Engineering Applications of Artificial Intelligence 23 (2010) 1149–1157

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Performance enhancement of extreme learning machine for multi-category sparse data classification problems

S. Suresh [a,*], S. Saraswathi [b], N. Sundararajan [c]

[a] School of Computer Engineering, Nanyang Technological University, Singapore
[b] Laurence H. Baker Center for Bioinformatics and Biological Statistics, Iowa State University, Ames, USA
[c] School of Electrical and Electronics Engineering, Nanyang Technological University, Singapore

ARTICLE INFO

Article history:
Received 18 December 2008
Received in revised form
4 January 2010
Accepted 12 June 2010
Available online 1 July 2010

Keywords:
Neural network
Extreme learning machine
K-fold validation
Genetic algorithm
Multi-category sparse classification
Micro-array gene expression data

ABSTRACT

This paper presents a performance enhancement scheme for the recently developed extreme learning machine (ELM) for multi-category sparse data classification problems. ELM is a single hidden layer neural network with good generalization capabilities and extremely fast learning capacity. In ELM, the input weights are randomly chosen and the output weights are analytically calculated. The generalization performance of the ELM algorithm for sparse data classification problem depends critically on three free parameters. They are, the number of hidden neurons, the input weights and the bias values which need to be optimally chosen. Selection of these parameters for the best performance of ELM involves a complex optimization problem.

In this paper, we present a new, real-coded genetic algorithm approach called 'RCGA-ELM' to select the optimal number of hidden neurons, input weights and bias values which results in better performance. Two new genetic operators called 'network based operator' and 'weight based operator' are proposed to find a compact network with higher generalization performance. We also present an alternate and less computationally intensive approach called 'sparse-ELM'. Sparse-ELM searches for the best parameters of ELM using *K*-fold validation. A multi-class human cancer classification problem using micro-array gene expression data (which is sparse), is used for evaluating the performance of the two schemes. Results indicate that the proposed RCGA-ELM and sparse-ELM significantly improve ELM performance for sparse multi-category classification problems.

1. Introduction

In the last few decades, extensive research has been carried out in developing the theory and applications of artificial neural networks (ANN). ANNs possess an inherent structure suitable for mapping complex characteristics, learning and optimization and have emerged as a powerful tool for solving various practical problems, like pattern classification and recognition, medical imaging, speech recognition and control. From a practical perspective, the massive parallelism and fast adaptability of neural network implementations provide more incentives for further investigation into problems with uncertainties which involve complex mapping. Of the many neural network architectures proposed, single layer feedforward network (SLFN) with sigmoidal or radial basis activation function is found to be effective in solving a number of real world problems. Normally, the free parameters of the network are learnt from the given training samples using gradient descent algorithms. The gradient descent algorithms are relatively slow and have many issues related to its convergence.

Recently in Huang et al. (2006a, 2004) and Huang (2003), it is shown that the SLFN network (with a sufficient number of H hidden neurons) with randomly chosen input weights and hidden bias can approximate any continuous function to any desirable accuracy. Here, the output weights are analytically calculated using the Moore–Penrose generalized pseudo-inverse. Since the learning algorithm is faster and has a good generalization ability, it is called as 'extreme learning machine' (ELM). The ELM algorithm overcomes many issues encountered in traditional gradient descent algorithms such as stopping criterion, learning rate, number of epochs and local minima. In fact, the superior performance of the ELM algorithm compared to other existing neural network approaches has been shown in relation to many real world problems (Huang, 2003; Huang et al., 2004, 2006b). In Huang et al. (2006b), real time learning capabilities of the single hidden layer network are presented. The universal approximation abilities of such network are demonstrated in Huang et al. (2006c).

In this paper, we show that the generalization performance of the ELM algorithm for sparse data classification problem depends

* Corresponding author.
E-mail addresses: sundaram.suresh@hotmail.com (S. Suresh), saras@iastate.edu (S. Saraswathi), ensundara@ntu.edu.sg (N. Sundararajan).

0952-1976/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2010.06.009

on the proper selection of the input weights and hidden bias values (fixed parameters) and the number of hidden neurons. Particularly, the impact of these three parameters on the generalization performance is very significant for sparse data classification problems which have fewer number of training samples and a high imbalance in the number training samples per class. We present a real coded genetic algorithm based ELM called 'RCGA-ELM' for selecting the optimal number of hidden neurons and its corresponding input weights and bias values for maximizing its classification performance.

The proposed RCGA-ELM is different from an earlier evolutionary ELM (E-ELM) presented in Zhu et al. (2005). In evolutionary ELM (E-ELM), given a certain number of hidden neurons, the algorithm searches for the best input weights and hidden bias values that optimizes its performances. In Zhu et al. (2005), a separate exhaustive search technique is used to find the appropriate number of hidden neurons, resulting in a computationally intensive scheme. However, the proposed RCGA-ELM algorithm searches for all the three parameters, simultaneously. RCGA-ELM algorithm uses two different types of genetic operators namely, 'weight based operators' and 'network based operators', to find the best number of hidden neurons and its corresponding connection weights. The network based operator controls the neuron growth and the weight based operator searches for the optimal weights. Even though the RCGA-ELM finds a compact network with higher generalization efficiency, RCGA-ELM is still computationally intensive. Hence, we proposed an alternate approach called 'sparse-ELM' (S-ELM) which is based on K-fold validation. S-ELM can quickly select the hidden neurons, input weights and bias values which enables the network to achieve better generalization performance within a shorter time.

The performance of the proposed S-ELM and RCGA-ELM schemes is evaluated on a real world problem of human cancer classification using the global cancer mapping (GCM) micro-array gene expression data set (Ramaswamy et al., 2002). This data are characterized by very few samples and a high imbalance in the number of samples per class. GCM data set has 198 samples from 14 different types of cancers with each sample having 16 063 features. The presence of few samples per class, high sample imbalance (imbalance in the number of samples in each class) and large number of input features increase the complexity of the classifier development.

The paper is organized as follows: In Section 2, we present a brief review of ELM and the issues related to ELM for sparse data classification. Section 3 describes the proposed RCGA-ELM and Section 4 describes the sparse-ELM scheme used for selecting the best parameters. Section 5 presents a detailed description of the GCM data set based cancer classification. Section 6 highlights the detailed performance comparison results for the above problem and Section 7 summarizes the conclusions from this study.

2. A brief review of extreme learning machine (ELM)

In this section, we present a brief overview of the extreme learning machine (ELM) algorithm (Huang et al., 2006a) and the problems faced in using ELM for sparse data classification. ELM is a single hidden layer feedforward network where the input weights are chosen randomly and the output weights are calculated analytically. For hidden neurons, many activation functions such as sigmoidal, sine, Gaussian and hard-limiting function can be used and the output neurons have a linear activation function. ELM uses non-differentiable or even discontinuous functions as an activation function.

In general, a multi-category classification problem can be stated in the following manner. Suppose, we have N observation samples $\{X_i,Y_i\}$, where $X_i=[x_{i1},\ldots,x_{in}]\in\Re^n$ is an n-dimensional feature of the sample i and $Y_i=[y_{i1},y_{i2},\ldots,y_{iC}]\in\Re^C$ is its coded class label. If the sample X_i is assigned to the class label c_k then kth element of Y_i is one ($y_{ik}=1$) and other elements are -1. Here, we assume that the samples belong to C distinct classes. The function which gives the necessary information on the probability of predicting the class label with the desired accuracy is called a classifier function and is defined as $Y=F(X)$. Given a known set of samples, the objective of the classification problem is to estimate the functional relationship between the random samples and their class labels.

Using universal approximation property, one can say that the single layer feedforward network with sufficient number of hidden neurons H can approximate any function to any arbitrary level of accuracy (Huang, 2003). It implies that for bounded inputs to the network, there exist optimal weights (not necessarily unique) to approximate the function. Let W be $H\times n$ input weights, B be $H\times 1$ bias of hidden neurons and V be $C\times H$ output weights. The output ($\hat{Y}$) of the ELM network with H hidden neurons has the following form:

$$\hat{y}_{ik}=\sum_{j=1}^{H}V_{kj}G_j(W,B,X_i),\quad k=1,2,\ldots,C \tag{1}$$

where $G_j(\cdot)$ is the output of the jth hidden neuron and is defined as

$$G_j=G\left(\sum_{k=1}^{n}W_{jk}x_{ik}+b_j\right),\quad j=1,2,\ldots,H \tag{2}$$

where $G(\cdot)$ is the activation function.

In case of radial basis function (RBF), the output of the jth Gaussian neuron $G_j(\cdot)$ is defined as

$$G_j=G(b_j\|X-W\|),\quad j=1,2,\ldots,H \tag{3}$$

where W and b_j ($b_j\in\Re^+$) are the center and width of the RBF neuron.

Eq. (1) can be written in matrix form as

$$\hat{Y}=VY_H \tag{4}$$

where

$$Y_H(W,B,X)=\begin{bmatrix}G_1(W,b_1,X_1) & G_1(W,b_1,X_2) & \cdots & G_1(W,b_1,X_N)\\ \vdots & \vdots & \vdots & \vdots\\ G_H(W,b_H,X_1) & G_H(W,b_H,X_2) & \cdots & G_H(W,b_H,X_N)\end{bmatrix}$$

Here, Y_H (is of dimension $H\times N$) is called the hidden layer output matrix of the neural network; the ith row of Y_H is the ith hidden neuron output with respect to inputs $X_1,X_2,\ldots,X_N$. For most practical problems, it is assumed that the number of hidden neurons are always less than the number of samples in the training set.

In ELM algorithm, for a given number of hidden neurons, it is assumed that the input weights W and bias B of hidden neurons are selected randomly, i.e., the elements of W and B are real numbers. By assuming the predicted output $\hat{Y}$ is equal to the coded labels Y, the output weights are estimated analytically as

$$\hat{V}=YY_H^{\dagger} \tag{5}$$

where $Y_H^{\dagger}$ is the Moore–Penrose generalized pseudo-inverse of the hidden layer output matrix.

In summary, the following are the steps involved in the ELM algorithm:

- For a given training samples (X_i,Y_i), select the appropriate activation function $G(\cdot)$ and the number of hidden neurons H.
- Select the input weights W and bias B randomly. Then, calculate the output weights V analytically: $V=YY_H^{\dagger}$.

2.1. Issues in ELM for sparse data classification problems

Multilayer perceptron networks, which are trained using a back propagation algorithm, search for optimal W, B and V for a given number of hidden neurons, such that the approximation error is minimum, i.e.,

$$\min_{W,B,V} \|VY_H - Y\| \tag{6}$$

which is equivalent to minimize the mean squared error $E = \sum_{i=1}^{N} (\hat{Y}_i - Y_i)^2$. Here, gradient descent algorithms are used to iteratively adjust the free parameters.

Similarly, in ELM, the output weights are found analytically by minimizing the errors for randomly selected W, B and number of hidden neurons, i.e.,

$$\min_{V} \|VY_H(W,B,X) - Y\| \tag{7}$$

Here, one has to find the optimal number of hidden neurons H and the corresponding W and B values to analytically calculate the V such that the generalization ability of the ELM network can be improved. Now, we pose this problem as an optimization problem.

Given: The input–output data and an activation function.

Find: Optimal number of hidden neurons (H^*), corresponding input weights (W) and hidden bias values (B) such that the ELM network with analytically calculated output weights has better generalization performance.

We know that for a randomly fixed W and B, one can directly use Eq. (2) to find the optimal output weights in the least square sense. However, the generalization performance of the ELM algorithm for sparse data classification problem depends on the proper selection of the fixed parameters (H, W and B). Particularly, the selection of the fixed parameters affect the generalization performance considerably in classification problems, where fewer number of training samples and high imbalance in the number of training samples per class are present. To illustrate this clearly, we consider a human cancer classification problem using micro-array gene expression data (Ramaswamy et al., 2002).

The GCM data are a collection of samples consisting of 14 different types of cancers. There are 190 samples with each sample having 16 063 features. From the remaining 190 samples, 144 are selected for training and 46 for testing. Here, we use a recursive feature elimination scheme to select 98 genes as explained in Ramaswamy et al. (2002). For more details, one should refer to the supplementary material provided along with the original work in Ramaswamy et al. (2002). The details on micro-array data will be presented in the Section 5.

We present a simulation study to analyze the behavior of ELM algorithm under sparse data condition using the GCM data. For our simulation study, we consider an ELM network with 40 hidden neurons. We call the ELM algorithm 100 times for the same training/testing data and network configuration and find the mean and standard deviation of training and testing accuracies. Each time the ELM algorithms is called, the fixed parameters (input weights and bias of hidden neurons) are initialized randomly using a uniform distribution. The input data are normalized between 0 and 1 and the weights and bias are initialized between ± 1. In this study, we use a unipolar sigmoidal function as the activation function ($1/(1+e^{-\lambda u})$) for the hidden neurons. The slope (λ) of the sigmoidal function is selected as 0.01 (approximately equal to the inverse of the number of features or number of input neurons). The mean and standard deviation of the training efficiency are 94.3% and 1.41, respectively. Similarly, the mean and standard deviation of the testing efficiency are 64.5% and 5.3, respectively. The variations in the training and testing efficiencies for different runs are illustrated in Fig. 1. From this figure, we can infer that the random selection of fixed parameters results in varied performances of the ELM classifier and affects the results significantly.

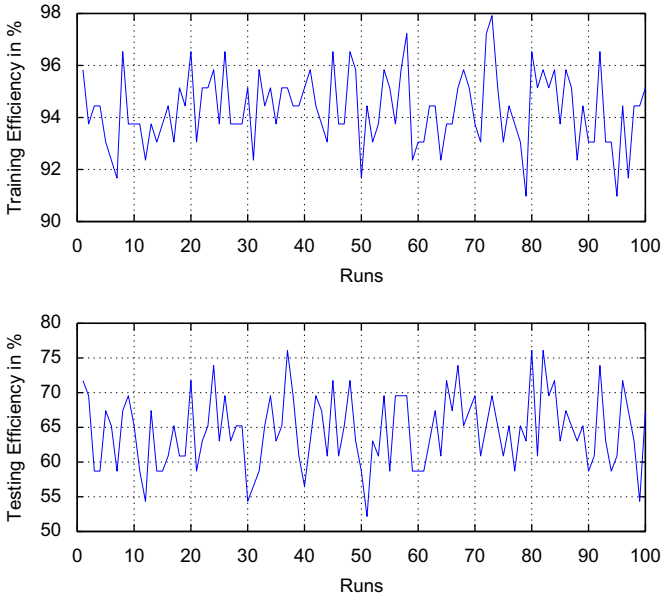

Fig. 1. Effect of initial parameter selection in ELM classifier on training/testing performances.

In addition, the behavior of ELM classifier with respect to the initial parameters changes considerably with the number of hidden neurons. To illustrate this behavior, we conducted an experimental study by varying the number of hidden neuron from 20 to 100 in steps of 10. The variation in the training and testing accuracies are given in Figs. 2(a and b). The mean training and testing efficiencies for various initial parameters are calculated for different number of hidden neurons. The variation of the mean training and testing efficiency with respect to hidden neurons is shown in Fig. 3. The standard deviation is 2% in the case of training and approximately 6% for testing. The standard deviation for training/testing at different number of hidden neurons are almost the same. From Fig. 2(a), we can see that the training efficiency increases with increase in number of hidden neurons and reaches a maximum at 80 hidden neurons. In addition, the variation in efficiency for different initial parameters decreases with increase in number of hidden neurons. From Fig. 2(b), we can observe that the testing/generalization efficiency increases up to certain number of hidden neurons and start decreasing afterwards. The same can be easily seen from the mean testing efficiency plot in Fig. 3. The testing efficiency reaches a maximum when the number of hidden neurons are between 40 and 60. During this interval, the training efficiency also reaches maximum at some random runs. When the number of hidden neurons reaches 40, the variation in performance with respect to initial parameters also increases considerably.

Since there are many choices of values to pick from in the generalization efficiency surface, it is difficult to find the best parameters (H, W and B) such that the training and generalization efficiency are maximized. In addition, the problem of finding the best set of parameters for ELM is a complex problem. One has to search for the optimal number of hidden neurons, input weights and bias suitable for the given problem. Specifically, this is the problem addressed in this paper. Here, two different approaches are presented for selecting ELM parameters such that the performance of the classifier is optimal under sparse data condition. First, we present an approach using a real-coded

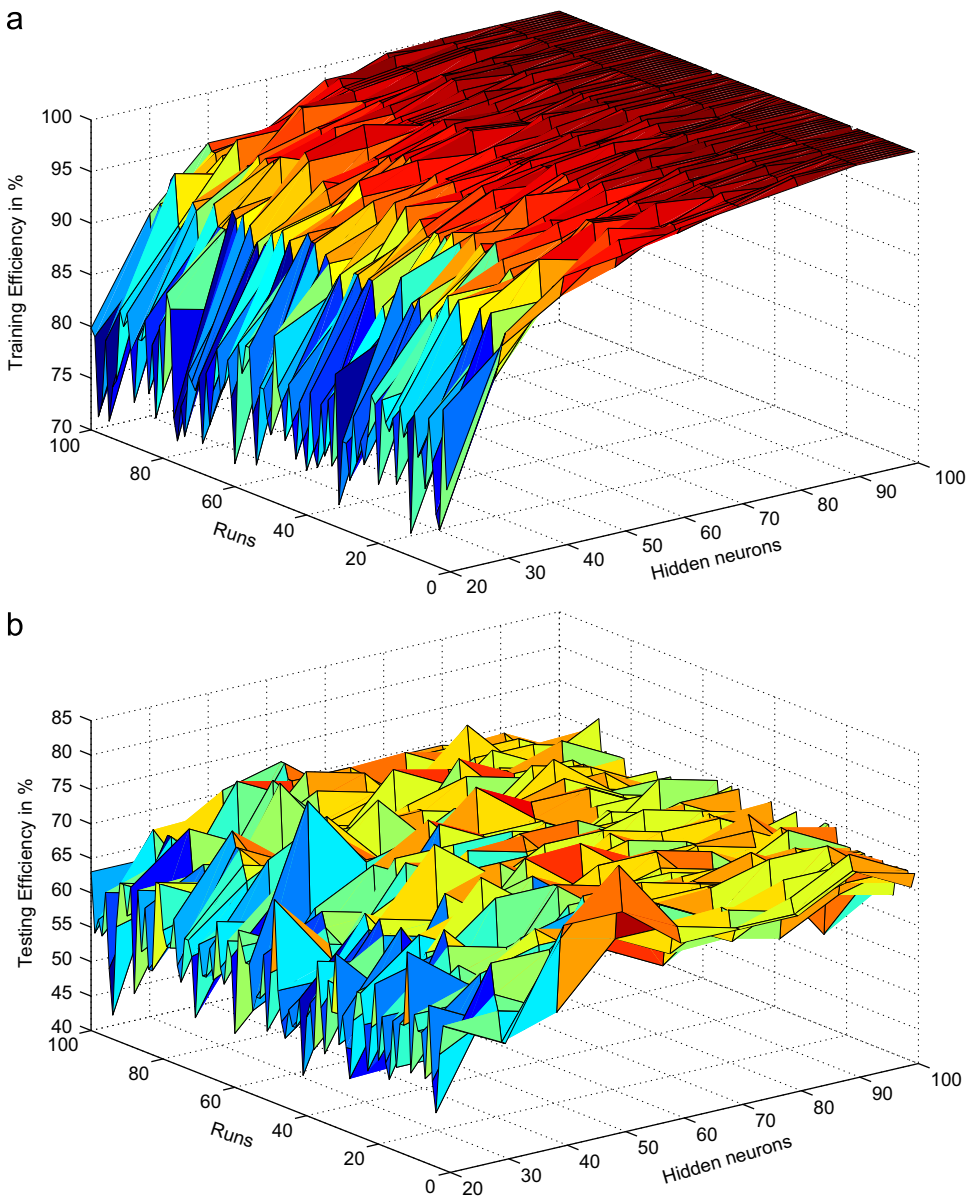

Fig. 2. ELM classifier training and testing efficiencies variation with respect to hidden neurons and initial parameters: (a) training results and (b) testing results.

genetic algorithm for selecting the best H, W and B values for sparse data classification. Next, we present less computationally intensive sparse-ELM approach using K-fold cross-validation to find the optimal number of neurons, input weights and hidden bias values of the ELM.

3. A real-coded genetic algorithm approach

The real-coded genetic algorithm (RCGA) is perhaps the most well-known of all evolution based search techniques (Michalewicz, 1994). Genetic algorithms are widely used to solve complex optimization problems where the number of parameters and constraints are large and the analytical solutions are difficult to obtain. In recent years, many schemes for combining genetic algorithms and neural networks have been proposed and tested. A detailed survey on evolving neural networks using genetic algorithms can be found in Schaffer et al. (1992).

In this paper, we use the hybrid real coded genetic algorithm approach for hidden neuron selection and its corresponding input weights and bias values. In this approach, two different genetic operators are used. The network based genetic operator controls the number of hidden neurons and the weight based genetic operator evolves the input weight and bias values. Since, the input weights and bias values are in a continuous domain (real value), real numbers are used to represent them as strings.

A real coded genetic algorithm for any particular optimization problem must have the following components:

- string representation,
- population initialization,
- selection function,
- genetic operators,
- fitness function and
- termination function.

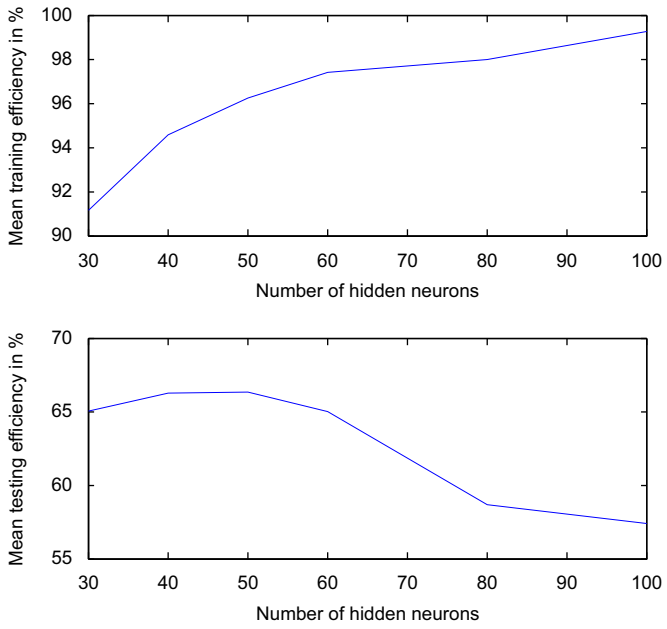

Fig. 3. Variation of mean training and testing efficiencies of ELM classifier with respect to number of hidden neurons.

Now, we describe the components of the real coded genetic algorithm which are used in designing the ELM network.

3.1. String representation

String representation is the process of encoding a potential search node (solution) as a string. The string representation depends on the structure of the problem in a genetic algorithm framework and on the genetic operators used in the algorithm. In our studies, the string representation for search node, is a string of real numbers. The real numbers represent the input weights (weight connection between the input neurons and the hidden neurons) of the single layer feedforward neural network. The weights and bias values are represented in the string as a two dimensional real array. The size of the array depends on the number of hidden neurons (rows) and number of input features (columns). For example, let us consider an ELM classifier with three input features and three hidden neurons. The input weights and bias values are coded in the string (S) as

$$S(W,B)=\begin{bmatrix} W^s_{11} & W^s_{12} & W^s_{13} & b^s_1 \\ W^s_{21} & W^s_{22} & W^s_{23} & b^s_2 \\ W^s_{31} & W^s_{32} & W^s_{33} & b^s_3 \end{bmatrix} \tag{8}$$

where W^s_{12} represents the input weight between first hidden neuron to second input neuron and b^s_1 represents the bias value for first hidden neuron.

In this string representation, the number of rows are different for different strings in the population. The genetic operators are defined such that it can handle the strings with different sizes. Using the aforementioned string representation we can uniquely represent all combinations of hidden neurons.

3.2. Selection function

In a genetic algorithm, new search nodes for the next generations are selected from the existing set of search nodes (population). This selection process plays an important and critical role. Using genetic operators, a probabilistic selection is performed based upon the fitness of existing search nodes, such that the better search nodes have a higher chance of being selected. It is possible that a search node in the population can be selected more than once for producing new search nodes. In the literature (Michalewicz, 1994), several schemes such as roulette wheel selection and its extensions, scaling techniques, tournament and ranking methods are presented for the selection process. In our studies, normalized geometric ranking method given in Michalewicz (1994) is used for the selection process. In normalized geometric ranking method, the search nodes are arranged in descending order of their fitness value. Let q be selection probability for selecting the best search node and r_j be the rank of the jth search node in the partially ordered set. The probability of search node j being selected, using normalized geometric ranking method is

$$s_j = q'(1-q)^{r_j-1} \tag{9}$$

where $q'=q/(1-(1-q)^{N_p})$ and N_p is the population size.

3.3. Genetic operators

Genetic operators provide the basic search mechanism of the genetic algorithm. The operators are used to create search nodes based on existing search nodes in the population. New search nodes are obtained by combining or rearranging parts of the old search nodes and a new search node obtained may give a better solution to the optimization problem. These genetic operators are analogous to those which occur in the natural world: reproduction (crossover, or recombination) and mutation. The probability of these operators affect the efficacy of the genetic algorithm. The real-coded genetic operators used in our study are described below.

3.3.1. Crossover operator

Crossover operator is a primary operator in genetic algorithm. The role of a crossover operator is to recombine information from two selected search nodes to produce two new search nodes. The crossover operator improves the diversity of the solution. In this paper, we present weight connection and network architecture based crossover operators. The operators which act on individual weight connections of the search nodes are called weight based crossover operators and those which act on network architecture are called network based crossover operators. Let R and S be the two search nodes selected for crossover operations.

$$R(W,b)=\begin{bmatrix} \mathbf{W^r_{11}} & W^r_{12} & W^r_{13} & b^r_1 \\ W^r_{21} & \mathbf{W^r_{22}} & W^r_{23} & b^r_2 \\ W^r_{31} & W^r_{32} & \mathbf{W^r_{33}} & \mathbf{b^r_3} \\ W^r_{41} & W^r_{42} & W^r_{43} & b^r_4 \end{bmatrix} \tag{10}$$

$$S(W,b)=\begin{bmatrix} \mathbf{W^s_{11}} & W^s_{12} & W^s_{13} & b^s_1 \\ W^s_{21} & \mathbf{W^s_{22}} & W^s_{23} & b^s_2 \\ W^s_{31} & W^s_{32} & \mathbf{W^s_{33}} & \mathbf{b^s_3} \end{bmatrix} \tag{11}$$

Now, we present the detailed description of these operators.

- **Weight based operator**: In case of weight connection based crossover, L weight values are randomly selected from the parent set such that $L \in [\min(row(R),row(S))]$. This operator uses an averaging operation to generate the values of the selected connections in the children. Let $P_1=[\mathbf{W^r_{11}}, \mathbf{W^r_{22}}, \mathbf{W^r_{33}}\ \mathbf{b^r_3}]$ be the weights selected from the parent R and $P_2=[\mathbf{W^r_{11}}, \mathbf{W^s_{22}}, \mathbf{W^s_{33}}\ \mathbf{b^s_3}]$ be the weights selected from the parent S. The new values of

weights in the children are generated as

$$H_1 = P_1 + \beta(P_1 - P_2) \tag{12}$$

$$H_2 = P_2 + \beta(P_2 - P_1) \tag{13}$$

where β is a scalar value in the range of $(0 \le \beta \le 1)$. In our simulation studies, β is set to 0.2. The new children generated after the crossover operation are

$$R'(W,b) = \begin{bmatrix} \mathbf{H_1(1)} & W_{12}^r & W_{13}^r & b_1^r \\ W_{21}^r & \mathbf{H_1(2)} & W_{23}^r & b_2^r \\ W_{31}^r & W_{32}^r & \mathbf{H_1(3)} & \mathbf{H_1(4)} \\ W_{41}^r & W_{42}^r & W_{43}^r & b_4^r \end{bmatrix} \tag{14}$$

$$S'(W,b) = \begin{bmatrix} \mathbf{H_2(1)} & W_{12}^s & W_{13}^s & b_1^s \\ W_{21}^s & \mathbf{H_2(2)} & W_{23}^s & b_2^s \\ W_{31}^s & W_{32}^s & \mathbf{H_2(3)} & \mathbf{H_2(4)} \end{bmatrix} \tag{15}$$

- **Network based operator**: In case of the network based operator, we randomly select the weights of L hidden neurons from the parent set. This operator uses heuristic operation to generate the weights of the L th hidden neuron ($L \le \min(row(R), row(S))$). Let hidden neurons selected for crossover operation be 1 (row 1 of R and S). The network weights selected for crossover operation are shown in boldface.

$$R(W,b) = \begin{bmatrix} \mathbf{W_{11}^r} & \mathbf{W_{12}^r} & \mathbf{W_{13}^r} & \mathbf{b_1^r} \\ W_{21}^r & W_{22}^r & W_{23}^r & b_2^r \\ W_{31}^r & W_{32}^r & W_{33}^r & b_3^r \\ W_{41}^r & W_{42}^r & W_{43}^r & b_4^r \end{bmatrix} \tag{16}$$

$$S(W,b) = \begin{bmatrix} \mathbf{W_{11}^s} & \mathbf{W_{12}^s} & \mathbf{W_{13}^s} & \mathbf{b_1^s} \\ W_{21}^s & W_{22}^s & W_{23}^s & b_2^s \\ W_{31}^s & W_{32}^s & W_{33}^s & b_3^s \end{bmatrix} \tag{17}$$

The weights connected to the neuron 1 of parent R be P_1 (highlighted weights in R) and parent S be P_2 (highlighted weights in S). The corresponding weights values of the first hidden neuron in the children are generated as

$$H_1 = P_1 \pm \gamma w_m \frac{P_1 - P_2}{\|P_1 - P_2\|} \tag{18}$$

$$H_2 = P_2 \pm \gamma w_m \frac{P_2 - P_1}{\|P_2 - P_1\|} \tag{19}$$

where w_m is the range of the weight vectors and γ is the positive constant. In our experiment, *range* and γ are set to 2 and 0.2, respectively. The children produced after the crossover operation will be

$$R(W,b) = \begin{bmatrix} \mathbf{H_1(1)} & \mathbf{H_1(2)} & \mathbf{H_1(3)} & \mathbf{H_1(4)} \\ W_{21}^r & W_{22}^r & W_{23}^r & b_2^r \\ W_{31}^r & W_{32}^r & W_{33}^r & b_3^r \\ W_{41}^r & W_{42}^r & W_{43}^r & b_4^r \end{bmatrix} \tag{20}$$

$$S(W,b) = \begin{bmatrix} \mathbf{H_2(1)} & \mathbf{H_2(2)} & \mathbf{H_2(3)} & \mathbf{H_2(4)} \\ W_{21}^s & W_{22}^s & W_{23}^s & b_2^s \\ W_{31}^s & W_{32}^s & W_{33}^s & b_3^s \end{bmatrix} \tag{21}$$

3.3.2. Mutation operator

The mutation operator alters one solution to produce a new solution. The mutation operator is needed to ensure diversity in the population, and to overcome premature convergence and local minima problems. Similar to the crossover operator, we have weight and network based mutation operators, as described below. Let us assume that R is the parent selected for the mutation operation.

- **Weight based operator**: This operator randomly selects M weight values for mutation operations. Let W_{12}^r be the weight selected for the mutation operation. If this operator is applied in generation t, where G is the maximum number of generations, then the new weight value after mutation is

$$W_{12}^r = W_{12}^r + \Delta(t, w_{min} - W_{12}^r), \quad \text{if } \gamma = 0 \tag{22}$$

$$W_{12}^r = W_{12}^r + \Delta(t, w_{max} - W_{12}^r), \quad \text{if } \gamma = 1 \tag{23}$$

where w_{min}, w_{max} are minimum and maximum values of weights (in our studies, it is selected as range of input), $\gamma \in [0,1]$ and

$$\Delta(t,y) = y\left(1 - r^{1-(t^b/G)}\right) \tag{24}$$

where r is the random number between the interval [0,1] and b is the parameter that determines the degree of dependency. This function gives a value in the range $[0,y]$ such that the probability of returning a number close to zero increases as the algorithm advances (Michalewicz, 1994).

- **Network based operator**: The network based mutation operator adds or deletes a hidden neuron in the selected parent. In this operator, addition or deletion of hidden neurons is selected randomly.

  In order to add a hidden neuron, random numbers from real space are assigned to the weights which connect the newly added hidden neuron and input neurons. The number of rows in the child after network based adding operator increase the dimension by one. Let R be the parent selected for network based mutation operation. The child R' produced after adding a new hidden neuron (5th neuron) is

$$R'(W,b) = \begin{bmatrix} W_{11}^r & W_{12}^r & W_{13}^r & b_1^r \\ W_{21}^r & W_{22}^r & W_{23}^r & b_2^r \\ W_{31}^r & W_{32}^r & W_{33}^r & b_3^r \\ W_{41}^r & W_{42}^r & W_{43}^r & b_4^r \\ W_{51}^r & W_{52}^r & W_{53}^r & b_5^r \end{bmatrix} \tag{25}$$

  In the case of deleting a hidden neuron, the operator randomly selects one hidden neuron and delete the row from the parent. After the deletion operation, the number of rows in the child decreases by one. Let R be the parent selected for hidden neuron deletion operation. The operator then selects the hidden neuron 2 (2nd row of weights) for deletion operation. Here, the child R' is created by removing the 2nd row of weights from the parent R.

$$R'(W,b) = \begin{bmatrix} W_{11}^r & W_{12}^r & W_{13}^r & b_1^r \\ W_{31}^r & W_{32}^r & W_{33}^r & b_3^r \\ W_{41}^r & W_{42}^r & W_{43}^r & b_4^r \end{bmatrix} \tag{26}$$

3.4. Fitness function

The objective of the ELM classifier is to develop a better model such that the resulting generalization efficiency is very high. In this approach, we use a fivefold cross-validation approach on 144 training samples to estimate the validation efficiency. Hence, for a given string, the fitness value is equal to the estimated cross-validation efficiency, i.e., first, we find the output weights using analytical equation (2) with four equal data sets and evaluate the generalization efficiency of the classifier using the leave-out set. GA does not use the 46 testing samples in the best input weight selection process.

3.5. Termination criterion

In genetic algorithm, the evolution process continues until a termination criterion is satisfied. The most widely used termination criterion is the maximum number of generations and we use this in our simulation studies.

3.6. RCGA-ELM algorithm

The steps involved in this approach are described below:

1. Randomly create initial population of search nodes.
2. Calculate the fitness of each search node.
3. Select the parents for genetic operations.
4. Generate new population of search nodes using genetic operators.
5. If termination criterion is satisfied then stop otherwise go to step 2.

We call the real-coded genetic algorithm based ELM algorithm as 'real-coded genetic algorithm extreme learning machine (RCGA-ELM)'. We have successfully implemented and tested the real coded hybrid genetic algorithm approach for ELM classifier in MATLAB on a Pentium-IV machine. The convergence of the genetic algorithm depends on population size (N_p), selection probability (q), crossover probability (p_c) and mutation probability (p_m). The following are the GA parameters used in our simulation: P_m is 0.05; P_c is 0.6; maximum number of generations (M_g) is 500; q is 0.08; and population size is 30.

4. A sparse-ELM algorithm

The real-coded genetic algorithm searches for the number of hidden neurons and its weights. This involves a significant amount of computational time to build a compact network with the desired higher generalization efficiency. Hence, we propose another approach using a K-fold cross-validation to select the ELM parameters quickly. The number of hidden neurons, the input weights W and B are selected using K-fold cross-validation.

Cross-validation (CV) and bootstrapping are commonly used methods for estimating the generalization error based on 'resampling' (Breiman, 1996). The resulting estimates of generalization error are often used as criteria for choosing among various models, such as different network architectures and parameters. In this paper, CV approach is used to select the input weights and bias such that the estimated generalization error is small. In K-fold cross-validation, the training data set is divided into K equal subsets (of approximately equal size). One can also use 5×2 cross-validation technique (Alpaydin, 2004) instead of K-fold cross-validation approach. The 5×2 cross-validation technique reduces the type-I error present in the K-fold cross-validation technique (Alpaydin, 2004). The ELM algorithm is called K times, each time leaving one of the subsets from training, and the generalization efficiency is calculated based on the omitted subset. Here, the value of K is 5. In order to avoid the bias/variability in estimating the generalization error, two random five-way splits of data are generated using the training samples. After selecting the input weights and bias value, the classifier model is developed using the complete training set. This ELM algorithm with 5×2-fold validation technique produces better training and generalization efficiency for sparse data classification problems than the conventional ELM algorithm. This algorithm requires lesser computational time compared to the RCGA-ELM approach. We call this algorithm 'sparse extreme learning machine' (S-ELM). The sparse-ELM is the same as the ELM algorithm with k-fold cross-validation for selecting the optimal input weights and bias. The following steps are used to select the H, W and B values:

1. Randomly select a set of hidden neurons $[H_1, H_2,\ldots,H_p]$.
2. For a given H_i, use 5×2-fold cross-validation to select W, B.
3. Develop a classifier model using the best W and B and calculate the training and cross-validation efficiencies.
4. Repeat the steps 2 and 3 for different values of H_i; $i=1,2,\ldots,p$ and select the H_i for which the training and cross-validation efficiencies are high.
5. For the best H, W and B calculate the testing efficiency.

5. Micro-array gene expression based cancer classification problem

Cancer detection and classification using standard clinical data are a difficult and complicated process. Identifying the anatomic site of origin of the cancer is an important component in cancer treatment. Hence, in recent years bio-medical research activities are focused on identification of site of origin, type and molecular classification of cancer. In Ramaswamy et al. (2002), global cancer mapping (GCM) data have been used to classify human cancer types using support vector machines. GCM data are a collection of micro-array gene expression data for cancer and normal tissue specimens. The data were collected from six medical institutions consisting of samples from 14 different types of cancers. Each tissue specimen consists of 16 063 genes and the database has 198 primary samples from 14 types of cancer. Since, eight of samples belong to metadata, we did not include them in our study. This classification problem is a typical example of sparse data where the dimension of the feature space is larger than the number of samples available. In Ramaswamy et al. (2002), recursive feature elimination method is used to identify the most significant genes. Using these genes as inputs to the neural network, the classifier models were developed for different combinations of the most significant genes. The simulation results clearly show that the performance increases with increase in number of significant genes as input to the network. The performance saturates when the number of most significant genes reaches 100. Hence, we use a maximum of 98 significant genes for our classifier development.

Out of 190 available tumor samples, 144 samples were randomly selected for training and 46 samples were used for testing. From the training data one can easily observe that the number of samples per class vary from 8 to 30. The imbalance in the training data and fewer training samples affects the performances of the classifier model considerably. We use recursive feature elimination approach as explained in Ramaswamy et al. (2002) to select 98 of the most significant genes from the complete set of 16 063 genes. In Ramaswamy et al. (2002), different combinations of 144 training samples and 46 testing samples were generated to study the sample independence of the classifier algorithm. We also generate similar training and testing sets for our study. For RCGA-ELM and S-ELM, the best input weights were found using one of the 144 training sample sets and these values were used to develop a classifier for different random trials.

6. Performance evaluation of RCGA-ELM and sparse-ELM

In this section, we compare the improved and evolutionary ELM algorithm with the standard ELM using the GCM data set

which has 98 input features and 14 distinct classes. In our studies, we use sigmoidal and radial basis function as activation functions for the hidden neuron layer in ELM. The mean and standard deviation of training and testing efficiency based on 100 random trials are given in Table 1. The performance of sparse-ELM and RCGA-ELM is compared with the ELM and support vector machine (Ramaswamy et al., 2002) algorithms.

Based on the GCM data set, the testing/generalization efficiency for SVM-OVO (one-versus-one) method is 73.78% as reported in Ramaswamy et al. (2002), which is approximately 6% less than the performance of ELM-RBF (network with radial basis function as the activation function in the hidden layer) algorithm. From Table 1, we can see that the optimal selection of input weights and bias values improves the performance of the ELM algorithm considerably, i.e., the testing efficiency of proposed sparse-ELM (RBF) and RCGA-ELM (RBF) is approximately 10% higher than the ELM (RBF) and SVM-OVO.

In addition to improve efficiency we can see that for the RBF network, the standard deviation is 30% less for RCGA-ELM and 33% less for sparse-ELM compared to the ELM algorithm. For the SIG network the variation is lesser by 8–18% for RCGA-ELM and sparse-ELM. Hence there is a less variation and more stability in the testing efficiency results for RCGA-ELM and sparse-ELM. Also, the RCGA-ELM algorithm finds a minimal network with better testing efficiency. Similar behavior is also observed from the ELM-SIG (sigmoidal activation function) network.

The simulation results clearly show that one has to find optimal input weight and bias values to obtain better generalization performance for ELM. The RCGA-ELM finds the smallest network to approximate the classifier function compared to other approaches. But, the time taken to obtain the number of hidden neurons, input weights and bias values using RCGA-ELM is approximately eight times more compared to sparse-ELM. The CPU time given here for the ELM algorithm is based on MATLAB implementation and those for the SVM algorithm is based on C++ implementation. In general, the C++ implementation usually runs 10–50 times faster than the MATLAB implementation. In spite of this, the training time for sparse-ELM algorithm is approximately the same as SVM-OVO, whereas the generalization efficiency of sparse-ELM is better than for SVM-OVO.

6.1. Statistical significance test

In this paper, we also present a statistical significance test called 'binomial test' between RCGA-ELM and SVM-OVO classifiers as explained in Salzberg (1997). Let n be the number of test samples for which RCGA-ELM and SVM produce different results. Let s (success) be the number of times RCGA-ELM predicts the class label correctly than SVM-OVO and f (failure) be the number of times SVM-OVO predicts the class label correctly compared to RCGA-ELM. Now, we find the p-value (probability of s success in n trials) using binomial distribution as

$$\sum_{j=s}^{n} \frac{n}{j(n-j)} p^j q^{n-j} \tag{27}$$

where p and q are the probability of success for RCGA-ELM and SVM-OVO algorithms. If we expect no difference between the two algorithms then $p=q=0.5$. In our experiment, class label for eight ($n=8$) samples are different between the two algorithms, and for $s=6$ samples RCGA-ELM classifier predicts the class label correctly when compared to SVM-OVO classifier. Then the p-value is 0.1094. From the binomial distribution for probability of success for six out of eight trails, there is only 11% probability that RCGA-ELM can outperform SVM-OVO, as it were to happen by random chance. Hence, we can say that the RCGA-ELM classifier is better than the SVM-OVO classifier with high confidence. Similar observations were made between sparse-ELM classifier and SVM classifier. Hence, one can say that the sparse-ELM and RCGA-ELM classifiers are more suitable for this problem than the SVM-OVO classifier.

7. Conclusions

Random selection of the number of hidden neurons, input weights and bias values affects the generalization performance of ELM classifiers considerably. In order to optimally select these parameters for multi-category classification problems with sparse data condition, we have proposed two approaches, namely, 'RCGA-ELM' and 'sparse-ELM'. The genetic operators are designed such that the RCGA-ELM finds a compact network with higher generalization efficiency. However, the RCGA-ELM algorithm requires a higher computational effort. To overcome this problem, we have presented a sparse-ELM using a K-fold validation scheme to select the number of neurons, input weights and bias values quickly. The performance of the proposed algorithms have been evaluated using the micro-array gene expression based GCM data for human cancer classification. The results clearly show that the proposed algorithms achieve approximately 10% increase in testing the efficiency compared to the standard ELM algorithm and SVM-OVO. One can observe that the sparse-ELM achieves almost similar generalization performance as that of RCGA-ELM

Table 1
Simulation results for cancer classification.

Type	Algorithm	H	Training efficiency		Testing efficiency		Training time in min[a]
			Mean	STD.	Mean	STD.	
SIG	ELM	45	95.12	1.32	76.50	5.31	8.20[b,d]
	S-ELM	40	94.44	1.62	88.13	4.88	30.41
	RCGA-ELM	38	94.91	1.42	89.97	4.35	270.34
RBF	ELM	50	92.30	2.25	79.43	6.23	10.20
	S-ELM	40	95.73	1.41	88.19	4.38	30.41
	RCGA-ELM	36	95.43	1.23	89.25	4.13	240.21
OVO	SVM	106[c]	96.50	1.85	73.78	5.10	29.45[b,d]

[a] MATLAB implementation.
[b] Training time includes the selection of number of hidden neurons.
[c] Number of support vectors.
[d] C++ implementation.

with significantly lesser computational time. Thus, one can use sparse-ELM approach for efficient selection of optimal parameters for function approximation and classification to achieve higher generalization performance.

Acknowledgements

The second author acknowledges gratefully the support extended to her at Bio-Informatics Research Center (BIRC), School of Computer Engineering, Nanyang Technological University, Singapore, where this study was undertaken while she was a visiting student at BIRC. The authors are also thankful to Dr. G.-B. Huang, School of Electrical and Electronics Engineering for his valuable suggestions. The authors also thank the reviewer for the valuable comments and suggestions, which improved the quality of the paper.

References

Alpaydin, E., 2004. Assessing and comparing classification algorithms. In: Introduction to Machine Learning. MIT Press, pp. 327–332.

Breiman, L., 1996. Heuristics of instability and stabilization in model selection. Annals of Statistics 24 (6), 2350–2383.

Huang, G.-B., 2003. Learning capability and storage capacity of two-hidden layer feedforward networks. IEEE Transactions on Neural Networks 14 (2), 274–281.

Huang, G.-B., Zhu, Q.Y., Siew, C.K., 2004. Extreme learning machine: a new learning scheme of feedforward neural networks. Proceedings of International Joint Conference on Neural Networks (IJCNN2004), vol. 2; 2004, pp. 985–990.

Huang, G.-B., Zhu, Q.Y., Siew, C.K., 2006a. Extreme learning machine: theory and applications. Neurocomputing 70 (1–3), 489–501.

Huang, G.-B., Zhu, Q.Y., Siew, C.K., 2006b. Real-time learning capability of neural networks. IEEE Transactions on Neural Networks 17 (4), 863–878.

Huang, G.-B., Zhu, Q.Y., Siew, C.K., 2006c. Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Transactions on Neural Networks 17 (4), 879–892.

Michalewicz, Z., 1994. Genetic Algorithm+Data Structures=Evolution Programs. Springer-Verlag.

Ramaswamy, S., Tamayo, P., Rifkin, R., Mukherjee, S., Yeang, C., Angelo, M., Ladd, C., Reich, M., Latulippe, E., Mesirov, J., Poggio, T., Gerald, W., Loda, M., Lander, E., 2002. Multi-class cancer diagnosis using tumor gene expression signatures. Proceedings of National Academic of Sciences 98 (26), 15149–15154.

Salzberg, S.L., 1997. On comparing classifiers: pitfalls to avoid and a recommended approach. Data Mining and Knowledge Discovery 1 (3), 317–328.

Schaffer, J.D., Whitley, D., Eshelman, L.J., 1992. Combinations of genetic algorithms and neural networks: a survey of the state of the art. Proceedings of International Workshop on Combinations of Genetic Algorithms and Neural Networks, vol. 1; 1992, pp. 1–37.

Zhu, Q.Y., Qin, A.K., Suganthan, P.N., Huang, G.-B., 2005. Evolutionary extreme learning machine. Pattern Recognition 38 (10), 1759–1763.