Applied Soft Computing 44 (2016) 101–107

Contents lists available at ScienceDirect

Applied Soft Computing

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

# A new pruning method for extreme learning machines via genetic algorithms

Alisson S.C. Alencar [b], Ajalmar R. Rocha Neto [a,*], João Paulo P. Gomes [b]

[a] Federal Institute of Ceará, Department of Teleinformatics Fortaleza, Ceará, Brazil
[b] Federal University of Ceará, Department of Computer Science Fortaleza, Ceará, Brazil

ARTICLE INFO

*Article history:*
Received 27 July 2015
Received in revised form 16 March 2016
Accepted 17 March 2016
Available online 6 April 2016

*Keywords:*
Extreme learning machines
Pruning methods
Genetic algorithms

ABSTRACT

Extreme learning machine (ELM) is a recently proposed learning algorithm for single hidden layer feed-foward neural networks (SLFN) that achieved remarkable performances in various applications. In ELM, the hidden neurons are randomly assigned and the output layer weights are learned in a single step using the Moore-Penrose generalized inverse. This approach results in a fast learning neural network algorithm with a single hyperparameter (the number of hidden neurons). Despite the aforementioned advantages, using ELM can result in models with a large number of hidden neurons and this can lead to poor generalization. To overcome this drawback, we propose a novel method to prune hidden layer neurons based on genetic algorithms (GA). The proposed approach, referred as GAP-ELM, selects subset of the hidden neurons to optimize a multiobjective fitness function that defines a compromise between accuracy and the number of pruned neurons. The performance of GAP-ELM is assessed on several real world datasets and compared to other SLFN and a well known pruning method called Optimally Pruned ELM (OP-ELM). On the basis of our experiments, we can state that GAP-ELM is a valid alternative for classification tasks.


## 1. Introduction

Single-hidden layer feedforward neural networks (SLFNs) have been successfully applied in various classification and regression problems like fault detection [1] and face recognition [2]. Among the training methods for SLFN, one can cite methods based on first-order optimization algorithms (gradient descent) as the error backpropagation used to train multilayer perceptrons (MLPs) [3]. Despite its popularity, gradient descent methods are known to have a very slow convergence.

In [4], Huang et al. proposed a new procedure to train SLFNs and the resulting model was termed as the extreme learning machine (ELM). ELM relies on fixed-weight hidden neurons (nodes) with non-linear activation functions [4]. The weights of the hidden neurons are assigned randomly. After that, a batch learning procedure is used to adjust the output layer weights. The ELM has two main advantages: the first one is the training speed when compared to most learning algorithms such as MLPs and support vector machines (SVMs) [5]. A second advantage is the use of a single hyperparameter (number of hidden neurons).

It is well known that neural networks with a large number of hidden neurons may have problems with the performance on unseen cases due to training data overfitting. Nevertheless, a neural network with a small hidden-layer suffers from training data underfitting. Taking these statements into account, defining the number of a hidden-layer neurons is an important issue for extreme learning machines. A possible approach consists on pruning unnecessary neurons during the training process [6].

Some previous works have tackled the reduction issue of the hidden-neurons such as the optimally pruned ELM (OP-ELM). To do so, OP-ELM ranks the best neurons using multiresponse sparse regression (MRSR) and then selects the optimal number of neurons by leave-one-out (LOO) [7]. An improvement of this methodology named Tikhonov Regularized OP-ELM (TROP-ELM) was made by introducing a $L_1$ regularization (penalty) to rank the neurons and a $L_2$ penalty to prune the network [8]. Other methodologies have dealt with finding the suitable number of neurons based on adding (removing) neurons to (from) the hidden-layer [9–11]. Recently, Bayesian methods are exploited how to learn the output weights of ELM and a sparse version called Sparse Bayesian ELM (SBELM) was presented in [12]. Even though OP-ELM and TROP-ELM outperform the standard ELM, a large number of hidden neurons remain in the trained model due to the minimization of training error in ranking neurons, resulting therefore in a highly computational cost [12].

* Corresponding author.
*E-mail addresses:* alencar.alisson@lia.ufc.br (A.S.C. Alencar), ajalmar@ifce.edu.br (A.R. Rocha Neto), jpaulo@lia.ufc.br (J.P.P. Gomes).

http://dx.doi.org/10.1016/j.asoc.2016.03.019
1568-4946/

Genetic algorithms (GAs) have been used to solve optimization problems in many different real-world areas, such as medicine, bio-informatics, economics, chemistry and so on [13,14]. GAs are optimization tools which can be used to generate useful solutions to search problems. Due to the underlying features of GAs, some optimization problems can be solved without the assumption of linearity, differentiability, continuity or convexity of the objective function. Unfortunately, these desired features are not found in many classical mathematical methods when applied to the same kind of problems.

In the literature, we can find several works combining GAs and ELMs. In [15], an weighted ELM for imbalanced classes is proposed. The GA is used to assign higher weights to examples belonging to the minority class and lower weights to the majority class [15]. In [16] GAs were used to optimize the weights of connections between the input layer and the hidden-layer, the hidden-layer neuron biases, the set of input variables, as well as the number of neurons and activation function of each neuron. Even though the method presented promising results, the fundamentals behind ELM had been somewhat lost, since the hidden-layer weights and biases are updated. Following the same idea of weight adjustment, Lahoz et al. applies a bi-objective genetic algorithm to minimize both the mean square error and the number of hidden neurons [17]. We must emphasize at this point that it would be interesting to work with only those neurons generated at random (with no weight adjustment) such as the aforementioned proposals OP-ELM and TROP-ELM do, since we suppose each and every required neuron for a high-performance model is already in the set of randomly generated neurons.

To combine the aforementioned advantages of ELMs and GAs, this work uses a GA to prune unnecessary or similar hidden-neurons while keeping the performance when compared with standard classifiers. Our proposal deals with a multi-objective optimization problem in a priori sense (i.e., a multi-objective problem is turned into a single one). The first objective is to achieve high-performance classifiers while the second is to prune as many neurons as possible from the full set of neurons, which were generated in previous phase (in the beginning of the process). To do this, we also propose a new a priori multi-objective fitness function which incorporates a cost of pruning in its formulation.

The remaining part of this paper is organized as follows. In Section 2 we review the fundamentals of the ELMs. In Section 3 we present the optimally pruned ELM we use for comparison. In Section 4, we introduce genetic algorithms which are necessary to understanding of our proposal presented in Section 5 and then, in Section 6, we present our simulations. Finally, the paper is concluded in Section 7.

## 2. Extreme learning machines

ELM is a single-hidden layer feedforward network proposed in [4]. For a network with $p$ input units, $q$ hidden neurons and $c$ outputs, the $i$-th output at time step $t$, is given by

$$o_i(t) = \mathbf{m}_i^T \mathbf{h}(t), \tag{1}$$

where $\mathbf{m}_i \in \mathbb{R}^q, \forall i \in \{1, \ldots, c\}$, is the weight vector connecting the hidden neurons to the $i$-th output neuron, and $\mathbf{h}(t) \in \mathbb{R}^q$ is the vector of hidden neurons' outputs for a given input pattern $\mathbf{x}(t) \in \mathbb{R}^p$ from a data set $\{\mathbf{x}(t)\}_{t=1}^n$. The vector $\mathbf{h}(t)$ itself is defined as

$$\mathbf{h}(t) = [f(\mathbf{w}_1^T \mathbf{x}(t) + b_1), f(\mathbf{w}_2^T \mathbf{x}(t) + b_2), \ldots, f(\mathbf{w}_q^T \mathbf{x}(t) + b_q)]^T, \tag{2}$$

where $b_l$ is the bias of the $l$-th hidden neuron, $\mathbf{w}_l \in \mathbb{R}^p$ is the weight vector of the $l$-th hidden neuron and $f(\cdot)$ is a sigmoidal activation function. The weight vectors $\mathbf{w}_l$ are randomly sampled from a normal or even from a uniform distribution.

Let $\mathbf{H} = [\mathbf{h}(1)\ \mathbf{h}(2) \cdots \mathbf{h}(n)]$ be a $q \times n$ matrix whose $t$-th column is the hidden-layer output vector $\mathbf{h}(t) \in \mathbb{R}^q$. Similarly, let $\mathbf{D} = [\mathbf{d}(1)\ \mathbf{d}(2) \cdots \mathbf{d}(n)]$ be a $c \times n$ matrix whose the $t$-th column is the desired (target) vector $\mathbf{d}(t) \in \mathbb{R}^c$ associated with the input pattern $\mathbf{x}(t)$, $t = 1, \ldots, N$. Finally, let $\mathbf{M} = [\mathbf{m}_1\ \mathbf{m}_2 \cdots \mathbf{m}_c]$ be a $q \times c$ matrix, whose $i$-th column is the weight vector $\mathbf{m}_i \in \mathbb{R}^q$, $i = 1, \ldots, c$. Thus, these three matrices are related by the linear mapping

$$\mathbf{D} = \mathbf{M}^T \mathbf{H}, \tag{3}$$

where the matrices $\mathbf{D}$ and $\mathbf{H}$ are known and built from the data, while the weight matrix $\mathbf{M}$ is unknown. Nevertheless, the weight matrix $\mathbf{M}$ can be easily computed by means of the Moore–Penrose pseudo-inverse method as follows

$$\mathbf{M} = (\mathbf{H}\mathbf{H}^T)^{-1} \mathbf{H}\mathbf{D}^T. \tag{4}$$

Assuming that the number of output neurons is equal to the number of classes, the class index $i^*$ for a new input pattern is given by the following decision rule:

$$i^* = arg \max_{i=1,\ldots,c} \{o_i\}, \tag{5}$$

where $o_i$ is computed as in Eq. (1).

## 3. Optimally pruned extreme learning machine

Optimally pruned extreme learning machine (OP-ELM) comprises three main steps:

- SFLN construction
- Ranking the neurons using MRSR
- Select the number of neurons using a LOO validation

In the first step, a SLFN is constructed with a large number of hidden layer neurons. To improve robustness and generality, the OP-ELM methodology uses a combination of three different sorts of kernels instead of the sigmoid kernel. Possible kernel types are linear, sigmoid and Gaussian.

The second step consist on building a multiresponse sparse regression (MRSR) that maps the outputs of the hidden layer to the targets. MRSR can be seen as an extension of the least angle regression (LARS) algorithm and hence is actually a variable ranking technique, rather than a selection one [7]. More details about the MRSR can be find in [18].

After ranking the neurons, the decision for the best number of neurons in the model is taken by a leave-one-out (LOO) validation method. However, the LOO method can be very time consuming, if the data set is large. To overcome this drawback, the authors use the PREdiction Sum of Squares (PRESS) statistics, which can give a direct and exact formula for the calculation of the LOO [7].

## 4. Genetic algorithms

Genetic algorithm is a meta-heuristic search method inspired by natural evolution, such as inheritance, mutation, natural selection and crossover. This meta-heuristic can be used to generate useful solutions to optimization and search problems. Due to the characteristics of GA, it is easier to solve various problems by GA than other mathematical methods which do have to rely on the assumption of linearity, differentiability, continuity or convexity of the objective function.

In a genetic algorithm, a population of candidate individuals (or solutions) to an optimization problem is evolved toward better solutions by natural selection, i.e., a fitness function. In this population, each individual has a set of genes (gene vector), named chromosome, which can be changed by mutation or combined generation-by-generation with other one to build new individuals

by reproduction processes which use crossover. The most common way of representing solutions is in a binary format, i.e., strings of 0s and 1s. Despite that, other kind of encodings are also possible.

The first population – set of solutions – is generated in a pseudo-randomized mode and it evolves in cycles, usually named generations. A value of the fitness function (its accuracy) for each individual ranks how 'good' the solution is. One can calculate the fitness value after decoding the chromosome and the best solution of the problem to be solved has the best fitness value. This value is used in order to guide the reproduction process where individuals with high fitness value have more chance to spread out their genes over the population.

Standard implementation of genetic algorithm for natural selection is the roulette wheel. After the selection process, the selected individuals are used as input to other genetic operators: crossover and mutation. Crossover operator combines two strings of chromosomes, but mutation modifies a few bits in a single chromosome.

A single objective GA is presented below.

1. Initiate $t = 0$, where $t$ stands for the generation;
2. Generate initial population $P(t)$, randomly;
3. For each individual $i$ in $P(t)$
4. Evaluate the fitness function;
5. While stopping criteria is not achieved
6. Select some individuals from $P(t)$;
7. Apply crossover operator to selected individuals;
8. Apply mutation operator to selected individuals;
9. Compute $t = t + 1$;
10. Evaluate fitness function for the new population $P(t)$;
11. Select the best individual or solution.

## 5. Proposal: Genetic algorithms for pruned ELM (GAP-ELM)

Our proposal, called GAP-ELM, is based on a priori multi-objective genetic algorithm which aims at improving (maximizing) the classifier accuracy over the training dataset while reducing (minimizing) the number of hidden neurons. The individual with only one chromosome is represented by a binary vector of genes where each one is set to either "one" (true) whether a certain hidden neuron will be used in the hidden-layer or "zero" (false) otherwise.

The scalarization-based fitness function [19] is a balance between the value of the resulting classifier accuracy when we take into account the genes that were set to "one" and the proportion of neurons pruned from the hidden-layer. In our approach each individual has as many hidden neurons as the number of genes set up to "one". The idea is to eliminate those hidden-layer neurons that are unnecessary or very similar to others. For this reason, the population can evolve to the best solution or at least to a better one.

### 5.1. Individuals or chromosomes

As stated, an ELM has $q$ neurons in the hidden-layer, thus an individual also has $q$ genes in the chromosome, since one individual is made of only one chromosome. Generally speaking, we model an individual with genetic material as $\mathbf{g} = [g_1\, g_2 \ldots g_i \ldots g_{q-1}\, g_q]$, where $g_i \in \{0, 1\}$ stands for the $i$-th hidden-layer neuron from all the candidate neurons.[1] Therefore, in order to eliminate a certain neuron from the hidden-layer we must set $g_i$ to zero. Hence, we have as many pruned neurons as the number of "zeros" in the vector $\mathbf{g}$.

To illustrate the proposed approach we present simple examples of how to model individuals for a GAP-ELM from five neurons

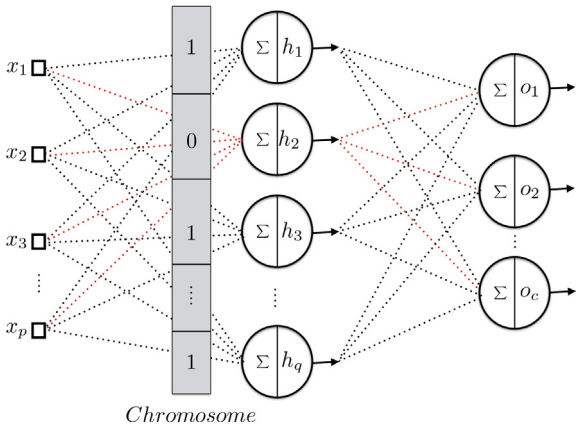


**Fig. 1.** Extreme learning machine for a chromosome $\mathbf{g} = [1\,0\,1 \ldots 1]$. In this figure, red dotted lines show connections we do not need any more. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

generated at random. The first example is about having all of the five neurons in the hidden-layer. To do so, we need to set up a gene vector $\mathbf{g}$ to [1 1 1 1 1]. The second example, we eliminate the second hidden-neuron by an individual with genetic material as $\mathbf{g} = [1\,0\,1\,1\,1]$. If the genetic material is set up as $\mathbf{g} = [1\,0\,0\,0\,1]$, we have just two hidden neurons. The idea behind this individual modeling is presented in the Fig. 1

### 5.2. Fitness function for GAP-ELM

The proposed fitness function relies on maximizing both the accuracy and the pruning rate. In this context, we look for high accuracy and few hidden neurons. Alternatively, we can also state that our fitness function is based on minimizing both the error rate and the ratio of the number of hidden neurons after the pruning process to the total number of candidate neurons, henceforth called neuron rate.

To have a simple fitness function for minimization in a priori sense [19], we weighted the relation between the error and neuron rate by a factor $\alpha \in [0, 1]$ as follows

$$fitness(\mathbf{g}) = \alpha * error\_rate(\mathbf{g}) + (1 - \alpha) * neuron\_rate(\mathbf{g}) \tag{6}$$

where $error\_rate(\mathbf{g})$ is the ratio of the number of misclassification patterns to the number of training patterns, $\alpha \in [0, 1]$ is a factor of importance[2] one gives to the error rate reduction; as well as

$$neuron\_rate(\mathbf{g}) = \frac{\sum_{i=1}^{q} g_i}{q}, \tag{7}$$

where $\sum_{i=1}^{q} g_i$ is upper bounded by $q$. We present the algorithm for GAP-ELM in Section 5.3.

Considering the proposed fitness function, the error rate can be obtained using any cross validation procedure on the training set. However, using standard procedures like 10-fold cross validation and LOO can lead to a high computational cost. To overcome this issue, we computed the error rate using the LOO method through the Prediction Sum of Squares (PRESS) statistic.

[1] Candidate neurons are those ones whose we generate randomly.

[2] We can see alpha as the cost of pruning. The lower the alpha value, the lower the number of hidden neurons.

**Table 1**
List of datasets evaluated in this work.

| Data set | Abbreviation | # Patterns |
|---|---|---|
| Adult | ADU | 48808 |
| Banana | BAN | 1001 |
| Breast Cancer Wisconsin | BCW | 683 |
| Haberman | HAB | 306 |
| HIV | HIV | 3272 |
| Pima Indians Diabetes | PID | 768 |
| Ripley | RIP | 1250 |

*5.2.1. Prediction sum of squares (PRESS)*

The PRESS statistic provides a direct and exact way of computing the LOO error for linear models. The PRESS statistic $e_i^{loo}$ for the model when the $i$-th pattern is left out can be obtained by

$$e_i^{loo} = \frac{d_i - \mathbf{h}_i^T \mathbf{m}^{-i}}{1 - z_{i,i}}, \tag{8}$$

where $\mathbf{m}^{-i}$ is the solution for the linear system when, as mentioned, the $i$-th pattern is left out of the training set and $z_{i,i}$ is the $i$-th main diagonal term for the hat matrix $\mathbf{Z}$, so that $\mathbf{Z} = \mathbf{H}\mathbf{H}^{\dagger} = \mathbf{H}(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$.

**Table 2**
Results for ELM, OP-ELM and GAP-ELM. (#TC is expressed in seconds).

| Data set | | ELM | OP-ELM | GAP-ELM | | | |
|---|---|---|---|---|---|---|---|
| | | | | $\alpha = 1.00$ | $\alpha = 0.99$ | $\alpha = 0.98$ | $\alpha = 0.97$ |
| HAB | ACC | 73.3 ± 4.9 | 73.8 ± 4.7 | 75.0 ± 4.2 | 75.3 ± 4.4 | 75.1 ± 4.9 | 74.8 ± 4.9 |
| | #HN | 12.1 ± 8.8 | 5.5 ± 1.5 | 5.3 ± 1.6 | 2.7 ± 0.9 | 2.3 ± 0.6 | 2.1 ± 0.3 |
| | #TC | 2.3 ± 0.3 | 8e − 3 ± 0.0 | 0.9 ± 0.1 | 0.7 ± 0.1 | 0.7 ± 0.0 | 0.7 ± 0.0 |
| BCW | ACC | 96.3 ± 1.7 | 96.5 ± 1.5 | 97.1 ± 1.6 | 97.2 ± 1.7 | 97.3 ± 1.6 | 97.1 ± 1.5 |
| | #HN | 61.2 ± 22.5 | 31.2 ± 1.5 | 21.7 ± 9.1 | 11.1 ± 5.2 | 8.3 ± 4.0 | 6.8 ± 3.2 |
| | #TC | 8.3 ± 0.8 | 6e − 2 ± 0.0 | 6.3 ± 2.1 | 3.7 ± 1.0 | 3.2 ± 0.7 | 2.8 ± 0.6 |
| PID | ACC | 74.5 ± 3.8 | 74.5 ± 3.4 | 76.0 ± 3.0 | 75.6 ± 2.6 | 75.3 ± 3.1 | 75.5 ± 2.8 |
| | #HN | 47.2 ± 23.6 | 17.2 ± 3.4 | 23.5 ± 4.8 | 15.1 ± 4.0 | 10.9 ± 2.8 | 8.8 ± 2.0 |
| | #TC | 9.5 ± 0.1 | 4e − 2 ± 0.0 | 5.6 ± 1.1 | 3.9 ± 0.7 | 3.4 ± 0.5 | 3.1 ± 0.4 |
| BAN | ACC | 99.8 ± 0.3 | 99.8 ± 0.3 | 99.7 ± 0.4 | 99.6 ± 0.4 | 99.6 ± 0.4 | 99.6 ± 0.4 |
| | #HN | 51.4 ± 19.5 | 43.3 ± 4.9 | 28.3 ± 17.0 | 11.9 ± 4.5 | 11.5 ± 4.7 | 10.0 ± 3.1 |
| | #TC | 10.7 ± 0.4 | 6e − 2 ± 0.0 | 9.2 ± 3.5 | 5.6 ± 1.3 | 5.3 ± 1.3 | 4.9 ± 1.0 |
| HIV | ACC | 86.5 ± 1.0 | 86.8 ± 1.3 | 86.6 ± 1.3 | 86.6 ± 1.3 | 86.6 ± 1.3 | 86.6 ± 1.3 |
| | #HN | 74.9 ± 63.9 | 17.8 ± 8.4 | 18.2 ± 3.2 | 13.8 ± 2.9 | 10.1 ± 2.3 | 9.3 ± 2.1 |
| | #TC | 26.4 ± 1.3 | 0.6 ± 0.0 | 74.1 ± 7.8 | 67.1 ± 5.4 | 63.6 ± 5.9 | 61.8 ± 5.0 |
| RIP | ACC | 90.0 ± 1.7 | 90.9 ± 1.6 | 90.6 ± 1.7 | 90.6 ± 1.7 | 90.3 ± 1.7 | 90.1 ± 1.7 |
| | #HN | 45.8 ± 15.3 | 29.7 ± 5.2 | 26.4 ± 6.1 | 13.7 ± 3.9 | 9.5 ± 2.3 | 8.1 ± 1.8 |
| | #TC | 11.7 ± 0.2 | 6e − 2 ± 0.0 | 11.6 ± 2.0 | 7.3 ± 1.1 | 6.2 ± 0.7 | 5.8 ± 0.6 |
| ADU | ACC | 84.2 ± 0.3 | 83.7 ± 0.3 | 83.2 ± 0.4 | 83.1 ± 0.4 | 82.9 ± 0.4 | 82.8 ± 0.4 |
| | #HN | 466.5 ± 35.9 | 145.7 ± 29.3 | 77.0 ± 7.1 | 55.3 ± 10.2 | 37.4 ± 6.4 | 30.8 ± 5.1 |
| | #TC | 369.8 ± 11.1 | 113.8 ± 3.0 | 497.4 ± 55.5 | 395.8 ± 56.2 | 319.9 ± 34.2 | 298.3 ± 36.4 |

**Table 3**
Results for ELM, OP-ELM, MLP, RBF and GAP-ELM with 40 hidden neurons at the beginning.

| Data set | | ELM | OP-ELM | MLP | RBF | GAP-ELM $\alpha = 1.00$ |
|---|---|---|---|---|---|---|
| HAB | ACC | 74.7 ± 2.5 | 75.3 ± 4.8 | 74.8 ± 0.9 | 69.5 ± 2.0 | 73.4 ± 4.6 |
| | #HN | 40 ± 0.0 | 8.5 ± 3.8 | 40 ± 0.0 | 40 ± 0.0 | 13.4 ± 6.6 |
| | #TC | 4e − 4 ± 0.0 | 2e − 2 ± 0.0 | 9.1 ± 0.6 | 1.8 ± 0.1 | 1.6 ± 0.5 |
| BCW | ACC | 97.0 ± 0.7 | 96.4 ± 1.6 | 96.3 ± 1.7 | 96.0 ± 0.8 | 96.6 ± 1.7 |
| | #HN | 40 ± 0.0 | 12.8 ± 7.8 | 40 ± 0.0 | 40 ± 0.0 | 13.7 ± 4.9 |
| | #TC | 6e − 4 ± 0.0 | 3e − 2 ± 0.0 | 34.6 ± 4.2 | 2.3 ± 0.4 | 3.7 ± 0.8 |
| PID | ACC | 72.4 ± 2.2 | 76.2 ± 3.9 | 76.5 ± 1.8 | 70.8 ± 2.1 | 74.4 ± 2.9 |
| | #HN | 40 ± 0.0 | 14.3 ± 5.2 | 40 ± 0.0 | 40 ± 0.0 | 19.7 ± 4.0 |
| | #TC | 7e − 4 ± 0.0 | 3e − 2 ± 0.0 | 37.3 ± 1.3 | 3.6 ± 0.6 | 4.7 ± 0.9 |
| BAN | ACC | 97.3 ± 2.0 | 99.7 ± 0.4 | 97 ± 0.3 | 99.3 ± 2.3 | 99.7 ± 0.4 |
| | #HN | 40 ± 0.0 | 36.7 ± 3.3 | 40 ± 0.0 | 40 ± 0.0 | 14.1 ± 9.0 |
| | #TC | 7e − 4 ± 0.0 | 4e − 2 ± 0.0 | 25.7 ± 0.5 | 2.9 ± 0.5 | 5.6 ± 1.8 |
| HIV | ACC | 87.1 ± 0.5 | 86.6 ± 1.3 | 86.8 ± 1.1 | 89.3 ± 1.7 | 86.7 ± 1.2 |
| | #HN | 40 ± 0.0 | 11.5 ± 4.2 | 40 ± 0.0 | 40 ± 0.0 | 17.7 ± 3.5 |
| | #TC | 2e − 3 ± 0.0 | 0.3 ± 0.0 | 150.3 ± 5.6 | 27.1 ± 0.9 | 65.0 ± 5.0 |
| RIP | ACC | 87.8 ± 1.1 | 91.1 ± 1.5 | 88.3 ± 2.7 | 88.6 ± 2.8 | 90.0 ± 2.0 |
| | #HN | 40 ± 0.0 | 27.5 ± 4.7 | 40 ± 0.0 | 40 ± 0.0 | 23.1 ± 4.5 |
| | #TC | 8e − 4 ± 0.0 | 5e − 2 ± 0.0 | 33.5 ± 2.2 | 7.0 ± 0.5 | 9.5 ± 1.4 |
| ADU | ACC | 78.7 ± 1.1 | 81.8 ± 0.6 | 84.4 ± 0.4 | 84.2 ± 0.7 | 82.4 ± 0.5 |
| | #HN | 40 ± 0.0 | 31.7 ± 4.0 | 40 ± 0.0 | 40 ± 0.0 | 31.6 ± 2.3 |
| | #TC | 3e − 2 ± 0.0 | 1.2 ± 0.0 | 19.1e3 ± 157 | 696.9 ± 32.5 | 271.1 ± 11.2 |

Thus, the leave-one-out estimate of the local mean integrated squared error $\bar{e}^{loo}$ can be obtained as

$$\bar{e}^{loo} = \frac{1}{n}\sum_{i=1}^{n}(e_i^{loo})^2, \tag{9}$$

where $n$ is the number of training patterns. More information about PRESS can be found in [20,7].

### 5.3. GAP-ELM algorithm

GAP-ELM algorithm for a genetic algorithms-based pruned ELM can be described as follows.

1. Initiate $t = 0$, where $t$ stands for the current generation;
2. Generate $q$ hidden neurons randomly;
3. Generate initial population $P(t)$, i.e., a set of $\{\mathbf{g}\}_{i=1}^{s}$ and its related matrices $\{\mathbf{H}_i\}_{i=1}^{s}$, where $s$ is the number of individuals at the generation $t$, randomly;
4. For each individual $i$ in $P(t)$
5. Solve the linear system described in Eq. (3) by Eq. (4);
6. Evaluate the fitness function shown in Eq. (6);
7. While $t \leq t_{max}$, where $t_{max}$ is the maximum number of generations
8. Select individuals $i$ from the population $P(t)$;
9. Apply crossover operation to selected individuals;
10. Apply mutation operation to selected individuals;

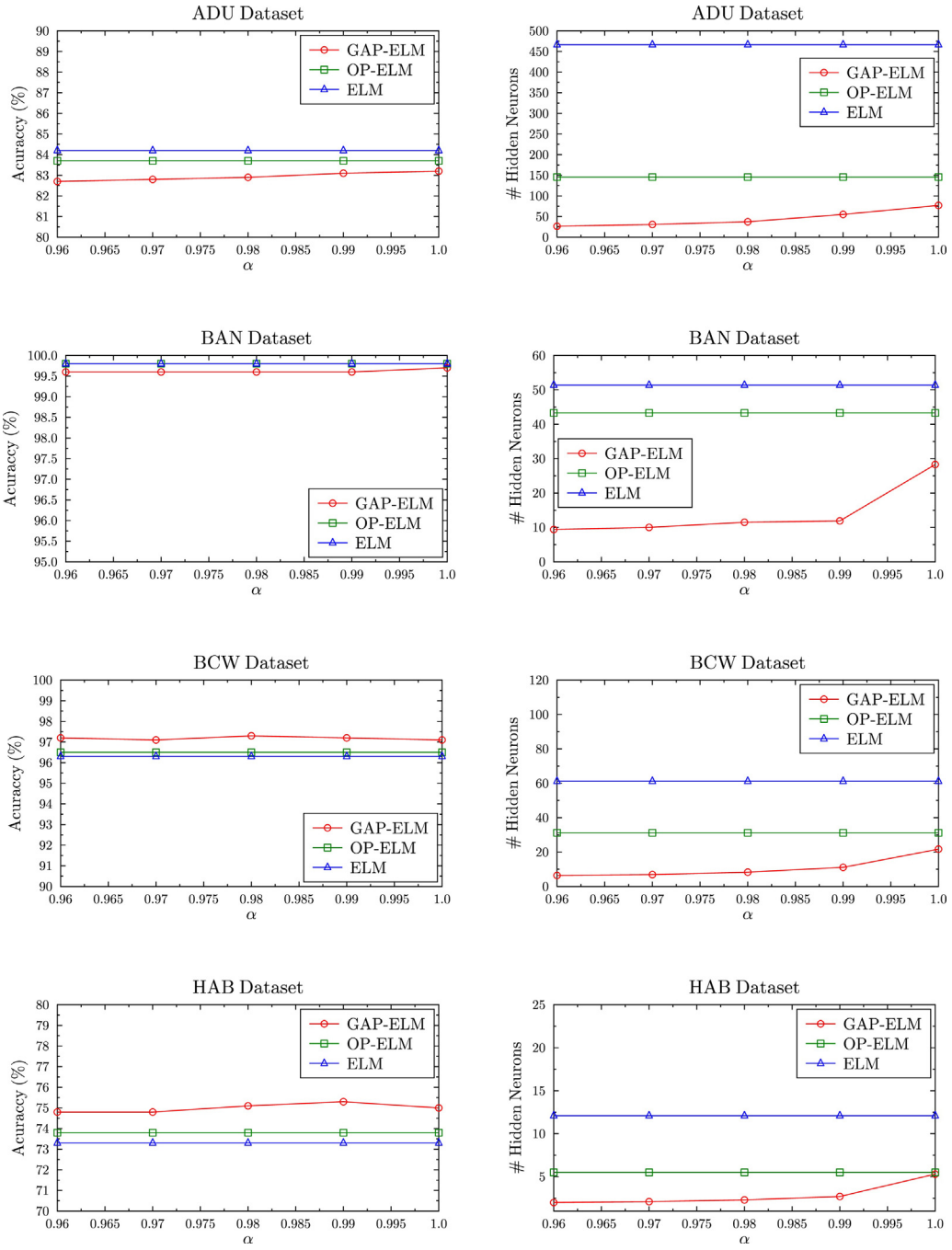


**Fig. 2.** (a) Accuracies versus the factor $\alpha$ in range [0.96, 1.00] for the ADU, BAN, BCW and HAB dataset, respectively. (b) Mean of # hidden neurons versus the factor $\alpha$ for ADU, BAN, BCW and HAB datasets, respectively.

11. Compute $t = t + 1$;
12. Solve the linear system described in Eq. (3) by Eq. (4);
13. Evaluate the fitness function shown in Eq. (6);
14. Select the best individual or solution, i.e., $\mathbf{g}^o$.

## 6. Simulations and discussion

We carried out simulations so that 80% of the full data set were randomly selected for training purposes and the remaining 20% of the examples were used to assess the classifier generalization performances. Tests with real-world benchmarking datasets were also evaluated in this work. We used UCI machine learning datasets: Adult, Breast Cancer Winconsin, Haberman, HIV, Pima Indians Diabetes and Ripley. Some information about evaluated data sets is presented in Table 1, such as data set name, abbreviation and number of Patterns (# Patterns). One can notice that we have handled large data sets such as Adult, HIV and Ripley with 48808, 3272 and 1250 patterns, respectively.

The training process for ELM were based on a grid search with 10-fold cross-validation to tune the number of hidden neurons. The activation functions for all ELM hidden neurons were the hiperbolic tangent. For OP-ELM and GAP-ELM, we defined the number of hidden neuron at the beginning of training process as the average of hidden neurons for ELM, hence OP-ELM and GAP-ELM started with the same number of neurons. The idea is to have a comparison between OP-ELM and GAP-ELM when a small number of hidden neurons is used.

Initially, the GA randomly creates a population of feasible solutions. Each solution is a string of binary values in which each bit represents the presence (1) or absence (0) of a neuron in the hidden layer. For the next generation, we take into account the fact that 10% of the best individuals are selected due to a elitist selection scheme, 80% of new individuals were generated as result of applying the crossover operator and then the remaining individuals were obtained by mutation. Our population has 50 individual and the maximum number of generations is 300. However, in general, the algorithm stops when the average relative change in the best fitness function value is less than or equal to $10^{-5}$.

In Table 2, we report performance metrics (mean value and standard deviation of the recognition rate, i.e., accuracy) on testing set averaged over 30 independent runs. We also show the average number of hidden neurons (#HN) and training time cost (#TC) during all the training process.

By analyzing Table 2, one can infer that the accuracies of the GAP-ELM were equivalent to those achieved by the ELM and OP-ELM. In some cases, as shown in this table for Haberman (HAB) and Breast Cancer (BCW) the accuracies of GAP-ELM classifiers were even better. On the other hand, it is possible to notice a significant reduction in the number of hidden neurons provided by GAP-ELM when compared to OP-ELM.

In addition to the previous results, Fig. 2(a) shows the accuracies of OP-ELM, ELM and GAP-ELM and Fig. 2(b) shows the average number of hidden neurons for $\alpha$ in range [0.96, 1.00] for 30 independent runs with all data sets when applied to ADU, BAN, BCW and

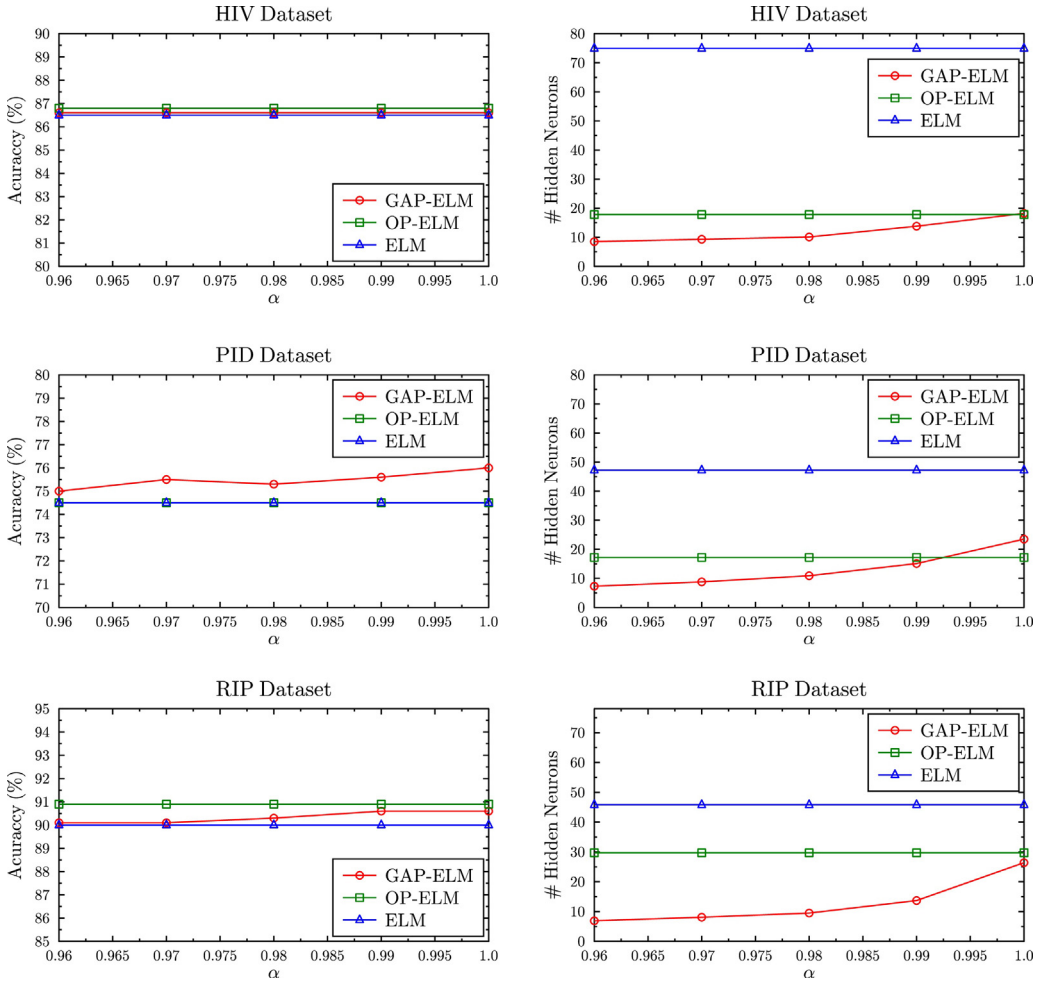


**Fig. 3.** (a) Accuracies versus the factor $\alpha$ in range [0.96, 1.00] for the HIV, PID and RIP dataset, respectively. (b) Average number of hidden neurons versus the factor $\alpha$ for the HIV, PID and RIP datasets, respectively.

HAB datasets. Similar results are presented in Fig. 3(a) and Fig. 3(b) for HIV, PID and RIP datasets.

As expected, when $\alpha$ is increased, GAP-ELM accuracies tend to increase and so the number of hidden neurons. Although GAP-ELM seems to be sensible to the value of $\alpha$, it is possible to notice that in almost all cases presented (with different $\alpha$), GAP-ELM reduced the number of hidden neurons while keeping the accuracy at a similar level.

Another interesting point can be observed when $\alpha$ is set to 1.0. In this situation, the multiobjective formulation is turned into a single objective fitness function that only considers the accuracy. As a result, GAP-ELM achieved similar (or even higher) accuracies than OP-ELM and ELM in almost all datasets with a number of neurons similar to OP-ELM.

A second set of experiments was conducted with a fixed number of hidden neurons. Along with ELM, OP-ELM and GAP-ELM, two widely used SFLN were assessed: the Radial Basis Function (RBF) and MLP neural networks. For the experiments, 40 hidden neurons were used for MLP, RBF and ELM. OP-ELM and GAP-ELM started the pruning process with 40 hidden neurons. Results are shown in Table 3.

Similarly to that observed in previous experiments, GAP-ELM achieved accuracies comparable to both OP-ELM and ELM but with significant reduction on the number of hidden neurons. On the other hand, it is possible to notice an increase in the training time. When comparing GAP-ELM with the RBF and MLP, it is noticeable that the accuracies were similar, however GAP-ELM presents a faster training procedure and also less hidden neurons.

## 7. Conclusions

In this work, we propose a pruning method for ELMs based on Multiobjective GAs. The proposed approach was called Genetic Algorithms for Pruned ELM (GAP-ELM). GAP-ELM is built upon a multiobjective fitness function considering both the accuracy and the reduction in the number of hidden neurons. The accuracy is estimated by the use of a Leave-one-out cross validation procedure called PRESS statistic. The use of PRESS statistic avoids computations related to standard cross validation procedures and thus reduces the final computational cost of GAP-ELM.

The performance of GAP-ELM was assessed on 7 real world datasets and compared to ELM, RBF, MLP and OP-ELM. All methods were compared according to three criteria: accuracy, number of hidden neurons and training time. On the basis of our experiments we can state that GAP-ELM is a valid alternative for classification tasks, since none of the methods was capable to outperform GAP-ELM in all performance criteria at the same time. Future works include the design of an ELM pruning method with a fixed number of neurons in the hidden layer.

## References


[1] R. Ahila, V. Sadasivam, K. Manimala, An integrated {PSO} for parameter determination and feature selection of {ELM} and its application in classification of power system disturbances, Appl. Soft Comput. 32 (2015) 23–37.

[2] D.P. Mesquita, J.P.P. Gomes, L.R. Rodrigues, R.K.H. Galvao, Pruning extreme learning machines using the successive projections algorithm, IEEE Latin Am. Trans. 13 (12) (2015) 3974–3979.

[3] S. Haykin, Neural Networks and Learning Machines, 3rd ed., Prentice Hall, 2009.

[4] G.B. Huang, Q.Y. Zhu, C.K. Ziew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.

[5] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995.

[6] H.-J. Rong, Y.-S. Ong, A.-H. Tan, Z. Zhu, A fast pruned-extreme learning machine for classification problem, Neurocomputing 72 (1–3) (2008) 359–366.

[7] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, A. Lendasse, Op-elm: optimally pruned extreme learning machine, IEEE Trans. Neural Netw. 21 (1) (2010) 158–162.

[8] Y. Miche, M. van Heeswijk, P. Bas, O. Simula, A. Lendasse, Trop-elm: a double-regularized {ELM} using {LARS} and tikhonov regularization, Neurocomputing 74 (16) (2011) 2413–2421.

[9] G.-B. Huang, L. Chen, Enhanced random search based incremental extreme learning machine, Neurocomputing 71 (16–18) (2008) 3460–3468.

[10] G.-B. Huang, M.-B. Li, L. Chen, C.-K. Siew, Incremental extreme learning machine with fully complex hidden nodes, Neurocomputing 71 (4–6) (2008) 576–583.

[11] G. Feng, G.-B. Huang, Q. Lin, R. Gay, Error minimized extreme learning machine with growth of hidden nodes and incremental learning, IEEE Trans. Neural Netw. 20 (8) (2009) 1352–1357.

[12] J. Luo, C.-M. Vong, P.-K. Wong, Sparse Bayesian extreme learning machine for multi-classification, IEEE Trans. Neural Netw. Learn. Syst. 25 (4) (2014) 836–843.

[13] A.E.R. Palencia, G.E.M. Delgadillo, A computer application for a bus body assembly line using genetic algorithms, Int. J. Prod. Econ. 140 (1) (2012) 431–438.

[14] Z.E. Brain, M.A. Addicoat, Using meta-genetic algorithms to tune parameters of genetic algorithms to find lowest energy molecular conformers, in: ALIFE, 2010, pp. 378–385.

[15] R. Sharma, A.S. Bist, Genetic algorithm based weighted extreme learning machine for binary imbalance learning, in: International Conference on Cognitive Computing and Information Processing (CCIP), 2015, pp. 1–6.

[16] T. Matias, R. Araujo, C. Henggeler Antunes, D. Gabriel, Genetically optimized extreme learning machine, in: IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA), 2013, pp. 1–8.

[17] D. Lahoz, B. Lacruz, P. Mateo, A bi-objective micro genetic extreme learning machine, in: IEEE Workshop On Hybrid Intelligent Models And Applications (HIMA), 2011, pp. 68–75.

[18] T. Similä, J. Tikka, Multiresponse sparse regression with application to multidimensional scaling, in: Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005, Vol. 3697 of Lecture Notes in Computer Science, Springer, Berlin Heidelberg, 2005, pp. 97–102.

[19] J. Dubois-Lacoste, M. Lopez-Ibanez, T. Stutzle, Combining two search paradigms for multi-objective optimization: two-phase and pareto local search, in: Hybrid Metaheuristics, 2013, pp. 97–117.

[20] R. Myers, Classical and Modern Regression with Applications, in: Bookware Companion Series, PWS-KENT, 1990.