used for unsupervised learning that sets the target values to be equal to its inputs, i.e. the number of neurons at the input and output layers is equal, and the optimization goal for output neuron i is set to $x_i = \hat{x}_i$. Between input and output layers one or more hidden layers are used. Generally the number of neurons in hidden layer is less compared to input or output layers, thus making a bottleneck.

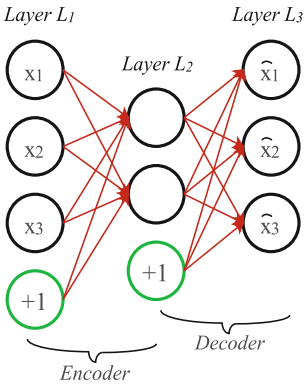

Fig. 1. Basic structure of autoencoder.

Architecturally, feedforward is the simplest form of an autoencoder. An autoencoder comprises of two parts encoder and decoder. As shown in Fig. 1, encoder consists of layer L_1 and L_2 while decoder consists of layer L_2 and L_3. The layer L_1 is input layer, L_2 is a hidden layer consisting of two neurons and each neuron represents by a function:

$$a(x) = f(Wx + b) \tag{1}$$

where W, b are weight matrix and bias vector respectively and $f(\cdot)$ is an activation function that can be sigmoid, hyperbolic, sine, gaussian function etc. And L_3 is the output layer that represents by a function $h(x) \approx x$:

$$h(a) = f'(W'a + b') \tag{2}$$

where $f'(\cdot)$, W' and b' of decoder may differ from encoder depending upon the design of autoencoder. Training of autoencoder is accomplished by reducing the reconstruction error (such as squared error):

$$E(x, \hat{x}) = \|x - \hat{x}\|^2 = \|x - f'(W'f(Wx + b) + b')\|^2 \tag{3}$$

A deep autoencoder consists of several layers of autoencoders such that the outputs of each layer are bound to the inputs of the next layer [19]. A greedy layer-wise procedure is used for obtaining good parameters for deep autoencoder.

3.2 Backpropagation Learning

Backpropagation is a technique of training artificial neural networks used in combination with an optimization method such as gradient descent. The algorithm has two main phases, propagation and weight update. The input data is transmitted forward layer by layer from the input layer to the output layer. Using a cost function, the desired output is compared to the output of network. At the output layer, for every neuron an error value is calculated. Starting from the output, the error values are then transmitted towards back and every single neuron takes its associated error value which shows its part in the original output. Later backpropagation uses these error values to compute the gradient of the cost function with respect to the weights in the network.

3.3 The Proposed Method

In this paper we propose a VLC-GA assisted approach which improves the performance of an autoencoder, and produce a sparser network. The autoencoder is trained with tied weights. We store various sets of weights W for a layer. That is, in our GA population each chromosome is one set of weights for an autoencoder. In this paper, the term of weights and chromosomes are used interchangeably. For creation of variable length chromosome, the chromosome size is multiplied with a variable v. This variable shows the maximum percent variation of the chromosome. For example if v is 20% and chromosome size is 392000, then the maximum variation in chromosome can be 78400 by choosing a number randomly between 0 and 78400. The same method is applied for the whole population of chromosome. The generation of variable length chromosome is also shown in Algorithm 1. While training the data, for each chromosome (which represents the weights of an autoencoder) the root mean squared error (RMSE) is calculated of training dataset m. The fitness of each chromosome is defined by a fitness function [20] as:

$$fitness(i) = {}^{1}/_{\sqrt{\frac{\sum_{i=1}^{m}(x_i - \hat{x}_i)^2}{m}}} \tag{4}$$

After calculating the fitness of all chromosomes, the least fit chromosomes are removed from the population and update the remaining chromosomes using backpropagation. After removing the least fit chromosomes, we used Roulette selection method for the selection of parent chromosomes from the rest population and then use uniform crossover method to create offspring. The offspring is mutated from the best chromosomes using specified mutation probability. The mutation process is described in Algorithm 2. The whole process (Algorithm 3) is run for a specific number of iterations and returns the best chromosomes. The same process applied on all layers and stacked all the layers to make a deep autoencoder.

To classify the compressed dimensional feature vector, softmax regression is used. As we are concerned in multi-class classification so it takes k different values instead of two (as in binary classification) and the equation becomes:

Algorithm 1. Creation of Variable Length Chromosome

```
chromosomeSize ← input * hiddenNodes
variation ← ν * chromosomeSize
for i = 1 to populationSize do
    r1 = randi([1 variation])
    for j = 1 to r1 do
        r2 = randi([1 chromosomeSize])
        chromosome_i(j, r2) = 0
    end for
end for
```

$$h_\theta(x)^i = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix} \tag{5}$$

Here θ_1, θ_2, $\cdots$, θ_k are the parameters of model and $\sum_{j=1}^{k} e_j^T x^{(i)}$ normalizes the distribution, so that it sums to one. After training all the layers of deep autoencoder, the encoded data and labels pass to sofmax layer to train it with supervised fashion.

As backpropagation method are accountable for trapping at local minima. Our proposed method supports backpropagation in this regards, by decreasing the possibility of trapping at local minima. Moreover making the chromosomes (weights) variable produce sparser network (few active weights).

Algorithm 2. Mutation of Offspring

```
mutationRate = mutationProb * chromosomeSize
for i = 1 to mutationRate do
    r = randi([1 chromosomeSize])
    offspring(i r) = bestChromosome([i r])
end for
```

4 Experiments

4.1 Data

In order to access the performance of the proposed approach, MNIST handwritten digits dataset [21] is used in all experiments of all methods. Each sample in the dataset is a $28 * 28$ image having a grey scale value between 0–255. Moreover each sample holds a target classification label between 0–9, which is used in supervised classification. The training dataset contains 60,000 samples and testing dataset contains 10,000 samples.

Algorithm 3. Training of Layer l_i

initialize population of real values chromosomes.
convert chromosomes to variable length.
$epoch = 1$
while $(epoch >= MaxEpoch)$ **do**
  calculation of fitness of chromosomes.
  removing of least fit chromosomes.
  using backpropagation to update the best chromosomes.
  selection of parents chromosomes to produce an offspring.
  mutation of offspring.
  $epoch = epoch + 1$
end while
return best chromosome (*weights*).

4.2 Setup

MATLAB R2015a is used for the realization of code. In all experiments we used a deep neural network (autoencoder) of five layers. Initially the biases b_i^l are set to zero and weights W_{ij}^l are set to random numbers generated uniformly from the interval $\{-\sqrt{\frac{6}{(n_{in}+n_{hu}+1)}}, \sqrt{\frac{6}{(n_{in}+n_{hu}+1)}}\}$, where n_{in} is the number of inputs to the layer and n_{hu} is the number of neurons (units) in the layer. The first layer is the input layer consists of 784 units (neurons), followed by four hidden layers consisting of 500, 250, 100 and 50 units. Each layer is trained independently. First we train $784 - 500$ layer and used the output of that layer as input for the next $500 - 250$ layer. Secondly we train $500 - 250$ layer and used its output for the next layer input. Using the same manner we train the remaining layers ($250 - 100$ and $100 - 50$). We used sigmoid function (6) for activation of neuron.

$$f(z) = \frac{1}{1 + e^{-1}} \tag{6}$$

For VLC-GA implementation we used a population of 8 chromosomes. In each generation the 3 least fit chromosomes are substituted by the remaining 5 chromosomes offspring. We used a uniform crossover method and mutation probability of 0.01. For classification of the data we attach softmax regression classifier at the end of last layer and trained it for 100 generations. The parameters setup for all methods are summarized in the Table 1.

4.3 Results

To compare the performance of the proposed method with other methods we did the following experiments. In all experiments we set learning rate α and weight decay parameter λ to 0.5 and 0.003 respectively. Each result is the average of 10 experiments.

Table 1. Comparison of parameters used in all methods

Parameters	Proposed method	Method in [20]	AE using BP
Deep layers	5	5	5
Learning rate	0.5	0.5	0.5
Weight decay	0.003	0.003	0.003
Population size	8	10	-
Chromosome type	Variable	Constant	-
Selection method	Roulette wheel	Uniformly	-
Least fit chromosomes	3	5	-
Mutation	From best chromosome	Randomly	-
Crossover	Uniform	Uniform	-

Experiment-1. In this experiment we change the percentage of v to 10%, 20%, 30% and 40% in the process of generation of variable length chromosomes and then check its effect on classification accuracy. We used training dataset and testing dataset of $1,000$ and $10,000$ samples respectively. And train each layer for $1,000$ generations. The result of this experiment is shown in the Fig. 2-left. We got best accuracy at $v = 20\%$ so we used the same value of v in all experiments.

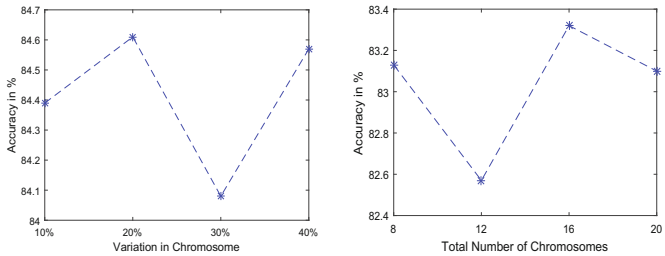

Fig. 2. Left-Classification accuracy of proposed method on different population size. **Right-**Classification accuracy of proposed method on different population size.

Experiment-2. In this experiment we used a population of 8, 12, 16 and 20 chromosomes in training of our method and check its effect on classification accuracy. This experiment is performed on 1,000 generations using subset of MNIST dataset. The subset created randomly by choosing 1,000 samples from training data. The result is shown in Fig. 2-right. As the number of chromosome increases the processing time also increases but there is no bigger change in classification accuracy that's why we used a population having less number of chromosomes (i.e. 8 chromosomes) in our experiments.

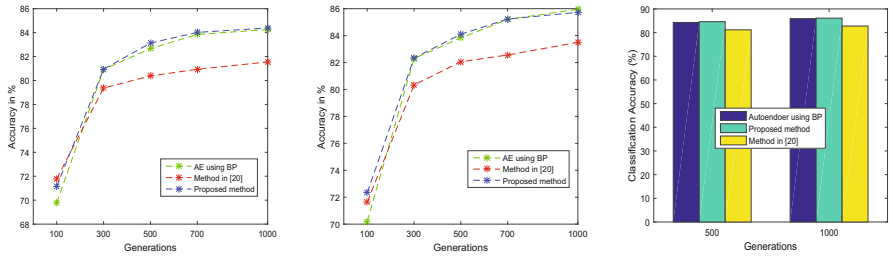

Fig. 3. Left-Comparison of classification accuracy of testing data using small training dataset. **Middle-**Comparison of classification accuracy of training data using small training dataset. **Right-**Comparison of classification accuracy of testing data using big training dataset.

Experiment-3. In this experiment we used small subset of MNIST dataset by randomly selecting 1,000 samples from training dataset. We train each method for 100, 300, 500, 700 and 1000 generations and then apply the classifier on the encoded data. The comparison of the classification accuracy of the testing dataset is shown in Fig. 3-left and the comparison of the classification accuracy of the training dataset is shown in Fig. 3-middle. On testing dataset our proposed method performs well on 300, 500, 700 and 1,000 generations but on 100 generations method in [20] perform better. On training dataset our method performs well on all generations expect 1,000 on which autoencoder using backpropagation method perform better.

Experiment-4. In this experiment we used the complete MNIST dataset (i.e. training dataset of 60,000 samples and the testing dataset of 10,000 samples). Firstly we train each method for 500 generations and compare the classification accuracy. Secondly we train each method for 1,000 generations and compare the classification accuracy. As shown in Fig. 3-right, on testing dataset our proposed method performs better than the other methods on both generations.

5 Conclusion

In this paper we presented a variable length chromosome genetic algorithm assisted deep autoencoder. We used roulette wheel selection method for the selection of parent chromosomes from the population. And the offspring are mutated from best parent chromosome. We used fewer chromosomes as compare to method in [20], this increase the processing speed of our method. According to results, our method improves the performance and produce sparser networks as compare to other methods. Though our implementation used an autoencoder, the same technique is applied to other forms of deep learning such as RBM, Convolutional Neural Networks (CNNs) etc. In future we will compare our work with more methods.

Acknowledgments. This work was supported by the National Natural Science Foundation of China (No. 61271374).

References

1. Bengio, Y., LeCun, Y., et al.: Scaling learning algorithms towards AI. Large-scale Kernel Mach. **34**, 1–41 (2007)
2. Utgoff Hinton, G.E., Osindero, S., Teh, Y.-W.: Many-layered learning. Neural Comput. **14**, 2497–2529 (2002). MIT Press
3. Hinton, G.E., Osindero, S., Teh, Y.-W.: A fast learning algorithm for deep belief nets. Neural Comput. **18**, 1527–1554 (2006). MIT Press
4. Freund, Y., Haussler, D.: Unsupervised learning of distributions of binary vectors using two layer networks, Computer Research Laboratory, University of California, Santa Cruz (1994)
5. Bengio, Y., Lamblin, P., Dan, P., et al.: Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems, vol. 19, p. 153. MIT Press (2007)
6. Ranzato, M., Poultney, C., Chopra, S., et al.: Efficient learning of sparse representations with an energy-based model. In: Proceedings of NIPS (2007)
7. Bengio, Y., et al.: Learning deep architectures for AI. In: Foundations and Trends in Machine Learning, vol. 2, pp. 1–127. Now Publishers, Inc. (2009)
8. Weston, J., Ratle, F., Mobahi, H., Collobert, R.: Deep learning via semi-supervised embedding. In: Montavon, G., Orr, G.B., Müller, K.-R. (eds.) Neural Networks: Tricks of the Trade. LNCS, vol. 7700, pp. 639–655. Springer, Heidelberg (2012). doi:10.1007/978-3-642-35289-8_34
9. Mobahi, H., Collobert, R., Weston, J.: Deep learning from temporal coherence in video. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 737–744. ACM (2009)
10. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks, vol. 313, pp. 504–507. American Association for the Advancement of Science (2006)
11. Vincent, P., Larochelle, H., Bengio, Y., et al.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th International Conference on Machine Learning, pp. 1096–1103. ACM (2008)
12. Ahmed, A., Yu, K., Xu, W., Gong, Y., Xing, E.: Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008. LNCS, vol. 5304, pp. 69–82. Springer, Heidelberg (2008). doi:10.1007/978-3-540-88690-7_6
13. Osindero, S., Hinton, G.E.: Modeling image patches with a directed hierarchy of Markov random fields. In: Advances in Neural Information Processing Systems, pp. 1121–1128 (2008)
14. Salakhutdinov, R., Mnih, A., Hinton, G.: Restricted Boltzmann machines for collaborative filtering. In: Proceedings of the 24th International Conference on Machine Learning, pp. 791–798. ACM (2007)
15. Hinton, G.E., Salakhutdinov, R.R.: Using deep belief nets to learn covariance kernels for Gaussian processes. In: Advances in Neural Information Processing Systems, pp. 1249–1256 (2008)
16. Levner, I.: Data Driven Object Segmentation. Citeseer (2009)
17. Mnih, A., Hinton, G.E.: A scalable hierarchical distributed language model. In: Advances in Neural Information Processing Systems, pp. 1081–1088 (2009)

18. Collobert, R., Weston, J.: A unified architecture for natural language processing: deep neural networks with multitask learning. In: Proceedings of the 25th International Conference on Machine Learning, pp. 160–167. ACM (2008)
19. Ranzato, M.A., Szummer, M.: Semi-supervised learning of compact document representations with deep networks. In: Proceedings of the 25th International Conference on Machine Learning, pp. 792–799. ACM (2008)
20. David, O.E., Greental, I.: Genetic algorithms for evolving deep neural networks. In: Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 1451–1452. ACM (2014)
21. LeCun, Y., Bottou, L., Bengio, Y., et al.: Gradient-based learning applied to document recognition. Proc. IEEE **86**, 2278–2324 (1998). IEEE
22. David, S.J., Whitley, D., Eshelman, L.J.: Combinations of genetic algorithms and neural networks: a survey of the state of the art. In: Combinations of Genetic Algorithms and Neural Networks, pp. 1–37. IEEE (1992)
23. Golberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Boston (1989)
24. Koehn, P.: Combining Genetic Algorithms and Neural Networks: The Encoding Problem. Citeseer (1994)
25. Schiffmann, W., Joost, M., Werner, R.: Application of genetic algorithms to the construction of topologies for multilayer perceptrons. In: Albrecht, R.F., Reeves, C.R., Steele, N.C. (eds.) Artificial Neural Nets and Genetic Algorithms, pp. 675–682. Springer, Wien (1993). doi:10.1007/978-3-7091-7533-0_98
26. Hancock, P.J.B., Smith, L.S.: Gannet: genetic design of a neural net for face recognition. In: Schwefel, H.-P., Männer, R. (eds.) PPSN 1990. LNCS, vol. 496, pp. 292–296. Springer, Heidelberg (1991). doi:10.1007/BFb0029766
27. Bishop, J.M., Bushnell, M.J., Usher, A., et al.: Genetic optimisation of neural network architectures for colour recipe prediction. In: Albrecht, R.F., Reeves, C.R., Steele, N.C. (eds.) Artificial Neural Nets and Genetic Algorithms, pp. 719–725. Springer, Wien (1993). doi:10.1007/978-3-7091-7533-0_104
28. Montana, D.J., Davis, L.: Training feedforward neural networks using genetic algorithms. In: IJCAI 1989, vol. 89, pp. 762–767 (1989)
29. Zhang, M., Deng, Y., Chang, D.: A novel genetic clustering algorithm with variable-length chromosome representation. In: Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 1483–1484. ACM (2014)
30. Yahya, A.A., Osman, A., Ramli, A.R., et al.: Feature selection for high dimensional data: an evolutionary filter approach. Citeseer (2011)
31. Salakhutdinov, R., Hinton, G.: Semantic hashing. Int. J. Approximate Reasoning **50**, 969–978 (2009). Elsevier
32. Brie, A.H., Morignot, P.: Genetic planning using variable length chromosomes. In: ICAPS 2005, pp. 320–329 (2005)
33. Baldi, P.: Autoencoders, unsupervised learning, and deep architectures. In: ICML Unsupervised and Transfer Learning, vol. 27, p. 1 (2012)

Springer

http://www.springer.com/978-3-319-70095-3

Neural Information Processing
24th International Conference, ICONIP 2017,
Guangzhou, China, November 14-18, 2017,
Proceedings, Part II
Liu, D.; Xie, S.; Li, Y.; Zhao, D.; El-Alfy, E.-S.M. (Eds.)

2017, XVIII, 926 p. 380 illus., Softcover
ISBN: 978-3-319-70095-3