An Elitist Genetic Algorithm Based Extreme Learning Machine

Vimala Alexander and Pethalakshmi Annamalai

Abstract Extreme Learning Machine (ELM) has been proved to be exceptionally fast and achieves more generalized performance for learning Single-hidden Layer Feedforward Neural networks (SLFN). In this paper, a Genetic Algorithm (GA) is proposed to choose the appropriate initial weights, biases and the number of hidden neurons which minimizes the classification error. The proposed GA incorporates a novel elitism approach to avoid local optimum and also speed up GA to satisfy the multi-modal function. The experimental results indicate the superior performance of the proposed algorithm with lower classification error.

Keywords Neural networks · Extreme learning machine · Genetic algorithm · Elitism

1 Introduction

Huang et al. [1] proposed a novel learning method called Extreme Learning Machine (ELM) to train Single-hidden Layer Feedforward Neural networks (SLFNs) faster than any other learning approaches. In case of classical learning algorithms, the learning process is done by iterative process which would be slower, however, the ELM simplifies the learning process in one step through a simple generalized inverse calculation [2]. Unlike other learning methods, ELM doesn't requires any parameters to be tuned except network architecture. However, ELM has its own limitation that the random selection of input weights and biases might

V. Alexander (✉)
Department of Computer Science, Fatima College, Madurai, Tamil Nadu, India
e-mail: cscfcvimala@gmail.com

P. Annamalai
Department of Computer Science, MVM Government Arts College (W), Dindigul, Tamil Nadu, India
e-mail: pethalakshmi@yahoo.com

© Springer Science+Business Media Singapore 2016

M. Senthilkumar et al. (eds.), *Computational Intelligence, Cyber Security and Computational Models*, Advances in Intelligent Systems and Computing 412, DOI 10.1007/978-981-10-0251-9_29

reduce the classification performance [3, 4]. Therefore, it is important to modify the ELM to overcome this inadequacy [5, 6]. Global searching methods such as Evolutionary Algorithms (EA) were used in the past decades for optimizing the weights and biases of the neural networks. In [7–11], Differential Evolution (DE) is adopted as a learning algorithm for SLFNs, but there is a chance that the DE algorithms may result in premature or slow convergence. Lahoz et al. [12] and Suresh et al. [13] used a Genetic Algorithm (GA) for optimizing the weights and number of hidden nodes.

This paper proposes a novel GA based learning for ELM. Genetic Algorithms (GAs) have proven useful in solving a variety of search and optimization problems. A major problem might occur in GA is that it might provide local optimum solution. The elitist strategy is widely adopted in the GAs' search processes to recover the chance of finding the global optimal solution. Elitism is able to preserve promising individuals from one generation to the next and maintain the diversity of the population [14, 15]. A novel elitist strategy is proposed in this paper, which improves the performance of the GA towards improving the classification accuracy of ELM.

The rest of the paper is organized as follows: Sect. 2 describes the background terminologies about ELM and GA, and explains the proposed EGA-ELM subsequently. Section 3 discusses the databases and parameter settings used for the experiments. Section 4 brings out the results and analyzes the performance of the proposed system with the existing ELM algorithms. Section 5 concludes the paper.

2 Materials and Methods

2.1 Basic ELM

ELM reduces the learning time of SLFNs by finding the weights using a simple inverse generalization calculation. Consider N arbitrary training samples (x_i, t_i), where x_i are the decision attributes, $x_i = [x_{i1}, x_{i2}, \ldots, x_{in}]^T \in R^n$ and ti are the target values or class attributes $t_i = [t_{i1}, t_{i2}, \ldots, t_{im}]^T \in R^m$. A typical SLFNs with $\tilde{N}$ number of hidden nodes and activation function $f(x)$ are defined as

$$\sum_{i=1}^{\tilde{N}} \alpha_i f_i(x_j) = \sum_{i=1}^{\tilde{N}} \alpha_i f(w_i.x_j + b_i) = o_j, j = 1, \ldots, N \tag{1}$$

where $w_i = [w_{i1}, w_{12}, \ldots, w_{in}]^T$ is the weight matrix between *i*th hidden and input nodes, $\alpha_i = [\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{im}]^T$ is the weight matrix between *i*th hidden and output nodes, and b_i is the bias value of the *i*th hidden node. $w_i \cdot x_j$ denotes the inner product of w_i and x_j.

The classical learning algorithm repeats the learning process till the network error mean becomes zero, that is

$$\sum_{j=1}^{\tilde{\mathrm{N}}} \left\| o_j - t_j \right\| \cong 0 \tag{2}$$

The target value is the mapping between the weights and the activation function as defined below

$$\sum_{i=1}^{\tilde{\mathrm{N}}} \alpha_i f\left(w_i \cdot x_j + b_i\right) = t_j, j = 1, \ldots, N \tag{3}$$

The above N equation could be simplified as Hα = T, where

$$H\left(w_1, \ldots, w_{\tilde{\mathrm{N}}}, b_1, \ldots, b_{\tilde{\mathrm{N}}}, x_1, \ldots, x_N\right) = \begin{bmatrix} f(w_1 \cdot x_1 + b_1) & \cdots & f(w_{\tilde{\mathrm{N}}} \cdot x_1 + b_{\tilde{\mathrm{N}}}) \\ \vdots & \ldots & \vdots \\ f(w_1 \cdot x_N + b_1) & \cdots & f(w_{\tilde{\mathrm{N}}} \cdot x_N + b_{\tilde{\mathrm{N}}}) \end{bmatrix}_{N \times \tilde{\mathrm{N}}} \tag{4}$$

where H is called the hidden layer output matrix of the neural network [16, 17]; *i*th column of H is the ith hidden node output with respect to inputs x_1, x_2, …, x_N. If the activation function f is infinitely differentiable we can prove that the required number of hidden nodes $\tilde{\mathrm{N}} \leq \mathrm{N}$. The main objective of any learning algorithms is to find the specific $\hat{w}_i, \hat{b}_i, \hat{\alpha}$ to reduce the network mean error, defined as

$$\left\| H\left(\hat{w}_1, \ldots, \hat{w}_{\tilde{\mathrm{N}}}, \hat{b}_1, \ldots, \hat{b}_{\tilde{\mathrm{N}}}\right)\hat{\alpha} - T \right\| = \min_{w_i, b_i, \alpha} \left\| H\left(\hat{w}_1, \ldots, \hat{w}_{\tilde{\mathrm{N}}}, \hat{b}_1, \ldots, \hat{b}_{\tilde{\mathrm{N}}}\right)\hat{\alpha} - T \right\| \tag{5}$$

This can be written as a cost function

$$E = \sum_{j=1}^{N} \left(\sum_{i=1}^{\tilde{\mathrm{N}}} \alpha_i f\left(w_i \cdot x_j + b_i\right) - t_j \right)^2 \tag{6}$$

The gradient-descent learning algorithms generally search for the minimum of $H\beta - T$, for that the weights and biases are iteratively adjusted as follows:

$$W_k = W_{k-1} - \eta \frac{\partial E(W)}{\partial W} \tag{7}$$

Here η is a learning rate. BP is one of the popular learning algorithm for SLFNs, which has the following issues [18, 19]:

- The learning rate has to be carefully chosen, the minor η slows down the learning algorithm, and however, if it is too large then the learning becomes unstable.

- Training with more number of iteration leads to inferior performance.
- Gradient-descent based learning is very time-consuming in most applications.

According to Huang et al. [2] the w_i and b_i of SLFNs could be assigned with random values rather than iteratively adjusting them, and the output weights can be estimated as

$$\hat{\alpha} = H^{\dagger}T \tag{8}$$

where $H^{\dagger}$ is the Moore–Penrose (MP) generalized inverse [20] of the hidden layer output matrix H. There are several methods to calculate the MP generalized inverse of a matrix, among that the Singular Value Decomposition (SVD) is known for the best inverse and thus is used in most of the ELM implementations [18, 21]. The ELM can be summarized as follows:

Pseudo code for Basic ELM

Given training set N samples, output function $f(x)$ and $\tilde{N}$ hidden nodes.

1. Randomly assign input weights w_i and bias b_i, $I = 1, \ldots, \tilde{N}$.
2. Calculate the hidden layer output matrix H.
3. Calculate the output weight $\hat{\alpha} = H^{\dagger}T$

2.2 *Genetic Algorithm*

Holland [22] introduces the basic principle of Genetic Algorithm (GA). GA is one of the best known search and optimization methods widely used to solve complex problems [23, 24]. The detailed description of the real coded GA is given in this section.

Procedure Standard GA

```
t ← 1          // iteration number
P ← initial population with randomly generated real values
Calculate the fitness value of each parent in P, g(P)
while (not termination condition) do
    t ← t + 1
    C ← { }      // Initialize children population
    While |C| < |P| do
       Select a pair of parents for crossover
       Mate the parents to create children c1 and c2
       C ← C ∪ { c1, c2 } and perform mutation
    end
    P ← C, and evaluate P, g(P)
end        // Next generation
```

Initial Population. The initial population is a possible solution set P, which is a set of real values generated randomly, $P = \{p_1, p_2, \ldots, p_s\}$.

Evaluation. A fitness function should be defined to evaluate each chromosome in the population, can be written as fitness = g(P).

Selection. After the fitness value is calculated, the chromosomes are sorted based on their fitness values, then the tournament selection method is widely used for parent selection.

Genetic Operators. The genetic operators are used to construct some new chromosomes (offspring) from their parents after the selection process. They include the crossover and mutation [25]. The crossover operation is applied to exchange the information between two parents, which are selected earlier. There are number of crossover operators are available, here we use arithmetical crossover. The crossed offspring will then apply with mutation operation, to change the genes of the chromosomes. Here we have used uniform mutation operation.

After the operations of selection, crossover and mutation, a new population is generated. The next iteration will continue with this new population and repeat the process. Each time the best chromosomes from the current or new population is selected and compared with the best from the previous population to maintain the global best chromosome. This iterative process could be stopped either the result converged or the number of iterations exceeds the maximum limit.

Elitism—The genetic operations such as crossover and mutation may produces weaker children than the parents. But the Evolutionary Algorithms (EA) could recover from this problem after consequent iterations but there is no assurance. Elitism is the known solution to solve this problem. The basic idea is to retain a copy of best parents from the previous population to the subsequent populations. Those parents are copied without changing them, helps to recover EA from re-discovering weaker children. Those elitist parents are still eligible to act as parents for the crossover and mutation operations [26]. Elitism can be implemented by producing only (s—ep) children in each generation, where ‘s’ is the population size and ‘ep’ is the user-defined number of elite parents. In this case, at each iteration, after the evaluation step, the parents are sorted based on their fitness values, and ‘ep’ number of best parents is selected for next generation, and only (s—ep) number of children are reproduced with genetic operations. Then the next iteration is continued with the population contains union of elitist parents and the children. The following pseudo code illustrates the elitist strategy.

```
Procedure Elitist GA
// Initialization
while (not termination condition) do
    t ← t + 1
    Elites ← Best 'ep' parents
    C ← { }      // Initialize children population
    While |C| < |P| – ep   do
      Select a pair of parents for crossover
      Mate the parents to create children c1 and c2
      C ← C ∪ { c1, c2 } and perform mutation
    end
    P ← C ∪ Elites and evaluate P
end        // Next generation
```

2.3 The Proposed Elitism Strategy

In classical elitist strategy, the number of elites are fixed and continued till the genetic algorithm terminates. Our idea is to make 'ep' as a variable depends on the age of the population. And it is noted that, the elitist parents what we are assuming at the initial iterations may not the best parents globally, this point makes the proposed approach to keep increasing the number of elites while the iteration grows. Initially we started less number of elite parents (for example 10 % of the population size), over a period of time, the number of elites are gradually increased to reach up to 50 % of the population size. The increment is done with a constant value scheduled to happen at regular interval, say after some 'q' number of iterations.

2.4 Elitist Genetic Algorithm Based ELM

The proposed Elitist GA (EGA) strategy is applied to find the optimal weights between input and hidden layers and bias values. The weights has the bound of [−1, 1], so the lbound and ubound takes the values −1 and 1 respectively. The population dimension depends on the number of input variable to ELM. Initially a population is generated with a set of random numbers and these values are fed to ELM and find the classification accuracy as fitness for each chromosome. Then the genetic iteration is continued with the new population as discussed earlier with genetically optimized weights. After termination, the proposed algorithm will provide the optimum weights and bias values for the ELM.

3 Experimental Setup

The performance of the proposed EGA-ELM for SLFN is evaluated on the benchmark datasets described in Table 1, which includes five classification applications. The datasets are received from UCI machine learning repository. Table 2 summarizes the ELM and Elitist GA parameter values used for the experiments.

4 Results and Discussions

EGA-ELM performance is compared with the evolutionary based ELMs such as SaE-ELM [10], E-ELM [9], LM-SLFN [8], and compared with general Elitist GA based ELM (Elitist-ELM), also with traditional ELM [1], based on classification accuracy measure. Table 3 summarizes the classification accuracy received from each ELM algorithms on different datasets. The results clearly indicate that the proposed EGA-ELM outperforms other ELM approaches in terms of higher classification accuracy. Comparatively, only the EGA-ELM and the Elitist-ELM improves the classification accuracy consistently, the rest of the algorithms' performance oscillates between a ranges, this way the proposed algorithm proves its stable performance.

Table 1 The datasets used for performance evaluation

Datasets	#Attributes	#Classes	#Samples
Breast cancer Wisconsin	10	2	699
Ecoli	8	6	336
Iris	4	3	150
Parkinsons	23	3	197
Pima	8	2	768

Table 2 Summary of Ex-ELM and elitist GA parameters settings

Ex-ELM		Genetic algorithm	
Parameters	Values	Parameters	Values
#Input neurons	#Input attributes	Number of iterations	1000
#Hidden neurons	Ñ	Crossover probability	0.8
Bias values	0 to 1	Mutation probability	0.01
Input weights	−1 to 1	Population size	100
Target outputs range	−1 to 1	Initial #elite parents	10
#Output neurons	#Class values		
Activation function	Sigmoid		
#Hidden layers	1 to 20		
Hidden layer weights	−1 to 1		

Table 3 Summary of classification accuracy from ELM algorithms

Datasets	Testing accuracy					
	EGA-ELM	Elitist-ELM	SaE-ELM	E-ELM	LM-SLFN	ELM
WBC	89.54	89.01	87.24	87.20	87.15	87.39
Ecoli	96.73	96.13	94.96	94.92	95.07	94.86
Iris	98.91	98.75	98.31	98.35	98.40	98.34
Parkinsons	90.36	89.36	87.22	86.52	86.77	86.87
Pima	82.07	80.67	77.69	77.49	77.55	77.38

5 Conclusions

Extreme Learning Machine (ELM) is one of the fastest learning algorithms used for classification and regression. One of the limitations of ELM is the random choice of input weights, might be results in lower prediction accuracy. Evolutionary algorithms have been implemented to optimize those weights. This paper proposed a novel Elitism based Genetic Algorithm (GA) based weight optimization strategy for ELM. The performance of the proposed algorithms is tested with five datasets from UCI machine learning repository against four other different evolutionary and traditional ELM algorithms. The result shows that the proposed EGA-ELM algorithm surpasses the other algorithms with superior classification accuracy in stable.

References

1. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. Neurocomputing **70**(1), 489–501 (2006)
2. Huang, G., Huang, G.B., Song, S., You, K.: Trends in extreme learning machines: a review. Neural Networks. **61**, 32–48 (2015)
3. Wang, Y., Cao, F., Yuan, Y.: A study on effectiveness of extreme learning machine. Neurocomputing **74**(16), 2483–2490 (2011)
4. Bartlett, P.L.: The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. IEEE Trans. Inf. Theory **44**(2), 525–536 (1998)
5. Levenberg, K.: A method for the solution of certain nonlinear problems in least squares. Quart. Appl. Math. **2**, 164–168 (1994)
6. Hagan, M.T., Menhaj, M.B.: Training feedforward networks with the Marquardt algorithm. IEEE Trans. Neural Networks **5**(6), 989–993 (1994)
7. Ilonen, J., Kamarainen, J.K., Lampinen, J.: Differential evolution training algorithm for feed-forward neural networks. Neural Process. Lett. **17**(1), 93–105 (2003)
8. Subudhi, B., Jena, D.: Differential evolution and Levenberg Marquardt trained neural network scheme for nonlinear system identification. Neural Process. Lett. **27**(3), 285–296 (2008)
9. Zhu, Q.Y., Qin, A.K., Suganthan, P.N., Huang, G.B.: Evolutionary extreme learning machine. Pattern Recogn. **38**(10), 1759–1763 (2005)
10. Cao, J., Lin, Z., Huang, G.B.: Self-adaptive evolutionary extreme learning machine. Neural Process. Lett. **36**(3), 285–305 (2012)

11. Qin, A.K., Huang, V.L., Suganthan, P.N.: Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans. Evol. Comput. **13**(2), 398–417 (2009)
12. Lahoz, D., Lacruz, B., Mateo, P.M.: A multi-objective micro genetic ELM algorithm. Neuro-computing **111**, 90–103 (2013)
13. Suresh, S., Babu, R.V., Kim, H.J.: No-reference image quality assessment using modified extreme learning machine classifier. Appl. Soft Comput. **9**(2), 541–552 (2009)
14. Li, J.P., Balazs, M.E., Parks, G.T., Clarkson, P.J.: A species conserving genetic algorithm for multimodal function optimization. Evol. Comput. **11**(1), 107–109 (2003)
15. Liang, Y., Leung, K.S.: Genetic Algorithm with adaptive elitist-population strategies for multimodal function optimization. Appl. Soft Comput. **11**(2), 2017–2034 (2011)
16. Huang, G.B., Babri, H.A.: Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. IEEE Trans. Neural Networks **9**(1), 224–229 (1998)
17. Huang, G.B.: Learning capability and storage capacity of two-hidden-layer feedforward networks. IEEE Trans. Neural Networks **14**(2), 274–281 (2003)
18. Huang, G.B., Chen, L., Siew, C.K.: Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans. Neural Networks **17**(4), 879–892 (2006)
19. Hykin, S.: Neural Networks: A Comprehensive Foundation. Printice-Hall. Inc., NJ (1999)
20. Rao, C.R., Mitra, S.K.: Generalized Inverse of Matrices and Its Aplications, vol. 7. Wiley, New York (1971)
21. Corwin, E.M., Logar, A.M., Oldham, W.J.: An iterative method for training multilayer networks with threshold functions. IEEE Trans. Neural Networks **5**(3), 507–508 (1994)
22. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press (1992)
23. Mohamed, M.H.: Rules extraction from constructively trained neural networks based on genetic algorithms. Neurocomputing **74**(17), 3180–3192 (2011)
24. Bi, C.: Deterministic local alignment methods improved by a simple genetic algorithm. Neurocomputing **73**(13), 2394–2406 (2010)
25. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, pp. 111–112. Springer Science & Business Media (1996)
26. Simon, D.: Evolutionary Optimization Algorithms, pp. 188–189. Wiley (2013)