Evolving Extreme Learning Machine Paradigm with Adaptive Operator Selection and Parameter Control

Ke Li, Ran Wang, Sam Kwong, Jingjing Cao

Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong.

keli.genius@gmail.com, ranwang3@student.cityu.edu.hk, cssamk@cityu.edu.hk, bettymore0501@gmail.com

Received (received date)
Revised (revised date)

Extreme Learning Machine (ELM) is an emergent technique for training *Single-hidden Layer Feedforward Networks* (SLFNs), which attracts huge amounts of interests during the recent years. However, the randomly assigned network parameters in the original ELM algorithm might cause high learning risks. Motivated by this matter of fact, in this paper, an evolving ELM paradigm is proposed for classification problems. In this learning paradigm, a *Differential Evolution* (DE) variant, which can online select the appropriate operator for offspring generation and adaptively adjust the corresponding control parameters, is proposed for optimizing the network parameters. In addition, a 5-fold cross validation is adopted in the fitness assignment procedure, for improving the generalization capability of the baseline ELM. Empirical studies on several real-world classification data sets have demonstrated that the evolving ELM paradigm can generally outperform the original ELM as well as several recent classification algorithms.

Keywords: Extreme learning machine, Adaptive operator selection, Parameter control, Differential evolution

1. Introduction

In machine learning and statistics, classification is the problem of identifying which of a set of categories (or classes) a new observation belongs to, on the basis of a training set containing observations whose classificatory memberships are known. Various techniques have been developed for solving classification problems, such as *Decision Trees* (DTs) [18], *Artificial Neural Networks* (ANNs) [1] and *Support Vector Machines* (SVMs) [29], while the performance of a classifier can be affected by several factors, such as the degree of data integrity and the parameter settings of the given model.

ANN, also called *Neural Network* (NN), is a mathematical model inspired by the structural and functional aspects of biological learning systems which are built of very complex networks of interconnected neurons. NN has been widely used in various fields due to its two prominent abilities:

(1) It can approximate complex nonlinear mappings directly from the input sam-

ples.

(2) It can provide models for a large variety of natural and artificial scenarios that are difficult to handle by traditional techniques.

As for solving pattern recognition problems, NN is considered as a supervised learning technique to construct mappings from the feature vectors to the corresponding classes. Once the mapping model has been constructed, it can generally categorize unseen data into one of the classes, according to the rules obtained from the training process. The most successful algorithms for training NNs are known as the gradient based techniques, such as the *Back-Propagation* (BP) and its variant *Levenberg-Marquardt* (LM). However, they suffer from three disadvantages which hold back their further developments:

(1) Gradient based algorithms are relatively slow due to the large number of iterations.
(2) They may easily get stuck in local optima.
(3) The activation functions used by them are usually with specific requirements, such as the differentiability.

Extreme Learning Machine (ELM) [12] is an emergent technique for training *Single-hidden Layer Feed-forward Neural networks* (SLFNs). Instead of adjusting the network parameters iteratively, in ELM, the input weights and hidden biases are chosen randomly while the output weights are calculated analytically by using *Moore-Penrose* (MP) generalized inverse. ELM has an error bound similar to RBFNN [8] which is experimentally proved to have a accuracy better than rule-based methodologies [36] [31] [28] [30]. In the past few years, much effort has been devoted to applying ELM for various real-world problems [3] [24]. A recent comprehensive survey on ELM can be found in [10]. Compared with gradient based methods, ELM not only obtains faster training speed and better generalization performance, but also avoids the problem of local optima. However, it is argued that the randomly assigned input parameters may affect the final performance a lot and the selection of appropriate parameters can reduce the learning risk of ELM [23]. Moreover, ELM may require more hidden neurons than conventional tuning based learning algorithms, in some applications, which may make ELM respond slowly to unknown data [39].

Evolutionary Algorithms (EAs) are stochastic global optimization techniques inspired by the Darwinian evolution theory. Different types of EAs have been proposed, such as *Genetic Algorithms* (GA) [6], *Differential Evolution* (DE) [26], *Ant Colony Optimization* (ACO) [5] and *Particle Swarm Optimization* (PSO) [14]. One of the important features of EAs is their population based search manner, which enables individuals in the population to compete and exchange information with each other during the optimization process. This also reduces the venture of EAs being trapped by local optima. Instead of depending on the gradient information, the search process of EAs is usually guided by some performance measurements (such as how good a given set of network parameters is), which are in the form of fitness function. In this case, individuals with better fitness values are able to survive for the next generation. Thus, EAs are quite suitable for handling problems even without good mathematical properties, and this also makes them more robust than many other search algorithms. As discussed in [35], the concept of evolution can be introduced to the training of NNs from three different levels:

(1) The evolution of the network parameters: it aims at finding a near optimal set of network parameters globally within a fixed network architecture.
(2) The evolution of network architectures: it enables NNs to adapt their topologies to different tasks without human intervention and thus provides an approach to automatic network design.
(3) The evolution of learning rules: this can be regarded as a process of "learning to learn", in which NNs can adapt the appropriate learning rules to the environments via evolution. It can also be regarded as an adaptive process of automatic discovery of novel learning rules.

This paper focuses on the first level mentioned above and proposes an evolving ELM paradigm, which hybridizes one of the EA variants, i.e. DE, with the ELM algorithm to optimize the intrinsic network parameters during the training process. The DE variant under use is with adaptive operator selection and parameter control. In particular, four DE mutation operators with diverse search characteristics are combined to form an operator pool, and the proposed DE algorithm can online select the appropriate operator for the current offspring generation. Meanwhile, the hyper-parameters related to those operators are tuned adaptively according to the current fitness landscapes. Moreover, the generalization performance of the classifier and the norm of the output weights are combined to form a fitness function, which is used to evaluate the quality of an individual.

The remainder of this paper is organized as follows: Section 2 presents some background descriptions on the ELM classifier and DE algorithm. Section 3 gives the algorithmic details of our proposed evolving ELM paradigm. Section 4 presents and analyzes the empirical results of the proposed algorithm on several real-world data sets. Finally, Section 5 gives the conclusions of this study and highlights some possible future directions.

2. Background and related works

In this section, some background knowledge about the ELM classifier and DE algorithm are given at first, while several related works on hybridizing EAs with ELM for training NNs are then reviewed.

2.1. *Extreme learning machine*

In [11], Huang et al. first proposed ELM, in which the input weights and hidden biases of a SLFN are chosen randomly and the output weights are calculated analytically. As for the hidden neuron layer, the activation function can be designed as sigmoid, sine, Gaussian, and hard-limiting functions, while for the output neurons, the function is usually linear.

Suppose that we have N arbitrary distinct samples (x_i, y_i), where $x_i = [x_{i1}, x_{i2}, \cdots, x_{in}]^T \in R^n$ and $y_i = [y_{i1}, y_{i2}, \cdots, y_{im}]^T \in R^m$. Let H be the number of hidden neurons, and $w_i = [w_{i1}, w_{i2}, \cdots, w_{in}]^T$ be the input weight vector connecting the ith hidden neuron and the input neurons, b be the $H \times 1$ bias values for each hidden neuron and β be the $H \times m$ output weight matrix, where $\beta_i = [\beta_{i1}, \beta_{i2}, \cdots, \beta_{im}]^T$ is the weight vector connecting the ith hidden neuron and the output neurons. In general, the ELM network structure can be mathematically modeled as:

$$y_j = \sum_{i=1}^{H} \beta_i g(w_i \cdot x_j + b_i) \tag{1}$$

where $j \in \{1, 2, \cdots, N\}$, $w_i \cdot x_j$ denotes the inner product of w_i and x_j, and $g(x)$ is the activation function. Without loss of generality, the sigmoid additive activation function is chosen as $g(x)$ here, which is formulated as:

$$g(x) = \frac{1}{1 + exp[-(w \cdot x + b)]} \tag{2}$$

It is worth noting that equation (1) can also be written in a matrix form as:

$$Y = Y_H \beta \tag{3}$$

where Y_H is the $N \times H$ hidden layer output matrix [12], and it can be further defined as:

$$Y_H = \begin{pmatrix} g(w_1 \cdot x_1 + b_1) & \cdots & g(w_H \cdot x_1 + b_H) \\ \vdots & \ddots & \vdots \\ g(w_1 \cdot x_N + b_1) & \cdots & g(w_H \cdot x_N + b_H) \end{pmatrix} \tag{4}$$

where the ith column of Y_H is the output of the ith hidden neuron with respect to inputs $x_1, x_2, \cdots, x_N$.

In the original ELM algorithm, the input weights and hidden biases are randomly chosen, while the output weights are determined by the least square solution to the linear system given in equation (3). In practise, the output weights, i.e. the minimum norm least square solution to equation (3), are calculated analytically as:

$$\beta = Y_H^{\dagger} Y \tag{5}$$

where $Y_H^{\dagger}$ is the Moore-Penrose (MP) pseudo-inverse of the hidden layer output matrix Y_H, and $Y = [y_1, y_2, \cdots, y_N]^T$. In summary, the pseudo code of the ELM algorithm is given in Algorithm 1.

Algorithm 1: Extreme Learning Machine

```
1 Fix the number of hidden neurons and the activation function for a given problem;
2 Initialize the input weight matrix $W$ and hidden bias vector $b$;
3 Calculate the hidden layer output matrix $Y_H$ using equation (4);
4 Calculate the output weight matrix $\beta$ using equation (5);
```

2.2. *Differential evolution*

Similar to other EAs, DE is a population based stochastic optimization algorithm which is originally proposed for solving global optimization problems in continuous search space [26]. The pseudo code of a basic DE is outlined in Algorithm 2, where NP is the population size; F is the mutation scaling factor; CR is the crossover rate; $x_{i,j}$ is the jth decision variable of the target vector x_i; v_i is the mutant vector; and u_i is the trial vector. $rnd(1, D)$ generates a random integer uniformly distributed over 1 to n; $rndreal()$ returns a random real number $x \in [0, 1]$.

Algorithm 2: Differential Evolution

```
1  Randomly generate an initial population;
2  Evaluate the fitness value of each individual;
3  while the stopping criterion is not satisfied do
4      for $i \leftarrow 1$ to $NP$ do
5          Randomly select $r_1, r_2, r_3 \in \{1, 2, \ldots, NP\}$ such that $r_1 \neq r_2 \neq r_3 \neq i$;
6          $j_{rand} = rnd(1, D)$;
7          for $j \leftarrow 1$ to $D$ do
8              if $rndreal() < CR$ or $j = j_{rand}$ then
9                  $u_{i,j} = x_{r_1,j} + F \times (x_{r_2,j} - x_{r_3,j})$;
10             else
11                 $u_{i,j} = x_{i,j}$;
12             end
13         end
14     end
15     for $i \leftarrow 1$ to $NP$ do
16         Evaluate the fitness value of trial vector $u_i$;
17         if $f(u_i)$ is better than or equal to $f(x_i)$ then
18             Replace $x_i$ with $u_i$;
19         end
20     end
21 end
```

In Algorithm 2, without loss of generality, we only take the classical mutation operator "DE/rand/1" as an example for illustration, although many other mutation operators have also been proposed [21]. This operator is featured by less greed, slower convergence speed and more reliability, and has been widely used in this literature [4]. Its expression is formulated as:

$$v_{i,j} = x_{r_1,j} + F \times (x_{r_2,j} - x_{r_3,j}) \tag{6}$$

where x_{r_1}, x_{r_2} and x_{r_3} are different from x_i. After the generation of v_i, a crossover operation is evoked to generate u_i. In general, there are two different crossover schemes, i.e. *exponential* and *binomial* [26], while we only elaborate the latter one in Algorithm 2 without loss of generality. This crossover scheme can be outlined as:

$$u_{i,j} = \begin{cases} v_{i,j}, & \text{if } rndreal() < CR or j = j_{rand} \\ x_{i,j}, & \text{otherwise} \end{cases} \tag{7}$$

It is worth noting that each decision variable, say $x_{i,j}$, has its own feasible range, i.e. $x_{i,j} \in [L_j, U_j]$. Thus, if the newly generated $u_{i,j}$ is out of this feasible region, the following repairing strategy is applied to set it back to its valid domain:

$$u_{i,j} = \begin{cases} \min\{U_j, 2 \times L_j - u_{i,j}\}, & \text{if } u_{i,j} < Lj \\ \max\{L_j, 2 \times U_j - u_{i,j}\}, & \text{if } u_{i,j} > U_j \end{cases} \tag{8}$$

After the generation of all trial vectors, a selection procedure is performed to select the better ones, from the target vectors and the trial vectors, to form the parent population for the next generation. In particular, the objective function $f(x_i)$ is to be maximized here.

$$x_i = \begin{cases} u_i, & \text{if } f(u_i) \geq f(x_i) \\ x_i, & \text{otherwise} \end{cases} \tag{9}$$

2.3. *Methods of tuning ELM parameters*

As discussed in Section 1, ELM can achieve a good generalization performance with an extremely fast learning speed, comparing to the traditional gradient based learning algorithms. However, the network parameters, i.e. input weights and hidden biases, can largely influence the performance of ELM. This can be explained by the fact that ELM tries to find the global optimum of a localized linear system while this might not be the global optimum in the problem space. Generally speaking, this kind of issue can be solved in two different ways:

(1) One can increase the size of the hidden units that make the randomly assigned network parameters be very close to their best settings.
(2) One can also adopt some heuristic methods to find the optimal settings of those parameters.

So far, several works have been proposed for the problem of finding the near optimal network parameters of ELM. In [16], Liu et al. proposed an ensemble based

ELM (EN-ELM), in which ensemble learning and cross validation are embedded into the training phase of ELM, in order to minimize the jeopardize caused by the overfitting problem and thus enhance the predictive stability. Miche et al. [17] proposed an optimally pruned ELM (OP-ELM), which assumes that not all neurons are necessary for training, and it improves the ELM by pruning the useless neurons. Cao et al. [2] presented a voting based ELM (V-ELM) by employing ELM as base classifier under the framework of majority voting scheme. Zhu et al. [39] presented an evolutionary ELM, called E-ELM, in which the basic DE algorithm is employed to evolve the network parameters. Based on the same framework of E-ELM, Silva et al. [25] and Xu et al. [34] proposed other two evolutionary ELMs. The only differences of the latter two are their evolutionary parts, which replace the original DE algorithm with *Group Search Optimization* (GSO) [9] and PSO, respectively. In [23], Saraswathi et al. proposed a hybrid EA with ELM for gene selection and cancer classification. In their proposal, the *Integer Coded Genetic Algorithm* (ICGA) is used to select an optimal subset of genes for constructing the classifier. At the same time, PSO is employed as a local optimizer to optimize the input weights, hidden biases and the number of hidden neurons. This algorithm, called ICGA-PSO-ELM, shows good performance on the sparse and imbalance data sets.

3. Evolving ELM Learning Paradigm

In this section, the chromosome design, fitness function, the proposed DE algorithm with adaptive operator selection and parameter control, and the system architecture of the proposed evolving ELM paradigm are described in detail as follows.

3.1. *Chromosome design*

Chromosome design is an important step of EAs. Different from [38], in which the feature chromosome is coded as a "0–1" binary string, here each allele represents a specific value of the network parameter as shown in Figure 1 and equation (10).

$$C = [w_{11}, \cdots, w_{1H}, w_{22}, \cdots, w_{2H}, \cdots, w_{n1}, \cdots, w_{nH}, b_1, \cdots, b_H] \tag{10}$$

More specifically, the chromosome comprises two parts, i.e. the input weights w and hidden biases b. Each input weight $w_{i,j}$ is initialized within the range of $[-1, 1]$ while the hidden bias b_i is initialized within the range of $[0, 1]$, according to the suggestion in [12].

3.2. *DE with adaptive operator selection and parameter control*

The general framework of the proposed DE algorithm with adaptive operator selection and parameter control is given in Figure 2. As can be seen, it is divided into three modules. In the middle, there is the main cycle of the DE algorithm, represented here by only three steps for the sake of brevity: once after the generation of the offsrping population, the fitness value (see Section 3.3) of each individual is

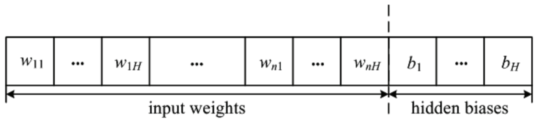


Fig. 1: The chromosome coding comprises two parts: the coding of input weights and the coding of hidden biases.

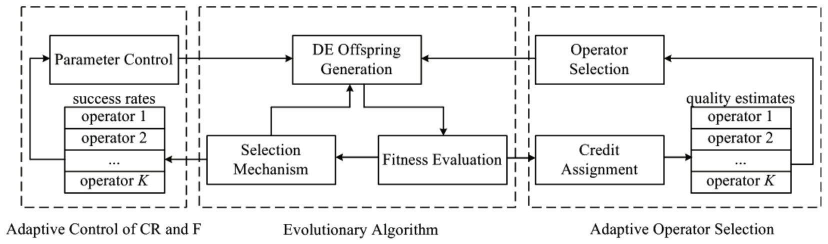


Fig. 2: The framework of the proposed DE algorithm with adaptive operator selection and parameter control

evaluated according to the generalization performance of the corresponding ELM model. This fitness measure is used as the criteria in the selection mechanism, that decides which of the individuals can survive for the next generation. Two adaptive mechanisms are employed in parallel. On the right hand side, there is the adaptive operator selection module which can select operators online based on their previous performances; while the parameter control module, which adjusts the parameters adaptively, is laid on the left hand side. Both adaptive mechanisms are described, respectively, in Sections 3.4 and Section 3.5.

3.3. *Fitness function design*

Fitness function is used to evaluate the quality of a specific individual. It is also a crucial point in the design of EAs, which determines the target an EA can optimize. In this paper, an individual's quality in terms of the classification accuracy is obtained by ELM models using K-fold cross validation with the training set. In particular, the training set is divided into K subsets, each of which acts as an independent holdout set for testing the model that is trained with the other $K-1$ subsets. The testing accuracy that measures the percentage of samples that are correctly classified is calculated K times, and then the cross validation accuracy (denoted as $E_{accuracy}$) is finally determined by averaging these K testing accuracies, as defined in the following equation:

$$E_{accuracy} = \frac{\sum_{i=1}^{K} TestingAccuracy_i}{K} \tag{11}$$

The advantage of cross validation is that the testing sets are independent from each other and the reliability of the results thus can be improved [15]. Specifically, $K = 5$ is employed for the calculation in this paper. On the other hand, as discussed in [39], the trained NN tends to have better generalization performance with small output weights. In this case, we further take the average norm of the output weights (denoted as $\|\beta\|$) into consideration. Finally, these above two evaluation criteria are combined into the so called fitness function which is used to evaluate the quality of each individual. Its formulation is represented as follows:

$$fitness = E_{accuracy} - \alpha_w \times \|\beta\| \tag{12}$$

where α_w is a user defined parameter to adjust the penalty term to the cross validation accuracy. From our preliminary experiments, we set $\alpha_w = 0.01$ without loss of generality. It is obvious that the larger the *fitness* is, the better the individual would be.

3.4. *Adaptive operator selection*

Adaptive Operator Selection (AOS) paradigm aims at autonomously controlling which operator (i.e. mutation operator in the case of DE algorithm) should be applied at each time point of the search, when solving the problem, based on their recent performances. Generally speaking, the AOS paradigm proposed in this paper consists of two components (as shown in the right hand side of Figure 2): the credit assignment module defines how each operator should be rewarded based on the impacts of its recent applications during the optimization progress; and the operator selection mechanism decides which of the available operators should be applied next, according to their respective empirical quality estimates, which are built and constantly updated by the corresponding rewards. Each of these components will be now detailed in turn.

3.4.1. *Credit assignment module*

In order to assign the credit value for an applied operator, in this work, we adopt the impact assessment scheme used in [20] as follows:

$$\xi = \frac{cf_i}{\varphi} \times |pf_i - cf_i| \tag{13}$$

where $i \in \{1, 2, \cdots, NP\}$. φ is the best-so-far fitness value in the current population. pf_i and cf_i are the fitness values of the parent solution and its corresponding offspring, respectively. It is worth noting that ξ is set as zero if no improvement is achieved after the application of an operator. In this case, the fitness value of the offspring should be worse than or equal to that of its parent, thus the impact of the application of this operator is considered to be null.

All impacts achieved by the application of mutation operator $a \in \{1, \cdots, K\}$ during generation g are stored in a specific set R_a. At the end of each generation

g, a unique credit (or reward) value is assigned to each operator, based on the values stored in R_a. It is calculated as the average of all impacts achieved by the application of operator a:

$$r_a(g) = \sum_{i=1}^{|R_a|} \frac{R_a(i)}{|R_a|}. \tag{14}$$

3.4.2. *Operator selection: probability matching*

The operator selection mechanism usually selects the appropriate operator, according to the empirical quality estimates it keeps for each operator. These estimates are built and constantly updated by the received credit values. In this work, we apply the state-of-the-art operator selection mechanism *Probability Matching* (PM) [27] which is formally described as follows. Let the set (or pool) of operators be denoted by $S = \{s_1, \ldots, s_K\}$ where $K > 1$. The probability vector $P(g) = \{p_1(g), \ldots, p_K(g)\}(\forall t : p_{min} \leq p_i(g) \leq 1; \sum_{i=1}^{K} p_i(g) = 1)$ represents the selection probability of each operator at generation g. An estimation of the current empirical quality estimate of operator i is denoted as $\hat{q}_i(g)$. Based on these definitions, we have the following two points at each generation g:

(1) the ith operator is selected, by roulette wheel selection, according to the probability $p_i(g)$, and obtains a reward $r_i(g)$ from the credit assignment module under use;
(2) the empirical quality estimate of the ith operator $\hat{q}_i(g)$ is updated using an additive relaxation mechanism with adaptation rate $\alpha \in [0, 1]$:

$$\hat{q}_i(g+1) = (1-\alpha) \times \hat{q}_i(g) + \alpha \times r_i(g) \tag{15}$$

After the calculation of the empirical quality estimation, the PM method updates the selection probability as follows:

$$p_i(g+1) = p_{min} + (1 - K \times p_{min}) \times \frac{\hat{q}_i(g+1)}{\sum_{i=1}^{K} \hat{q}_i(g+1)} \tag{16}$$

where $p_{min} \in [0, 1]$ is the minimal selection probability value for each operator. This parameter is used to ensure that all operators will always have a minimal chance of being selected, in order to avoid "losing" a currently bad operator that might become useful at a further moment of the search. Then, any inefficient operator (getting only null credits, i.e., no impact) will have its selection probability converging towards p_{min}, while the best operator (getting very good credits after some time) will be selected with probability $p_{max} = 1 - (K-1) \times p_{min}$.

3.5. *Adaptive parameter control of CR and F*

The adaptive parameter control method used here is inspired by the mechanism used in [37] and [13]. Let CR_i^a denotes the crossover rate for the individual i using operator

$a \in \{1, \ldots, K\}$. At each generation, CR_i^a is independently generated according to the normal distribution with mean μ_{CR}^a and standard deviation 0.1:

$$CR_i^a = norm(\mu_{CR}^a, 0.1) \tag{17}$$

CR_i^a is regenerated whenever it exceeds 1. All successful crossover rates at generation g for operator a are stored in a specific set denoted as S_{CR}^a. In particular, the crossover rates, which can generate offspring successfully survive for the next generation, are considered as available ones. The mean μ_{CR}^a is initialized to be 0.5 and updated after each generation as:

$$\mu_{CR}^a = (1 - w_{cr}) \times \mu_{CR}^a + w_{cr} \times mean(S_{CR}^a) \tag{18}$$

where $mean(S_{CR}^a)$ is the arithmetic mean of values in S_{CR}^a and w_{cr} is a random number generated as follows:

$$w_{cr} = 0.2 \times rndreal() \tag{19}$$

The adaptation of the scaling factor F_i^a is similar to that of CR_i^a. At each generation, F_i^a is independently generated as:

$$F_i^a = Cauchy(\mu_F^a, 0.1) \tag{20}$$

where $Cauchy(\mu_F^a, 0.1)$ is a random number sampled from a Cauchy distribution with location parameter μ_F^a and scale factor 0.1. The value of F_i^a is regenerated whenever F_i^a is out of the range $[0, 1]$. All successful scaling factors at the current generation for operator a are stored in a specific set denoted as S_F^a. The mean μ_F^a is initialized to be 0.5 and independently updated after each generation as:

$$\mu_F^a = (1 - w_f) \times \mu_F^a + w_f \times mean_r(S_F^a) \tag{21}$$

where $mean_r(S_{CR}^a)$ is the root-mean-square of values in S_F^a and w_f is a random number generated as:

$$w_f = 0.1 \times rndreal() \tag{22}$$

3.6. *Operator pool*

Many different DE mutation operators have been proposed in this literature [21], with each one owning its distinctive search behavior. As discussed in [22], the theoretical studies on the choice of the optimal number of operators in the pool and the organization of the pool with appropriate operators are still open. In this work, we generally maintain an operator pool including four well-known DE mutation operators with effective yet diverse characteristics, which are described as follows.

(1) DE/rand/1:

$$v_{i,j} = x_{r_1,j} + F \times (x_{r_2,j} - x_{r_3,j}) \tag{23}$$

(2) DE/best/1:

$$v_{i,j} = x_{best,j} + F \times (x_{r_1,j} - x_{r_2,j}) \tag{24}$$

(3) DE/rand/2:

$$v_{i,j} = x_{r_1,j} + F \times (x_{r_2,j} - x_{r_3,j}) + F \times (x_{r_4,j} - x_{r_5,j}) \tag{25}$$

(4) DE/current-to-rand/1:

$$v_{i,j} = x_{i,j} + rnderal() \times (x_{r_1,j} - x_{i,j}) + F \times (x_{r_2,j} - x_{r_3,j}) \tag{26}$$

where x_{r_1}, x_{r_2}, x_{r_3}, x_{r_4}, x_{r_5} and x_i are different from each other. "DE/rand/1" operator is the most commonly used operator in this literature. This operator randomly selects individuals from the population for mutation, it thus has no bias towards any special search directions and bears strong exploration ability, but with slow convergence speed. "DE/best/1" operator utilizes the information of the best solution (denoted as x_{best}) found so far, therefore, it usually has a fast convergence speed, but with the risk of being stumbled by the local optima. "DE/rand/2" operator is an extension of "DE/rand/1", in which two difference vectors are added to the base vector. As discussed in [32], it might lead to better perturbation than those with only one difference vector. In "DE/current-to-rand/1", a rotation invariant arithmetic crossover is performed after the mutation, so that it is more suitable for rotated problems.

3.7. *Algorithm Composition*

By combining the above discussed issues, our proposed DE algorithm with adaptive operator selection and parameter control (denoted as AdapDE), is illustrated in Algorithm 3. At each generation, for each target vector x_i, an operator a is independently selected from the operator pool, according to its selection probability, to generate the offspring. After the generation of the entire offspring population, their qualities are evaluated by the fitness function as equation (12), and then the impact ξ, related to the application of operator a, is calculated and stored in the set R_a correspondingly. Consequently, the reward, quality, and probability of each operator are updated based on the criteria discussed in the previous sections.

3.8. *System Architecture of the evolving ELM paradigm*

The system architecture of the evolving ELM paradigm (denoted as Evo-ELM) based on the proposed AdapDE is shown in Figure 3. Some of the major steps are described specifically in the following.

(1) Data preprocessing: In order to avoid the problem that attributes in greater numeric ranges dominate those in smaller numeric ranges, we adopt the linear scaling to preprocess the input data. According to equation (27), each feature value of the data set can be linearly scaled to the range $[0, 1]$.

$$v^* = \frac{v - \min}{\max - \min} \tag{27}$$

Algorithm 3: DE with adaptive operator selection and parameter control (AdapDE)

```
Randomly generate an initial population;
Evaluate the fitness value of each individual;
Set the generation counter $g = 1$;
while the stopping criterion is not satisfied do
    for $i \leftarrow 1$ to $NP$ do
        Select the operator $a$ based on its selection probability;
        Randomly select $r_1, r_2, r_3, r_4, r_5 \in \{1, 2, \ldots, NP\}$ such that
        $r_1 \neq r_2 \neq r_3 \neq r_4 \neq r_5 \neq i$;
        $j_{rand} = rnd(1, D)$;
        for $j \leftarrow 1$ to $D$ do
            if $rndreal() < CR$ or $j = j_{rand}$ then
                $u_{i,j}$ is generated by operator $a$ selected by the AOS;
            else
                $u_{i,j} = x_{i,j}$;
            end
        end
    end
    for $i \leftarrow 1$ to $NP$ do
        Evaluate the fitness value of target vector $u_i$;
        if $f(u_i)$ is better than or equal to $f(x_i)$ then
            Calculate $\xi$ using equation (13);
            $\xi \rightarrow R_a$;
            $CR_i^a \rightarrow S_{CR}^a$, $F_i^a \rightarrow S_F^a$;
            Replace $x_i$ with $u_i$;
        else
            $\xi = 0$;
        end
        Update the $\mu_{CR}^a$ and $\mu_F^a$ for each operator $a$;
        Calculate the reward $r_a(g)$ for each operator $a$;
        Update the quality $q_a(g)$ for each operator $a$;
        Update the probability $p_a(g)$ for each operator $a$ by PM technique;
        $g = g + 1$;
    end
end
```

where v is the original value, v^* is the scaled value, max and min is the upper and lower bounds of the feature value, respectively.

(2) Train ELM classifier: ELM classifier is trained by the training set with the specific network parameters, i.e. the input weights and hidden biases, in K-fold

cross validation manner.

(3) Evolution operation: In this step, the system searches for better solutions by the proposed AdapDE, with the selected mutation operator and control parameters.

(4) Convert genotype to phenotype: This step converts the network parameters encoded in each chromosome to the corresponding variable values.

(5) Fitness evaluation: As discussed in Section 3.3, the cross validation accuracy and the norm of the output weights are combined to construct the fitness function as shown in equation (12).

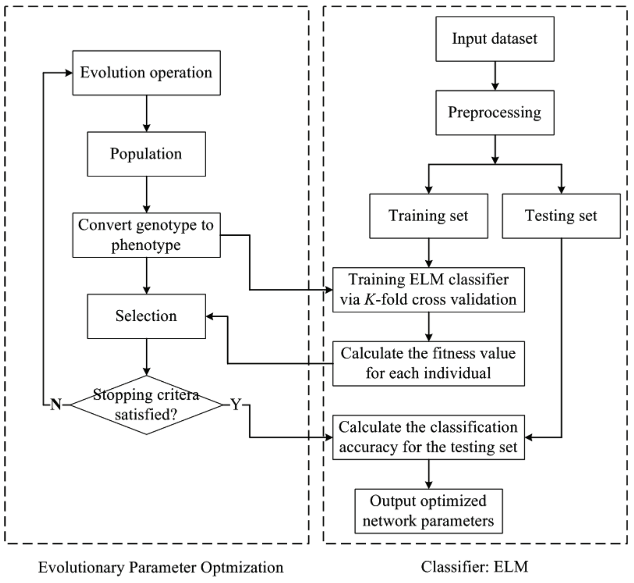

Evolution operation
Population
Convert genotype to phenotype
Selection
Stopping critera satisfied?
N
Y
Input dataset
Preprocessing
Training set
Testing set
Training ELM classifier via K-fold cross validation
Calculate the fitness value for each individual
Calculate the classification accuracy for the testing set
Output optimized network parameters
Evolutionary Parameter Optmization
Classifier: ELM

Fig. 3: System architecture of the proposed evolving ELM paradigm

4. Performance verification

In this section, the performance of our proposed evolving ELM paradigm is compared with the original ELM, as well as several other recent classification methods, including evolutionary ELM (E-ELM) [39], voting based ELM (V-ELM) [2], ensemble based ELM (EN-ELM) [16], back-propagation algorithm (BP) [19] and the radial basis function NN (RBFNN) [8]. The general experimental setup is given in Section 4.1 at first. Then, the performance comparisons are shown and discussed in Section 4.2 and Section 4.3. The benefits brought by the AdapDE and its underlying mechanism are further analyzed in Section 4.4.

4.1. *Experimental setup*

Simulations are conducted on many real-world classification data sets obtained from the UCI machine learning repository[a], including 3 binary data sets and 11 multiclass data sets. The numbers of classes, instances, and features are all given in Table 1. These data sets have been widely used as benchmarks to compare the performances of different classification methods. In our experiments, all the algorithms are implemented on the basis of the training sets and finally measured by their testing accuracies which are evaluated as the rates of the correctly classified testing data.

Table 1: Specification of the classification problems

Data sets	# classes	# training samples	# testing samples	# features
SPECTF	2	178	45	44
Diabetes	2	576	192	8
Breast cancer	2	379	190	30
Iris	3	100	50	4
Wine	3	118	60	13
Yeast	10	989	495	8
Vowel	11	528	462	10
Soybean	19	307	376	35
Segment	7	1500	810	19
Car	4	1152	576	6
Optdigits	10	3823	1797	64
Pen	10	7494	3498	16
Satellite	6	4435	2000	36
Letter	26	15000	5000	16

The related parameters of the comparative methods are generally set as follows. As for the proposed evolving ELM paradigm, the population size and the maximum number of generations are all set to be 100; μ^a_{CR} and μ^a_F are all initialized to be 0.5 for each operator a; the minimum operator selection probability $p_{min} = 0.05$ and the adaptation rate $\alpha = 0.3$, as recommended in [27]; the penalty rate to the cross validation accuracy is $\alpha_w = 0.01$. As E-ELM is also based on DE algorithm, for the sake of peer comparison, the population size and maximum number of generations are set as the same as our proposed algorithm; while the control parameter of DE in E-ELM, i.e. CR and F, are set constantly to be 0.8 and 1.0, respectively, according to the suggestions in [39]. For the sake of peer comparison, the number of

[a]http://archive.ics.uci.edu/ml/

independent ELMs for voting and constructing ensemble in V-ELM and EN-ELM are all set to be 7, according to the recommendation in [2]. For all the methods, the number of hidden neurons is set to be 50 and the activation functions are set as sigmoid here. Moreover, 30 independent runs are conducted with fixed size of SLFN on each data set, while the mean results and standard deviations are given in the corresponding tables. The best mean results are highlighted in bold face with grey background. The last row of each table presents the number of data sets on which the method is better than others. In order to have statistical sound conclusions, the nonparametric Wilcoxon's rank sum test at a 0.05 significance level is adopted to compare the statistical meaning of the difference between two competing algorithms. All these benchmark algorithms are developed by MATLAB 7.8. The platform is Intel P8500 2.5GHz CPU, 3G RAM with Windows 7 operating system.

4.2. *Comparison with ELM*

In this subsection, the performance of our proposed evolving ELM paradigm, i.e. Evo-ELM, is compared with the original ELM on the given real-world data sets. From the comparative results given in Table 2, we can easily find that the proposed Evo-ELM completely outperforms the original ELM on all the 14 data sets. These better performances are all with statistical significance according to the Wilcoxon's rank sum test. It is also worth noting that the improvements of the testing accuracies are remarkable on several data sets. For example, the improvements of the testing accuracies on the data sets Vowel, Optdigits, Satellite and Letter are more than 10%. In addition, the standard deviations of Evo-ELM are also lower than the original ELM on most of these 14 data sets, except for the data sets SPECTF, Breast cancer and Segment. From these empirical results, we can fully conclude that the performance of the original ELM has been improved by our proposed Evo-ELM, which mainly considers optimizing the network parameters of ELM.

4.3. *Comparison with several ELM variants, BP and RBFNN*

After the performance comparison with the original ELM, in this subsection, our proposed Evo-ELM is further compared with several other recently proposed ELM variants. In addition, two classical NNs, i.e. BP and RBFNN, are also considered here for comparison. From the comparative results given in Table 3, we can find that Evo-ELM is the best choice in most of the cases, as it shows significantly better performances on 9 data sets out 14. Specifically, similar to our proposed algorithm, E-ELM also considers finding the optimal network parameters by an EA, which is the basic DE algorithm in that case. However, different from the traditional DE algorithm, in which only a single mutation operator with static parameter settings is used for offspring generation, our proposed AdapDE online selects the appropriate operator and adjusts the control parameters in an adaptive way. From the results shown in the second and last column of Table 3, it is clear that our proposed Evo-ELM shows significantly better performance than E-ELM on 12 data sets out 14,

Table 2: Performance comparisons with ELM

Data sets	ELM	Evo-ELM
SPECTF	0.6978(9.43E-4)†	**0.7585(2.61E-3)**
Diabetes	0.7325(1.67E-3)†	**0.7660(1.28E-3)**
Breast cancer	0.9484(1.19E-4)†	**0.9599(2.69E-4)**
Iris	0.9273(1.31E-3)†	**0.9633(1.03E-3)**
Wine	0.9750(6.04E-4)†	**0.9917(2.18E-4)**
Yeast	0.5888(4.59E-4)†	**0.6043(4.54E-4)**
Vowel	0.4457(9.11E-4)†	**0.7733(8.71E-4)**
Soybean	0.8908(6.23E-4)†	**0.9421(2.39E-4)**
Segment	0.9165(9.29E-5)†	**0.9329(1.98E-4)**
Car	0.8684(3.53E-4)†	**0.9051(1.60E-4)**
Optdigits	0.8261(4.99E-4)†	**0.9295(7.46E-5)**
Pen	0.8240(5.53E-4)†	**0.9151(1.67E-4)**
Satellite	0.5862(4.45E-3)†	**0.7464(1.92E-4)**
Letter	0.5039(6.58E-4)†	**0.7146(9.02E-5)**
Statistics	0/14	14/14

† and ‡ denote the corresponding algorithm is significantly worse than and better than that of the proposed Evo-ELM, respectively, according to the Wilcoxon's rank sum test at a 0.05 significance level.

while E-ELM only significantly outperforms our proposed algorithm on the data set Segment. V-ELM and EN-ELM are recently proposed ELM variants, both of which perform multiple independent ELM training instead of single ELM training, the performance of EN-ELM is slightly inferior to that of V-ELM. Nevertheless, both of them are outperformed by Evo-ELM on most of the cases, while V-ELM only wins on the data set Breast cancer. BP and RBFNN are two classical NNs whose underlying mechanisms are different from that of ELM. The comparative results of them are given in the fifth and sixth columns of Table 3, respectively. Generally speaking, our proposed Evo-ELM still outperforms them on most of the data sets, except for Car and Satellite, on which the results of BP are better than that of Evo-ELM, while on the data sets Breast cancer and Optdigits, RBFNN outperforms Evo-ELM with statistical meaning.

4.4. *Comprehensive study on the underlying mechanism of the proposed AdapDE*

From the empirical studies on the performance comparisons, it is clear that our proposed Evo-ELM not only gives sufficient improvement to its baseline ELM, but also outperforms several recently proposed ELM variants. The superior performance of Evo-ELM probably benefits from the AdapDE, which can optimize the network parameters of the baseline ELM. Different from the traditional DE

Table 3: Performance comparisons with ELM variants, BP and RBFNN

Data sets	E-ELM	V-ELM	EN-ELM	BP	RBFNN	Evo-ELM
SPECTF	0.7150(1.52E-3)†	0.7217(3.02E-4)†	0.7217(1.12E-3)†	0.7502(5.54E-4)†	0.7440(4.91E-4)†	**0.7585(2.61E-3)**
Diabetes	0.7203(3.31E-4)†	0.7342(7.01E-5)†	0.7328(2.25E-4)†	0.7240(4.04E-4)†	0.7366(6.76E-5)†	**0.7660(1.28E-3)**
Breast cancer	0.9470(1.70E-4)†	0.9607(5.11E-5)	0.9561(7.90E-5)	0.9570(2.50E-5)	**0.9654(2.84E-5)**‡	0.9599(2.69E-4)
Iris	0.9227(1.70E-3)†	0.9240(1.00E-3)†	0.8933(1.31E-3)†	0.9307(4.49E-3)†	0.9616(4.71E-4)	**0.9633(1.03E-3)**
Wine	0.9500(6.90E-4)†	0.9700(2.38E-4)†	0.9517(5.24E-4)†	0.9778(1.60E-4)†	0.9577(1.16E-4)†	**0.9917(2.18E-4)**
Yeast	0.5838(3.15E-4)†	0.6018(5.79E-5)	0.5945(5.52E-5)	0.5012(1.34E-2)†	0.6002(9.57E-5)	**0.6043(4.54E-4)**
Vowel	0.3961(9.29E-4)†	0.4973(3.48E-4)†	0.4612(1.20E-3)†	0.4837(1.17E-2)†	0.5089(9.05E-4)†	**0.7733(8.71E-4)**
Soybean	0.8895(5.14E-4)†	0.9206(5.40E-5)†	0.8949(4.40E-4)†	0.6519(7.64E-2)†	0.8570(7.58E-4)†	**0.9421(2.39E-4)**
Segment	**0.9383(6.89E-5)**‡	0.9291(1.79E-5)	0.9288(4.90E-5)	0.8865(4.29E-2)†	0.9141(7.46E-5)†	0.9329(1.98E-4)
Car	0.9054(2.46E-4)	0.8646(7.23E-5)†	0.8559(9.48E-5)†	**0.9477(4.52E-3)**‡	0.8064(2.65E-4)†	0.9051(1.60E-4)
Optdigits	0.8664(1.40E-4)†	0.9031(6.73E-5)†	0.8300(8.12E-4)†	0.8137(6.76E-2)†	**0.9452(3.14E-5)**‡	0.9295(7.46E-5)
Pen	0.8639(3.03E-4)†	0.8451(8.00E-5)†	0.8065(4.83E-4)†	0.7284(7.93E-2)†	0.8882(1.01E-4)†	**0.9151(1.67E-4)**
Satellite	0.7389(1.47E-4)†	0.6425(1.17E-5)†	0.6445(1.64E-5)†	**0.8242(6.82E-3)**‡	0.6365(7.32E-5)†	0.7464(1.92E-4)
Letter	0.6013(2.10E-4)†	0.6095(1.31E-4)†	0.5359(1.02E-3)†	0.6525(4.89E-2)†	0.3510(9.44E-5)†	**0.7146(9.02E-5)**
Statistics	1/14	0/14	0/14	2/14	2/14	9/14

† and ‡ denote the corresponding algorithm is significantly worse than and better than that of the proposed Evo-ELM, respectively, according to the Wilcoxon's rank sum test at a 0.05 significance level.

algorithms, which only use single mutation operator with static parameter settings, AdapDE maintains an operator pool and selects the appropriate operator online, while the corresponding control parameters are also tuned adaptively. In order to have a better knowledge about the proposed AdapDE, its underlying mechanism is comprehensively studied in the subsequent subsections step by step.

4.4.1. *Performance comparison with the static variants*

As introduced in Section 3.6, the operator pool consists of four different DE mutation operators, each of which owns its distinctive and diverse search behavior. People might have the following two questions:

(1) Is there any mutation operator that is capable of efficiently tackling the whole set of benchmark problems?
(2) If yes, would the baseline ELM with the best mutation operator be able to outperform the proposed Evo-ELM?

In order to answer these questions, the individual performances of each of the four mutation operators considered by the AOS method with static parameter settings (the so called "static variant") are empirically analyzed in this subsection. The control parameters CR and F are set constantly as 0.8 and 1.0, respectively, as in E-ELM. For the sake of clearness, the static variant with DE mutation operator a will be shortly denoted as "Operator a", without loss of generality.

From the comparative results given in Table 4, it becomes clear that no single mutation operator is able to perform best over all different kinds of data sets. This empirical finding is consistent with the *No Free Lunch* (NFL) theorem [33] which briefly states that no algorithm can perform best over all problems. It not only answers the first question, but also greatly motivates for the use of AOS methods. And indeed, Evo-ELM keeps on being the best option in most of the cases, with significantly better performances on 9 data sets out of 14. It becomes clear, then, that the baseline ELM with the incorporation of AdapDE, which can automatically select the appropriate operator and control the parameter settings, is significantly more efficient than the DE algorithm with single mutation operator, even if the most suitable operator is applied. It is also worth noting that the best static variant for each data set outperforms E-ELM. This might attribute to the redesigned fitness function, which uses 5-fold cross validation, so that the generalization performance of the baseline ELM can be improved to some extent.

4.4.2. *Empirical studies on the adaptive module*

There are two major adaptive modules in AdapDE, which schedule the operator selection and parameter control, respectively. In this case, two more questions are still remained for further understanding the underlying mechanisms of the proposed AdapDE.

Table 4: Performance comparisons with static variants

Data sets	Operator 1	Operator 2	Operator 3	Operator 4	Evo-ELM
SPECTF	0.7443(3.11E-3)†	0.7505(2.02E-3)†	0.7452(2.71E-3)†	**0.7630(2.47E-3)**‡	0.7585(2.61E-3)
Diabetes	0.7551(2.31E-4)†	0.7593(2.27E-4)†	0.7603(1.11E-4)†	0.7573(3.15E-4)†	**0.7660(1.28E-3)**
Breast cancer	0.9475(1.78E-4)†	**0.9611(2.19E-4)**‡	0.9317(3.24E-4)†	0.9281(2.28E-4)†	0.9599(2.69E-4)
Iris	0.9279(1.53E-3)†	0.9315(1.77E-3)†	0.9522(1.01E-3)†	0.9518(1.32E-3)†	**0.9633(1.03E-3)**
Wine	0.9742(4.72E-4)†	0.9698(3.11E-4)†	0.9821(3.06E-4)†	0.9471(3.57E-4)†	**0.9917(2.18E-4)**
Yeast	0.5898(3.72E-4)†	**0.6174(5.21E-4)**‡	0.5872(4.33E-4)†	0.5806(4.10E-4)†	0.6043(4.54E-4)
Vowel	0.7625(7.17E-4)†	0.7459(6.97E-4)†	0.7228(8.74E-4)†	0.7207(8.12E-4)†	**0.7733(8.71E-4)**
Soybean	0.9389(3.19E-4)†	0.9401(2.52E-4)	0.9278(3.01E-4)†	0.9022(4.09E-4)†	**0.9421(2.39E-4)**
Segment	0.9237(1.05E-4)†	0.9230(1.69E-4)†	0.9315(2.11E-4)	0.9307(1.89E-4)	**0.9329(1.98E-4)**
Car	**0.9094(2.51E-4)**‡	0.8875(2.05E-4)†	0.8952(1.79E-4)†	0.8901(2.11E-4)†	0.9051(1.60E-4)
Optdigits	0.8841(4.29E-4)†	0.9014(3.78E-4)†	0.8856(3.99E-4)†	0.8763(3.63E-4)†	**0.9295(7.46E-5)**
Pen	0.8765(3.73E-4)†	0.8731(2.75E-4)†	0.8918(1.99E-4))†	0.8932(2.18E-4)†	**0.9151(1.67E-4)**
Satellite	0.7359(1.64E-4)†	0.7351(1.97E-4)†	**0.7471(2.01E-4)**	0.7122(2.35E-4)†	0.7464(1.92E-4)
Letter	0.6018(3.71E-4)†	0.6427(2.54E-4)†	0.5971(3.23E-4)†	0.5889(2.67E-4)†	**0.7146(9.02E-5)**
Statistics	1/14	2/14	1/14	1/14	9/14

† and ‡ denote the corresponding algorithm is significantly worse than and better than that of the proposed Evo-ELM, respectively, according to the Wilcoxon's rank sum test at a 0.05 significance level.

(1) How about the performance of AdapDE with only AOS module, while the control parameters are set constantly?
(2) How about the performance of AdapDE with only adaptive parameter control, while the operators are selected randomly?

The performance of Evo-ELM with its "semi-adaptive" variants, which respectively use the AOS and the adaptive parameter control only, are given in Table 5. From these comparative results, it is not clear which of the adaptive modules is the most beneficial one to the performance of AdapDE: on some data sets, the "AOS only" method (denoted as AOS) is better than the "parameter control only" one (denoted as Uniform), while on others the opposite case occurs. In particular, the prior variant wins on 4 data sets out of 14, but only 2 of them are statistically meaningful; while the latter one significantly outperforms Evo-ELM on 2 data sets out of 14. Nevertheless, these results clearly demonstrate that the combined use of both adaptive modules is better than their sole use, which is shown by the fact that Evo-ELM significantly outperforms both of them on 7 data sets out of 14.

Table 5: Performance comparisons with semi-adaptive variants

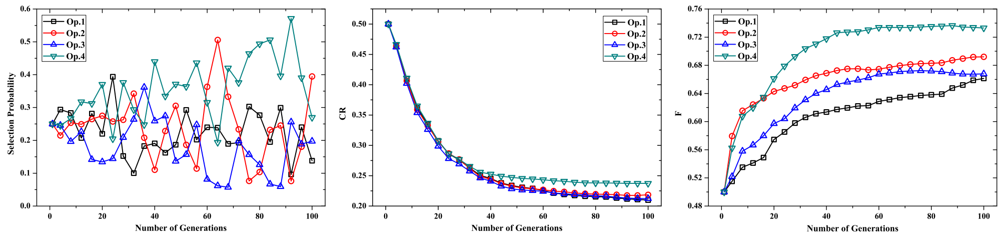
Data sets	Uniform	AOS	Evo-ELM
SPECTF	**0.7755(1.31E-3)**‡	0.7585(2.38E-3)	0.7585(2.61E-3)
Diabetes	0.7571(1.26E-3)†	**0.7710(8.99E-4)**‡	0.7660(1.28E-3)
Breast cancer	0.9509(2.49E-4)†	**0.9608(2.11E-4)**	0.9599(2.69E-4)
Iris	0.9267(1.30E-3)†	0.9389(2.38E-3)†	**0.9633(1.03E-3)**
Wine	**0.9939(1.28E-4)**	0.9826(4.34E-4)†	0.9917(2.18E-4)
Yeast	0.6017(4.28E-4)	**0.6093(4.15E-4)**‡	0.6043(4.54E-4)
Vowel	0.7600(8.61E-4)†	0.7224(8.71E-4)†	**0.7733(8.71E-4)**
Soybean	0.9311(2.11E-4)†	0.9224(2.42E-4)†	**0.9421(2.39E-4)**
Segment	0.9215(2.13E-4)	0.9093(3.14E-4)†	**0.9329(1.98E-4)**
Car	**0.9107(1.93E-4)**‡	0.8709(2.30E-4)†	0.9051(1.60E-4)
Optdigits	0.9009(1.43E-4)†	0.8976(1.19E-4)†	**0.9295(7.46E-5)**
Pen	0.8735(4.34E-4)†	0.8885(2.75E-4)†	**0.9151(1.67E-4)**
Satellite	0.7358(1.35E-4)†	**0.7477(1.85E-4)**	0.7464(1.92E-4)
Letter	0.6757(1.50E-3)†	0.6202(1.33E-3)†	**0.7146(9.02E-5)**
Statistics	3/14	4/14	7/14

† and ‡ denote the corresponding algorithm is significantly worse than and better than that of the proposed Evo-ELM, respectively, according to the Wilcoxon's rank sum test at a 0.05 significance level.

4.5. *Analysis of adaptation property of AdapDE*

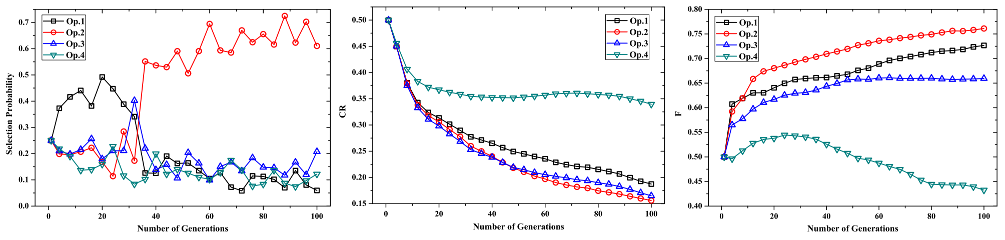
After the previous comprehensive studies on the performance of the proposed evolving ELM paradigm and the effectiveness of the AdapDE, two additional issues are still remained for further understanding the adaptive properties of AdapDE.

(1) How does the AOS module select the appropriate operators?
(2) How does the parameter control module adjust the corresponding parameters adaptively?



(a) Selection Probabilities (b) CR (c) F

Fig. 4: Adaptation characteristics of the operator selection probabitlities, mean values of CR and F on SPECFT



(a) Selection Probabilities (b) CR (c) F

Fig. 5: Adaptation characteristics of the operator selection probabitlities, mean values of CR and F on Letter

Generally speaking, when DE solves optimization problems, there are no specific values for the control parameters CR and F, which can be constantly effective during the entire optimization process. Instead, it makes more sense that, in different search processes, the effective combinations of CR and F values will be different. As the CR and F values for different operators are generated from normal and cauthy distributions, independently, here we just plot their mean values, i.e. μ_{CR}^a and μ_F^a for each operator a, when Evo-ELM is working on the data sets SPECFT and Letter in Figure 4 and Figure 5, respectively (it is worth noting that the other data sets bear the similar trends). As shown in the corresponding figures, we can clearly find that the μ_{CR}^a value for each operator a keeps decreasing during the early stages of evolution, and then converge to some specific values. This makes sense, since a low CR value means the offspring inherents more information from its parent, as the optimal solution or area has been located in the later stages of the evolution. Oppositely, μ_F^a value for each operator a keeps increasing during the early generations before converging to some degree.

In order to investigate the adaptation characteristics of the operator selection process, we also plot the evolution progresses of these four mutation operators' selection probabilities on data sets SPECFT and Letter in Figure 4(a) and Figure 5(a), respectively. As shown in Figure 4(a), the selection probability for each opeartor varies a lot during the entire optimization process. This can be explained by that those four mutation operators perform similarly when tackling this problem. It also confirms the observation from Table 4, in which the comparative results of those four static variants do not differ a lot. However, we are still able to find that the AOS module gives more chances for Operator 4, which is the best static variant in Table 4 when considering the data set SPECFT. As for Figure 5(a), the selection probabilities vary a lot during the early 30 generations before Operator 2 dominates the selection probabity. This also confirms the comparative results given in Table 4 where the static variant with Operator 2 performs much better than the others.

5. Conclusion

In this paper, an evolving ELM paradigm that integrates ELM and DE is presented for classification problems. Specifically, in the proposed evolving ELM paradigm, a DE algorithm (AdapDE) with adaptive operator selection and parameter control is proposed to optimize the network parameters of the baseline ELM. AdapDE is able to online select the appropriate operator for offspring generation, while the corresponding control parameters can be adjusted in an adaptive manner. In order to improve the generalization performance of the baseline ELM, 5-fold cross validation is employed as the fitness function. The optimization target is thus to maximize this fitness value in this case. From the empirical studies on several real-world classification data sets, it is clear that not only the performance of the baseline ELM is improved, but also the proposed algorithm can outperform several recent classification methods.

In future, the proposed evolving ELM paradigm can be further extended to the areas of ensemble learning and ensemble evolution with ELM. In addition, more advanced adaptive operator selection mechanism, such as multi-armed bandit which is proposed in [7], can be considered in AdapDE.

References

1. Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1996.
2. Jiuwen Cao, Zhiping Lin, Guang-Bin Huang, and Nan Liu. Voting based extreme learning machine. *Information Sciences*, 185(1):66–77, 2012.
3. Binu Chacko, V. Vimal Krishnan, G. Raju, and P. Babu Anto. Handwritten character recognition using wavelet energy and extreme learning machine. *International Journal of Machine Learning and Cybernetics*, 3:149–161, 2012.
4. S. Das and P.N. Suganthan. Differential Evolution: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, Feb. 2011.
5. Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1):29–41, 1996.
6. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin, 2005.
7. Álvaro Fialho. *Adaptive Operator Selection for Optimization*. PhD thesis, Université Paris-Sud XI, Orsay, France, December 2010.
8. P.E. Hart and D.G. Stork. *Pattern classification*. 2001.
9. S. He, Q.H. Wu, and J.R. Saunders. Group search optimizer: An optimization algorithm inspired by animal searching behavior. *Evolutionary Computation, IEEE Transactions on*, 13(5):973–990, Oct. 2009.
10. Guang-Bin Huang, DianHui Wang, and Yuan Lan. Extreme learning machines: a survey. *International Journal of Machine Learning and Cybernetics*, 2:107–122, 2011.
11. Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforwad neural networks. In *IJCNN'04: Proc. of the 2004 International Joint Conference on Neural Network*, pages 25–29, July 2004.
12. Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70:489–501, 2006.
13. Sk. Minhazul Islam, Swagatam Das, Saurav Ghosh, Subhrajit Roy, and Ponnuthurai Nagaratnam Suganthan. An Adaptive Differential Evolution Algorithm With Novel Mutation and Crossover Strategies for Global Numerical Optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 42(2):482–500, 2012.
14. James Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948. IEEE Service Center, 1995.
15. Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI'95: Proc. of the 14th International Joint Conference on Artificial Intelligence*, pages 1137–1143. Morgan Kaufmann, 1995.
16. Nan Liu and Han Wang. Ensemble based extreme learning machine. *IEEE Signal Processing Letters*, 17(8):754–757, Aug. 2010.
17. Yoan Miche, Antti Sorjamaa, Patrick Bas, Olli Simula, Christian Jutten, and Amaury Lendasse. OP-ELM: Optimally Pruned Extreme Learning Machine. *IEEE Transactions on Neural Networks*, 21(1):158–162, 2010.
18. Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.
19. M.F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.
20. Yew-Soon Ong, Meng-Hiot Lim, Ning Zhu, and Kok Wai Wong. Classification of adaptive memetic algorithms: a comparative study. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(1):141–152, 2006.
21. Kenneth Price, Rainer Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer-Verlag, Berlin, Germany, 2005.

22. A.K. Qin, V.L. Huang, and P.N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 13(2):398–417, Apr. 2009.
23. Saras Saraswathi, Suresh Sundaram, Narasimhan Sundararajan, Michael Zimmermann, and Marit Nilsen-Hamilton. Icga-pso-elm approach for accurate multiclass cancer classification resulting in reduced gene sets in which genes encoding secreted proteins are highly represented. *IEEE/ACM Transactions Computational Biology and Bioinformatics*, 8(2):452–463, 2011.
24. Peter Sarlin. Visual tracking of the millennium development goals with a fuzzified self-organizing neural network. *International Journal of Machine Learning and Cybernetics*, 3:233–245, 2012.
25. Danielle N. G. Silva, Luciano D. S. Pacifico, and Teresa Bernarda Ludermir. An evolutionary extreme learning machine based on group search optimization. In *CEC'11: Proc. of the 2011 IEEE Congress on Evolutionary Computation*, pages 574–580, 2011.
26. Rainer Storn and Kenneth Price. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11:341–359, 1997.
27. Dirk Thierens. An adaptive pursuit strategy for allocating operator probabilities. In *GECCO'05: Proc. of the 2005 conference on Genetic and evolutionary computation*, pages 1539–1546, New York, NY, USA, 2005. ACM.
28. Eric C.C. Tsang, Xi-Zhao Wang, and Daniel S. Yeung. Improving learning accuracy of fuzzy decision trees by hybrid neural networks. *IEEE Transactions on Fuzzy Systems*, 8(5):601–614, Oct. 2000.
29. Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.
30. Xi-Zhao Wang and Jiarong Hong. Learning optimization in simplifying fuzzy rules. *Fuzzy Sets and Systems*, 106(3):349–356, 1999.
31. Xi-Zhao. Wang, Daniel S. Yeung, and Eric C.C. Tsang. A comparative study on heuristic algorithms for generating fuzzy decision trees. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 31(2):215–226, apr 2001.
32. Yong Wang, Zixing Cai, and Qingfu Zhang. Differential evolution with composite trial vector generation strategies and control parameters. *IEEE Transactions on Evolutionary Computation*, 15(1):55–66, Feb. 2011.
33. David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.
34. You Xu and Yang Shu. Evolutionary extreme learning machine - based on particle swarm optimization. In *ISNN (1)*, pages 644–652, 2006.
35. Xin Yao. Evolving Artificial Neural Networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.
36. Daniel S. Yeung, Wing W.Y. Ng, Defeng Wang, Eric C.C. Tsang, and Xi-Zhao Wang. Localized generalization error model and its application to architecture selection for radial basis function neural network. *IEEE Transactions on Neural Networks*, 18(5):1294–1305, Sep. 2007.
37. Jingqiao Zhang and Arthur C. Sanderson. JADE: Adaptive Differential Evolution With Optional External Archive. *IEEE Transactions on Evolutionary Computation*, 13(5):945–958, 2009.
38. Mingyuan Zhao, Chong Fu, Luping Ji, Ke Tang, and Mingtian Zhou. Feature selection and parameter optimization for support vector machines: A new approach based on genetic algorithm with feature chromosomes. *Expert Systems with Applications*, 38(5):5197 – 5204, 2011.
39. Qin-Yu Zhu, A. Kai Qin, Ponnuthurai N. Suganthan, and Guang-Bin Huang. Evolu-

tionary extreme learning machine. *Pattern Recognition*, 38(10):1759–1763, 2005.