Neural Process Lett (2012) 36:285–305
DOI 10.1007/s11063-012-9236-y

Self-Adaptive Evolutionary Extreme Learning Machine

Jiuwen Cao · Zhiping Lin · Guang-Bin Huang

Published online: 18 July 2012

Abstract In this paper, we propose an improved learning algorithm named self-adaptive evolutionary extreme learning machine (SaE-ELM) for single hidden layer feedforward networks (SLFNs). In SaE-ELM, the network hidden node parameters are optimized by the self-adaptive differential evolution algorithm, whose trial vector generation strategies and their associated control parameters are self-adapted in a strategy pool by learning from their previous experiences in generating promising solutions, and the network output weights are calculated using the Moore–Penrose generalized inverse. SaE-ELM outperforms the evolutionary extreme learning machine (E-ELM) and the different evolutionary Levenberg–Marquardt method in general as it could self-adaptively determine the suitable control parameters and generation strategies involved in DE. Simulations have shown that SaE-ELM not only performs better than E-ELM with several manually choosing generation strategies and control parameters but also obtains better generalization performances than several related methods.

Keywords Differential evolution · Extreme learning machine · Single hidden layer feedforward networks · Levenberg–Marquardt algorithm · Support vector machine

1 Introduction

Machine learning has been developed and used in industry in the past several decades [1–8]. Among these methods, algorithms and applications on artificial neural networks [1–4, 10–17, 19–21, 24] and support vector machine (SVM) [7–9] have been widely researched.

J. Cao (✉) · Z. Lin · G.-B. Huang
School of Electrical and Electronic Engineering, Nanyang Technological University,
Singapore 639798, Singapore
e-mail: caoj0003@e.ntu.edu.sg

Z. Lin
e-mail: ezplin@ntu.edu.sg

G.-B. Huang
e-mail: egbhuang@ntu.edu.sg

Recently, a new method called extreme learning machine (ELM) [1,2] was developed for SLFNs and was popular for its fast training speed by means of utilizing random hidden node parameters and calculating the output weights with least square algorithm [13–17]. These features enable ELM to overcome several limitations existed in gradient-descent based algorithms, such as stuck in the local minima and slow convergence performance. Since in ELM, the number of hidden nodes is assigned a priori, the hidden node parameters are randomly chosen and they remain unchanged during the training phase. Many non-optimal nodes may exist and contribute less in minimizing the cost function. Moreover, Huang et al. also pointed out in [2] that ELM tends to require more hidden nodes than conventional tuning-based algorithms [10,11] in many cases.

During past several years, evolutionary algorithms (EA) are widely used as a global searching method for optimizing the neural network parameters. Among all these evolutionary algorithms, differential evolution (DE) [18] which is a simple but powerful population-based stochastic direct searching technique is a frequently used method for selecting the network parameters [19–21]. In [19], DE is directly adopted as a training algorithm for feedforward networks where all the network parameters are encoded into one population vector and the error function between the network approximate output and the expected output is used as the fitness function to evaluate all the populations. However, Subudhi and Jena [20] have pointed out that using the DE approach alone for the network training may yield a slow convergence speed. Therefore, they developed an improved method by combining the DE approach and the Levenberg–Marquardt (LM) method (DE-LM) for training a feedforward network. In DE-LM, DE is first utilized to train the network and to choose suitable candidates of the network parameters, and then the LM method is adopted to find the optimal ones. The role of DE in DE-LM is to move towards the global minimum and then LM is used to tune better the candidate parameters obtained by DE and move forward to the minimum. In [21], a new algorithm named evolutionary extreme learning machine (E-ELM) based on DE and ELM has been developed for SLFNs. Using the DE method to optimize the network input parameters and the ELM algorithm to calculate the network output weights, E-ELM has shown several promising features. It not only ensures a more compact network size than ELM, but also has better generalization performance than several related algorithms in general, such as DE-LM, ELM and LM.

However, in all the above DE based neural network training algorithms, the trial vector generation strategies and the control parameters in DE have to be manually chosen. For example, the control parameters in E-ELM are manually selected according to an empirical suggestion and the simple random generation method is adopted to produce the trial vector. As pointed out by many researchers, the performance of the DE algorithm highly depends on the chosen trial vector generation strategy and the control parameters, and inappropriate choices of strategies and control parameters may result in premature convergence or stagnation. Therefore, for different applications, fixing the trial vector generation strategy and the control parameters in the training of SLFNs may also lead to different network generalization performances.

In this paper, we propose a novel learning algorithm named self-adaptive evolutionary extreme learning machine (in short, as SaE-ELM) for SLFNs. In SaE-ELM, the hidden node learning parameters are optimized by the self-adaptive differential evolution algorithm [22], whose trial vector generation strategies and their associated control parameters are self-adapted in a strategy pool by learning from their previous experiences in generating promising solutions, and the network output weights are calculated using the Moore–Penrose (MP) generalized inverse. Simulations on regression and classification problems are carried out to show the effectiveness of the proposed algorithm. To show the efficiency of the proposed algorithm, we compare our method with several related algorithms, including

E-ELM [21], DE-LM [20], SVM [9], LM [2] and wavelet neural networks with ELM [24]. For E-ELM and DE-LM, several manually combinations of trial vector generation strategies and control parameters are used for simulations where eight mutation strategies are utilized to generate new mutant candidates and the control parameters are selected followed studies in [22,25,26]. Simulation results show that our method outperforms all these related algorithms in general.

2 Preliminaries

In this section, we give brief reviews of ELM and DE.

2.1 Extreme Learning Machine

For N arbitrary training samples $\{(\boldsymbol{x}_i, \boldsymbol{t}_i)\}_{i=1}^{N}$, where $\boldsymbol{x}_i \in \mathbf{R}^d$ and $\boldsymbol{t}_i \in \mathbf{R}^m$, the output of a SLFN with L hidden nodes is

$$\boldsymbol{o}_i = \sum_{j=1}^{L} \boldsymbol{\beta}_j g(\boldsymbol{a}_j, b_j, \boldsymbol{x}_i), \quad i = 1, 2, \cdots, N \tag{1}$$

where $\boldsymbol{a}_j \in \mathbf{R}^d$ and $b_j \in \mathbf{R}$ $(j = 1, 2, \cdots, L)$ are learning parameters of the jth hidden node, respectively. $\boldsymbol{\beta}_j \in \mathbf{R}^m$ is the link connecting the jth hidden node to the output node. $g(\boldsymbol{a}_j, b_j, \boldsymbol{x}_i)$ is the output of the jth hidden node with respect to the input sample $\boldsymbol{x}_i$.

To minimize the cost function $E = \sum\limits_{i=1}^{N} \|\boldsymbol{o}_i - \boldsymbol{t}_i\|$, the traditional gradient based algorithm is to iteratively update the network parameter W according to the following equation

$$W_{k+1} = W_k - \eta \frac{\partial E\,(W_k)}{\partial W_k} \tag{2}$$

where W is a set of $\left(\boldsymbol{a}_j, b_j, \boldsymbol{\beta}_j\right)$ and η is the learning rate. Although this algorithm works well in many applications, it still has several issues, such as local minimal, overfitting, slow convergence rate, etc.

Different to traditional gradient descend algorithm, ELM states that SLFNs with L hidden nodes each with activation function $g\,(\cdot)$ can approximate these N samples with zero error means that $\sum\limits_{i=1}^{N} \|\boldsymbol{o}_i - \boldsymbol{t}_i\| = 0$, i.e. there exist $\boldsymbol{\beta}_j$, $\boldsymbol{a}_j$ and b_j such that

$$\mathrm{H}\boldsymbol{\beta} = \mathrm{T} \tag{3}$$

where $\mathrm{H} = \left\{h_{ij}\right\}$ is the hidden-layer output matrix and $h_{ij} = g\left(\boldsymbol{a}_j, b_j, \boldsymbol{x}_i\right)$. $\boldsymbol{\beta} = \left(\boldsymbol{\beta}_1\ \boldsymbol{\beta}_2 \cdots \boldsymbol{\beta}_L\right)$ is the output weight matrix and $\mathrm{T} = \left(\boldsymbol{t}_1\ \boldsymbol{t}_2 \cdots \boldsymbol{t}_N\right)$ is the target output.

By simply randomly choosing hidden nodes and then adjusting the output weights, single hidden layer feedforward networks (SLFNs) work as universal approximators with any bounded non-linear piecewise continuous functions for additive nodes [12]. ELM algorithm claims that the hidden node parameters can be randomly assigned [1,2], then the system equation becomes a linear model and the network output weights can be analytically determined by finding a least-square solution of this linear system as follow

$$\hat{\boldsymbol{\beta}} = \mathrm{H}^{\dagger}\mathrm{T} \tag{4}$$

where $\mathrm{H}^{\dagger}$ is the MP generalized inverse of H.

2.2 Differential Evolution

To minimize a target function

$$\min_{\boldsymbol{\theta} \in \mathbf{R}^D} f(\boldsymbol{\theta})$$

with respect to the parameter vector $\boldsymbol{\theta} \in \mathbf{R}^D$, the differential evolution algorithm aims at evolving a population of NP individual parameter vectors towards the global optimum [18]. At the generation G, the ith candidature parameter vector is as follows

$$\boldsymbol{\theta}_{i,G} = \left[\theta_{i,G}^1, \theta_{i,G}^2, \cdots, \theta_{i,G}^D\right] \quad i = 1, 2, \cdots, NP \tag{5}$$

The general procedures of DE are as follows.

Step 1. Initialization

A set of NP individual parameter vectors $\boldsymbol{\theta}_{i,G}$ are initialized to cover the parameter space as much as possible by the following equation

$$\boldsymbol{\theta}_{i,G} = \boldsymbol{\theta}_{\min} + \text{rand}(0, 1) \cdot (\boldsymbol{\theta}_{\max} - \boldsymbol{\theta}_{\min}) \tag{6}$$

where $\boldsymbol{\theta}_{\min} = \left[\theta_{\min}^1, \theta_{\min}^2, \cdots, \theta_{\min}^D\right]$ and $\boldsymbol{\theta}_{\max} = \left[\theta_{\max}^1, \theta_{\max}^2, \cdots, \theta_{\max}^D\right]$ are the prescribed minimum and maximum parameter bounds, respectively.

Step 2. Mutation

After the initialization, DE employs a self-organizing scheme that takes the difference vector of randomly chosen population vectors to perturb an existing vector to generate a new mutant vector. With respect to each individual parameter vector $\boldsymbol{\theta}_{i,G}$ at current generation, a mutant vector $\boldsymbol{v}_{i,G}$ is generated via a certain mutation strategy. Several mutation strategies have been suggested in [18] for different problems. Here, we list four frequently used mutation strategies (in short, as St) as follows:

St 1: DE/rand/1

$$\boldsymbol{v}_{i,G} = \boldsymbol{\theta}_{r_1^i,G} + F \cdot \left(\boldsymbol{\theta}_{r_2^i,G} - \boldsymbol{\theta}_{r_3^i,G}\right)$$

St 2: DE/rand-to-best/2

$$\begin{aligned}\boldsymbol{v}_{i,G} = \boldsymbol{\theta}_{r_1^i,G} &+ F \cdot \left(\boldsymbol{\theta}_{\text{best},G} - \boldsymbol{\theta}_{r_1^i,G}\right) \\ &+ F \cdot \left(\boldsymbol{\theta}_{r_2^i,G} - \boldsymbol{\theta}_{r_3^i,G}\right) + F \cdot \left(\boldsymbol{\theta}_{r_4^i,G} - \boldsymbol{\theta}_{r_5^i,G}\right)\end{aligned}$$

St 3: DE/rand/2

$$\boldsymbol{v}_{i,G} = \boldsymbol{\theta}_{r_1^i,G} + F \cdot \left(\boldsymbol{\theta}_{r_2^i,G} - \boldsymbol{\theta}_{r_3^i,G}\right) + F \cdot \left(\boldsymbol{\theta}_{r_4^i,G} - \boldsymbol{\theta}_{r_5^i,G}\right)$$

St 4: DE/current-to-rand/1

$$\boldsymbol{v}_{i,G} = \boldsymbol{\theta}_{i,G} + K \cdot \left(\boldsymbol{\theta}_{r_1^i,G} - \boldsymbol{\theta}_{i,G}\right) + F \cdot \left(\boldsymbol{\theta}_{r_2^i,G} - \boldsymbol{\theta}_{r_3^i,G}\right).$$

In all these equations, the indices $r_1^i, r_2^i, r_3^i, r_4^i, r_5^i$ are mutually exclusive integers randomly generated within the range $[1, 2, \cdots, NP]$, which are also different from the index i. The positive amplification factor F is used to control the scaling of the difference vectors and is usually selected within the range $0 \leq F \leq 2$. The control parameter K is randomly generated within the region $0 \leq K \leq 1$. As evident by many researchers, different vector generation strategies usually perform differently when solving different optimization problems [18,22]. The "DE/rand/1" strategy is suitable for solving multimodal problems due to its stronger exploration capability. However, this strategy usually demonstrates slow convergence speed [18,22]. The "DE/rand-to-best/2" strategy relying on the best solution found so far converges rapidly and performs well when dealing with unimodal problems. However, when solving multimodal problems, this strategy is more likely to get stuck at a local optimum and lead to a premature convergence. Two-difference-vectors-based strategies, "DE/rand-to-best/2" and "DE/rand/2" could lead to a better perturbation than one-difference-vector-based strategies. But they also require a high computational cost. "DE/current-to-rand/1" is a rotation-invariant strategy. It has been shown that this strategy is efficient in solving multiobjective optimization problems [22].

Step 3. Crossover

After generating all the mutant vectors, a crossover procedure is used to increase the diversities of the perturbed parameter vectors. With respect to each mutant vector $\boldsymbol{v}_{i,G} = \left[v_{i,G}^1, v_{i,G}^2, \cdots, v_{i,G}^D\right]$ at generation G, a trial vector $\boldsymbol{u}_{i,G} = \left[u_{i,G}^1, u_{i,G}^2, \cdots, u_{i,G}^D\right]$ is created according to the following crossover equation

$$u_{i,G}^j = \begin{cases} v_{i,G}^j, & \text{if } \left(\text{rand}_j \leq CR\right) \quad \text{or} \quad (j = j_{\text{rand}}) \\ \theta_{i,G}^j, & \text{Otherwise} \end{cases} \tag{7}$$

where CR is the crossover rate to control the fraction of the parameter values copied from the mutant vector and is a positive value chosen in the region $0 \leq CR < 1$. rand_j is the jth evaluation of a uniform random number generator with outcome in [0, 1]. j_{rand} is a randomly chosen integer from $[1, D]$ and is introduced to ensure that there exist at least one parameter in $\boldsymbol{u}_{i,G}$ differing from the target vector $\boldsymbol{\theta}_{i,G}$.

Step 4. Selection

For each target vector and its corresponding trial vector, a selection step is conducted using a fitness function. The one with the lower value of the fitness function is kept as the population of the next generation.

Step 2 to Step 4 are repeated until the goal is met or the maximum iterations are reached.

3 SaE-ELM

We first discuss the two existing training algorithms for SLFNs based on DE, and then present the SaE-ELM algorithm in this section.

3.1 Differential Evolution for SLFNs

Differential evolution has been widely used for SLFNs parameter optimizations [19–21]. The two representative methods are DE-LM [20] and E-ELM [21], respectively. DE-LM adopts DE to optimize the network input and output weight vectors with several iteration steps and then the best candidature parameter vector with the lowest value of the cost function is used as the initial value of the LM algorithm. E-ELM utilizes the DE method to select the hidden node parameters and the ELM algorithm to analytically determine the output weights. As shown in [21], E-ELM is able to achieve a better generalization performance with a more compact network size than ELM and learn much faster than DE-LM.

Although these algorithms have shown good performance in regressions, system identifications and classifications, they still suffer from manually selecting the mutation strategy and control parameters in DE. For example, the control parameter F and the crossover rate CR in DE-LM and E-ELM are manually determined. The "DE/rand/1" mutation strategy is used for all experiments conducted in DE-LM [20] and E-ELM [21] to generate new mutant vectors. As discussed by many researchers that for different optimization problems, different vector generation strategies and control parameters may perform highly different. Hence, fixing the generation strategy and manual setting control parameters in E-ELM and DE-LM may not be the proper way for different applications.

3.2 SaE-ELM

To overcome the limitations of manually choosing trial vector generation strategies and its associate control parameters in DE-LM and E-ELM, we propose a self-adaptive evolutionary extreme learning machine (SaE-ELM) for SLFNs by incorporating the self-adaptive differential evolution algorithm [22] to optimize the network input weights and hidden node biases and the extreme learning machine to derive the network output weights.

Given a set of training data and L hidden nodes with an activation function $g(\cdot)$, we summarize the SaE-ELM algorithm in the following steps.

Step 1. Initialization

A set of NP vectors where each one includes all the network hidden node parameters are initialized as the populations of the first generation

$$\boldsymbol{\theta}_{k,G} = \left[\boldsymbol{a}_{1,(k,G)}^T, \cdots, \boldsymbol{a}_{L,(k,G)}^T, b_{1,(k,G)}, \cdots, b_{L,(k,G)}\right],$$

where $\boldsymbol{a}_j$ and b_j $(j = 1, \cdots, L)$ are randomly generated, G represents the generation and $k = 1, 2, \cdots, NP$.

Step 2. Calculations of output weights and RMSE

Calculate the network output weight matrix and root mean square error (RMSE) with respect to each population vector with the following equations, respectively

$$\boldsymbol{\beta}_{k,G} = \mathrm{H}_{k,G}^{\dagger}\mathrm{T} \tag{8}$$

$$\mathrm{RMSE}_{k,G} = \sqrt{\frac{\sum_{i=1}^{N}\left\|\sum_{j=1}^{L}\boldsymbol{\beta}_j g\left(\boldsymbol{a}_{j,(k,G)}, b_{j,(k,G)}, \boldsymbol{x}_i\right) - \boldsymbol{t}_i\right\|}{m \times N}} \tag{9}$$

where $\mathrm{H}_{k,G}$ is shown in equation (10) and $\mathrm{H}_{k,G}^{\dagger}$ is the MP generalized inverse of $\mathrm{H}_{k,G}$.

$$\mathrm{H}_{k,G} = \begin{bmatrix} g\left(\boldsymbol{a}_{1,(k,G)}, b_{1,(k,G)}, \boldsymbol{x}_1\right) & \cdots & g\left(\boldsymbol{a}_{L,(k,G)}, b_{L,(k,G)}, \boldsymbol{x}_1\right) \\ \vdots & \ddots & \vdots \\ g\left(\boldsymbol{a}_{1,(k,G)}, b_{1,(k,G)}, \boldsymbol{x}_N\right) & \cdots & g\left(\boldsymbol{a}_{L,(k,G)}, b_{L,(k,G)}, \boldsymbol{x}_N\right) \end{bmatrix} \tag{10}$$

$$\boldsymbol{\theta}_{k,G+1} = \begin{cases} \boldsymbol{u}_{k,G+1} & \text{if}\, \mathrm{RMSE}_{\boldsymbol{\theta}_{k,G}} - \mathrm{RMSE}_{\boldsymbol{u}_{k,G+1}} > \epsilon \cdot \mathrm{RMSE}_{\boldsymbol{\theta}_{k,G}}, \\ \boldsymbol{u}_{k,G+1} & \text{if } \left|\mathrm{RMSE}_{\boldsymbol{\theta}_{k,G}} - \mathrm{RMSE}_{\boldsymbol{u}_{k,G+1}}\right| < \epsilon \cdot \mathrm{RMSE}_{\boldsymbol{\theta}_{k,G}} \\ & \text{and } \left\|\boldsymbol{\beta}_{\boldsymbol{u}_{k,G+1}}\right\| < \left\|\boldsymbol{\beta}_{\boldsymbol{\theta}_{k,G}}\right\|, \\ \boldsymbol{\theta}_{k,G} & \text{else.} \end{cases} \tag{11}$$

In the first generation, the population vector with the best RMSE is stored as $\boldsymbol{\theta}_{\mathrm{best},1}$ and $\mathrm{RMSE}_{\boldsymbol{\theta}_{\mathrm{best},1}}$.

Step 3. Mutation and Crossover

Similar to [22], for each target vector in the current generation, the trial vector generation strategy is chosen from a candidate pool constructed by 4 strategies mentioned in Sect. 2.2 according to probability $p_{l,G}$, where $p_{l,G}$ represents the probability that the strategy l

($l = 1, 2, 3, 4$) should be chose at the Gth generation. A fixed number of iterations (generations) is defined as the learning period (LP) in [22] and the probability $p_{l,G}$ is updated in the following ways

1. When $G \leq \text{LP}$, each strategy has the equal probability to be chosen, i.e. $p_{l,G} = \frac{1}{4}$.
2. When $G > \text{LP}$, $p_{l,G} = \frac{S_{l,G}}{\sum_{l=1}^{4} S_{l,G}}$ with
$S_{l,G} = \frac{\sum_{g=G-\text{LP}}^{G-1} ns_{l,g}}{\sum_{g=G-\text{LP}}^{G-1} ns_{l,g} + \sum_{g=G-\text{LP}}^{G-1} nf_{l,g}} + \varepsilon$, ($l = 1, 2, 3, 4$) where $ns_{l,g}$ denotes the number of trial vectors generated by the lth strategy at gth generations (iterations) that can successfully enter the next generation while $nf_{l,g}$ is the number of trial vectors generated by the lth strategy at gth generations (iterations) that are discarded in the next generation. LP generations' success and failure numbers of trial vectors are stored. Once the iterations go beyond the initial LP generations, the earliest records are removed and the new numbers in the current generation are stored. ε is a small positive constant value to avoid the possible null success rate.

Besides the probability update of trial vector generation strategy, a set of control parameters F and crossover rate CR are randomly generated for each target vector according to the normal distributions $\mathcal{N}(0.5, 0.3)$ and $\mathcal{N}(0.5, 0.1)$, respectively. For a given problem, proper values of CR usually fall into a small range, the mean value of CR is gradually adjusted according to previous CR values that have generated trial vectors successfully entering the next generation.

Step 4. Evaluation

All the trial vectors $\boldsymbol{u}_{k,G+1}$ generated at the $(G+1)$th generation are evaluated using equation (11) where ϵ is the preset small positive tolerance rate. The norm of the output weight $\|\boldsymbol{\beta}\|$ is added as one more criteria for the trial vector selection as pointed out in [23] that the neural networks tend to have better generalization performance with smaller weights. Steps 3) and 4) are repeated until the preset goal is reached or the maximum learning iterations are completed.

The number of populations NP is left as a user-specified parameters in the proposed SaE-ELM algorithm because it highly depends on the complexity of the given real world application. A validation set that has no overlap with the training set used in the initialization step is adopted in the evolutionary phase to avoid overfitting.

Remark 1 It is obvious that choosing a trial vector generation strategy in a pool would be more robust and practical than fixing the trial vector generation strategy for different real world applications. Moreover, the self-adapting probabilities and crossover rates according to previous experiences are able to ensure that the most suitable trial vector generation strategy and control parameters can be utilized in optimizing the network parameters. The lower the RMSE predicted by the trial vector generated with the trial vector generation strategy is, the larger the probability of applying this strategy to generate the trial vectors in the current generation is.

Remark 2 Note that besides the number of populations NP, no other parameters are needed to assign manually in the DE procedure of SaE-ELM. This makes SaE-ELM more feasible than DE-LM and E-ELM when it is applied to real world applications.

4 Performance Evaluation

The performance of SaE-ELM is compared with E-ELM and DE-LM on several regression and classification applications in this section. For E-ELM and DE-LM, eight combinations of manually choosing generation strategies and their associate parameters F and CR are used for evaluations. The followings show the details of these eight combinations.

Combination 1 (C1):
Strategy is set to be the rand/1 case and the other two parameters are set to be $F = 0.5$ and $CR = 0.3$.
Combination 2 (C2):
Strategy is set to be the rand/2 and the other two parameters are set to be $F = 0.5$ and $CR = 0.3$.
Combination 3 (C3):
Strategy is set to be the rand/1 and the other two parameters are set to be $F = 1$ and $CR = 0.8$.
Combination 4 (C4):
Strategy is set to be the rand/2 and the other two parameters are set to be $F = 1$ and $CR = 0.8$.
Combination 1 (C5):
Strategy is set to be the rand-to-best/2 case and the other two parameters are set to be $F = 0.5$ and $CR = 0.3$.
Combination 2 (C6):
Strategy is set to be the current-to-rand/1 and the other two parameters are set to be $F = 0.5$ and $CR = 0.3$.
Combination 3 (C7):
Strategy is set to be the rand-to-best/2 and the other two parameters are set to be $F = 1$ and $CR = 0.8$.
Combination 4 (C8):
Strategy is set to be the current-to-rand/1 and the other two parameters are set to be $F = 1$ and $CR = 0.8$.

The experimental results are also compared with several related methods, namely composite function wavelet neural network with ELM (CFWNN-ELM) [24], LM [2] and SVM [9]. All these simulations are conducted in Matlab 7.4 environment running on an ordinary PC with 2.66 GHZ CPU and 2 GB RAM. For SaE-ELM, E-ELM, DE-LM, LM and CFWNN-ELM, the suitable numbers of hidden nodes for all these algorithms are determined by using a validation dataset. Each training dataset is separated into two nonoverlapped datasets: the training and the validation datasets. When gradually increasing the number of hidden nodes in a preset region, the one with the lowest validation error is selected as the suitable number of hidden nodes. For SVM, the radial basis function (RBF) is used as the kernel function, the cost parameter C and the kernel parameter γ are searched in a grid formed by $C = \left[2^{12}, 2^{11}, \ldots, 2^{-2}\right]$ and $\gamma = \left[2^{4}, 2^{3}, \ldots, 2^{-10}\right]$ as suggested in [9] and the best parameters are then obtained in terms of the generalized performance. For all applications, 50 trials of simulations are conducted and the final results are obtained by averaging all these trials. For all these results, the best RMSE, standard deviation, and classification rate are shown in bold face in the table.

4.1 Evaluation on Regression Problems

In this subsection, the performance of SaE-ELM is first tested on several regression problems, including an artificial function approximation and 6 real world datasets regression problems.

Table 1 Results comparisons for the function approximation

NNs	Strategy	Testing		Training time (s)	Nodes
		RMSE	Dev		
SaE-ELM		**0.0411**	**0.0094**	116.6797	100
E-ELM	C1	0.1149	0.0308	179.0352	140
	C2	0.0731	0.0231	178.3164	140
	C3	0.4254	0.0923	180.8594	140
	C4	0.4500	0.1019	177.6734	140
	C5	0.0722	0.0207	178.8842	140
	C6	0.1200	0.0221	178.4450	140
	C7	0.4100	0.0815	177.8465	140
	C8	0.4274	0.1005	177.4425	140
DE-LM	C1	0.2121	0.0925	636.1773	140
	C2	0.2097	0.0999	675.7156	140
	C3	0.4789	0.2812	687.9047	140
	C4	0.3781	0.1131	707.1477	140
	C5	0.1987	0.0755	687.2532	140
	C6	0.2012	0.0815	646.2551	140
	C7	0.4745	0.2756	687.7745	140
	C8	0.3652	0.1541	707.6574	140

4.1.1 Approximation of a two-dimensional function with noise

SaE-ELM is used to approximate a two-dimensional function with the following mathematical model

$$f = \left(x_1^2 - x_2^2\right) \sin\left(0.5x_1\right), \quad x_1, x_2 \in [-10, 10].$$

For all three algorithms SaE-ELM, E-ELM and DE-LM, NP is set to be 80. A training dataset $((x_{1i}, x_{2i}), f(i))$ and a testing dataset $((x_{1i}, x_{2i}), f(i))$ are collected where (x_{1i}, x_{2i}) are uniformly randomly distributed over the input domain $[-10, 10] \times [-10, 10]$, which forms a 40×40 grid. To make the approximation 'real', uniformly distributed noise in $[-0.2, 0.2]$ has been added to all the training samples, while the testing data remain noise-free. For all these algorithms, 30 % of the training dataset is used as the validation dataset.

Table 1 shows the average results of all these three algorithms for approximating the function f. It is easy to find that SaE-ELM wins the lowest testing RMSE 0.0411 and the lowest standard deviation (Dev) 0.0094. Moreover, it learns faster than E-ELM and DE-LM. From this table, we also find that different combinations of trial vector generation strategies and its associate control parameters adopted in E-ELM and DE-LM do have different performances. The combination 5 is the most efficient one for both E-ELM and DE-LM.

Figure 1 shows the variations of the probabilities of the eight trial vector generation strategies during the evolution phase of learning function f using SaE-ELM. In the first 20 iterations, each strategy has equal probability $\frac{1}{4}$. After this learning period, the probability of each strategy is updated with the number of trial vectors generated by this strategy that can successfully enter to the next generation. From this figure, we see that the"DE/rand-to-best/2" strategy has the most successful trial vectors. The testing RMSEs of approximating function f using SaE-ELM and E-ELM with respect to different number of hidden nodes are shown in Fig. 2, where Ci ($i = 1, 2, \ldots, 8$) in the figure represents the eight different combinations used for E-ELM correspondingly. The proposed SaE-ELM wins the lowest testing RMSEs at all cases.

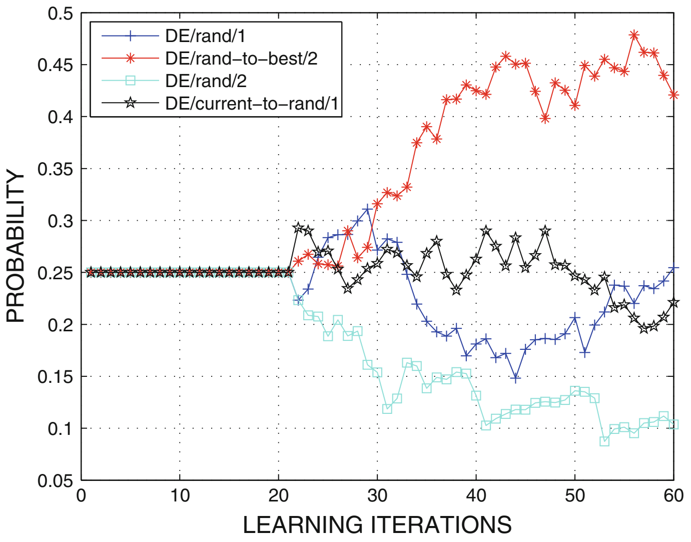

Fig. 1 Variations of the probabilities of four trial vector generating strategies for learning the function f

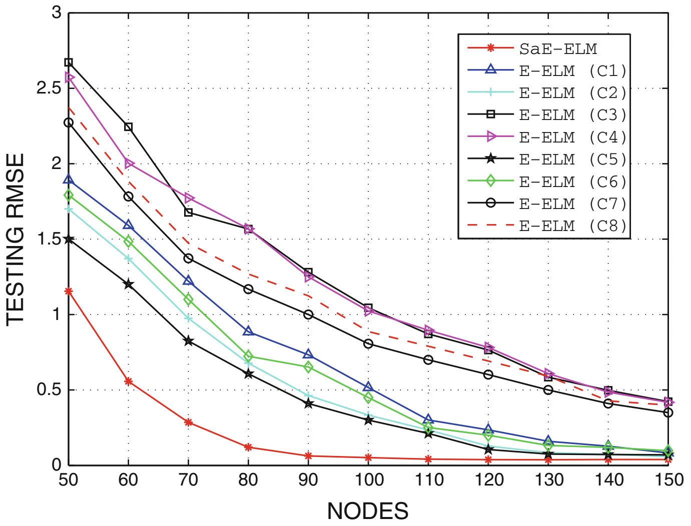

Fig. 2 Testing RMSEs for approximating the function f w.r.t different nodes

Table 2 shows the comparisons of the approximating results between the proposed SaE-ELM method and three existing methods: CFWNN-ELM, LM and support vector machine for regression (SVR). Compared with these three approaches, SaE-ELM is able to achieve the lowest testing RMSE and standard deviation.

Table 2 Results comparisons with several existing methods

NNs	Testing		Training time (s)	Nodes/ SVs
	RMSE	Dev		
SaE-ELM	**0.0411**	**0.0094**	116.6797	100
LM	1.2461	0.3823	44.6602	180
CFWNN-ELM	0.3938	0.2005	0.1944	180
SVR	0.4527	0.1880	397.6044	912.94

Table 3 Specifications of real-world regression datasets

Datasets	Data		Attributes	
	Training	Testing	Continuous	Nominal
Autompg	220	178	8	0
Price	80	79	14	1
Housing	250	256	13	0
CPU	100	109	6	0
Servo	80	87	0	4
Cancer	100	94	32	0

4.1.2 Evaluation on real-world data regressions

In this simulation, the performances of SaE-ELM, E-ELM and DE-LM are compared on the regressions of 6 real-world datasets covering various domains from the UCI database[1], where these 6 datasets are Autompg, Auto Price (Price), Boston Housing (Housing), Machine CPU (CPU), Servo and Wiscoin Breast Cancer (Cancer). The following lists a detail description of these 6 real-world datasets.

Autompg: This dataset concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes.
Auto Price: This dataset concerns the insurance risk rating corresponds to the degree to which the auto is more risky than its price indicates.
Boston Housing: This dataset concerns the task of predicting housing values in areas of Boston.
Machine CPU: This dataset concerns the instance-based prediction of relative cpu performance using 6 predictive attributes.
Servo: This dataset concerns a robot control problem. The dataset was from a simulation of a servo system involving a servo amplifier, a motor, a lead screw/nut, and a sliding carriage of some sort. The output value is almost certainly a rise time, or the time required for the system to respond to a step change in a position set point.
Wiscoin Breast Cancer: This dataset concerns the predicting time to recur of breast cancer. The breast cancer databases were obtained from the University of Wisconsin Hospitals.

Table 3 lists the specifications of these benchmark problems. All the attributes (inputs) have been normalized to the range of $[-1, 1]$ while the outputs have been normalized into

[1] http://www.ics.uci.edu/~mlearn/MLRepository.html.

Table 4 Results comparisons for real-world data regressions

Data	NNs	Strategy	Testing		Training time (s)	Nodes
			RMSE	Dev		
Autompg	SaE-ELM		**0.0631**	0.0100	12.0641	26
	E-ELM	C1	0.0718	0.0138	2.4375	30
		C2	0.0691	0.0129	2.1867	30
		C3	0.0752	0.0141	2.3609	30
		C4	0.0743	0.0113	2.2516	30
		C5	0.0703	0.0130	2.3251	30
		C6	0.0698	0.0142	2.2765	30
		C7	0.0744	0.0135	2.2511	30
		C8	0.0728	0.0105	2.2014	30
	DE-LM	C1	0.0706	0.0084	3.0578	10
		C2	0.0726	0.0093	3.6281	12
		C3	0.0701	0.0115	5.5031	16
		C4	0.0761	0.0080	3.7437	12
		C5	0.0702	0.0074	3.1024	12
		C6	0.0741	0.0120	3.2857	12
		C7	0.0722	0.0111	3.7452	12
		C8	0.0734	0.0113	3.5246	12
Price	SaE-ELM		**0.0437**	0.0094	5.8500	16
	E-ELM	C1	0.0489	0.0063	0.6992	16
		C2	0.0494	0.0094	0.7453	18
		C3	0.0499	0.0138	0.6172	14
		C4	0.0508	0.0102	0.7008	16
		C5	0.0472	0.0074	0.7451	16
		C6	0.0483	0.0112	0.8845	16
		C7	0.0502	0.0121	0.7564	16
		C8	0.0521	0.0101	0.6254	16
	DE-LM	C1	0.0502	0.0067	1.1938	2
		C2	0.0506	0.0071	1.1344	2
		C3	0.0536	0.0086	1.1656	2
		C4	0.0515	0.0114	1.0391	2
		C5	0.0497	0.0098	1.2015	2
		C6	0.0512	0.0102	1.1867	2
		C7	0.0523	0.0121	1.2841	2
		C8	0.0542	0.0113	1.1524	2
Housing	SaE-ELM		**0.0904**	0.0132	12.7141	26
	E-ELM	C1	0.0999	0.0149	2.1680	30
		C2	0.0965	0.0137	1.7914	24
		C3	0.0982	0.0089	2.3445	30
		C4	0.1009	0.0117	1.8898	26
		C5	0.0985	0.0138	2.0815	30

Table 4 continued

Data	NNs	Strategy	Testing		Training time (s)	Nodes
			RMSE	Dev		
		C6	0.0998	0.0142	2.0125	30
		C7	0.0976	0.0125	1.9756	30
		C8	0.0986	0.0105	1.8852	30
	DE-LM	C1	0.0950	0.0125	1.8734	4
		C2	0.0956	0.0097	2.5145	6
		C3	0.0942	0.0199	2.6148	6
		C4	0.0949	0.0163	2.6492	6
		C5	0.0963	0.0145	2.1542	6
		C6	0.0983	0.0139	2.1867	6
		C7	0.0975	0.0123	2.6321	6
		C8	0.0968	0.0133	2.5421	6
CPU	SaE-ELM		**0.0396**	0.0083	1.5422	8
	E-ELM	C1	0.0528	0.0260	0.5391	10
		C2	0.0528	0.0238	0.5008	8
		C3	0.0504	0.0187	0.5852	12
		C4	0.0518	0.0216	0.5445	10
		C5	0.0502	0.0238	0.5375	10
		C6	0.0588	0.0229	0.5867	10
		C7	0.0576	0.0181	0.5609	10
		C8	0.0542	0.0203	0.5516	10
	DE-LM	C1	0.0609	0.0217	0.8898	2
		C2	0.0601	0.0227	0.8977	2
		C3	0.0565	0.0170	0.8555	2
		C4	0.0662	0.0245	1.3805	6
		C5	0.0598	0.0238	0.9875	2
		C6	0.0602	0.0229	0.8567	2
		C7	0.0652	0.0241	0.9936	2
		C8	0.0532	0.0213	0.9562	2
Servo	SaE-ELM		**0.0884**	0.0161	2.7266	30
	E-ELM	C1	0.1032	0.0213	0.8078	22
		C2	0.1051	0.0226	0.7445	20
		C3	0.0964	0.0252	0.8078	22
		C4	0.0922	0.0202	0.8859	24
		C5	0.0988	0.0222	0.8542	24
		C6	0.1023	0.0235	0.8845	24
		C7	0.0996	0.0256	0.8752	24
		C8	0.1052	0.0231	0.8975	24
	DE-LM	C1	0.0963	0.0210	1.6750	10
		C2	0.1008	0.0214	1.3773	8
		C3	0.1017	0.0214	1.4898	8

Table 4 continued

Data	NNs	Strategy	Testing RMSE	Testing Dev	Training time (s)	Nodes
		C4	0.1109	0.0169	1.6961	10
		C5	0.1025	0.0251	1.5689	10
		C6	0.1001	0.0221	1.6231	10
		C7	0.1032	0.0234	1.5524	10
		C8	0.1004	0.0205	1.6541	10
Cancer	SaE-ELM		**0.2561**	0.0217	3.0250	12
	E-ELM	C1	0.2618	0.0221	0.5477	8
		C2	0.2663	0.0208	0.6102	10
		C3	0.2685	0.0220	0.5422	8
		C4	0.2778	0.0325	0.5516	8
		C5	0.2652	0.0213	0.5521	10
		C6	0.2714	0.0231	0.6241	10
		C7	0.2688	0.0204	0.5241	10
		C8	0.2742	0.0221	0.6322	10
	DE-LM	C1	0.4142	0.0740	4.7086	2
		C2	0.4194	0.0712	4.9375	2
		C3	0.3883	0.0569	4.6266	2
		C4	0.3674	0.0644	4.7844	2
		C5	0.3956	0.0552	4.8521	2
		C6	0.4421	0.0852	4.1125	2
		C7	0.4025	0.0456	4.7895	2
		C8	0.4006	0.0687	4.2356	2

[0, 1] in our simulations. For each trial of simulations, the training set and testing set are randomly generated from the whole dataset with the partition number shown in Table 3. In our simulations, 30 % of the training dataset is used as the validation dataset for all these algorithms. The number of populations NP is set to be 80 for all these three algorithms.

Table 4 lists the averaging results of multiple trials using all these three algorithms. For all these algorithms, the number of hidden nodes is gradually increased and the one with the best testing RMSE is reported in this table. It is easy to find that SaE-ELM achieves the lowest testing RMSEs in all 6 datasets among these three algorithms. For different datasets, different combinations of generation strategies and control parameters perform different in E-ELM and DE-LM. For E-ELM, the first combination has the best performance in the dataset Breast Cancer among all these eight combinations, while the second combination is most suitable for the datasets Autompg and Boston Housing. However, for dataset Price, the sixth combination used in E-ELM has the best performances, for dataset Machine CPU, the fifth combination used in E-ELM has the best performances and for Servo, the fourth combination has the best performance. For DE-LM, the first combination is most suitable for the dataset Servo, the third combination is most suitable for the datasets Autompg, Boston Housing, the forth combination is most suitable for the Breast Cancer dataset, the fifth combination

Table 5 Results comparisons for real-world data regressions with several existing methods

Data	Methods	Testing		Training time (s)	Nodes/SVs
		RMSE	Dev		
Autompg	SaE-ELM	**0.0631**	0.0100	12.0641	26
	LM	0.0764	0.0079	0.4078	2
	CFWNN-ELM	0.0743	0.0115	0.0084	50
	SVR	0.0987	0.0026	22.336	80.54
Price	SaE-ELM	**0.0437**	0.0094	5.8500	16
	LM	0.1157	0.0231	0.2216	5
	CFWNN-ELM	0.0912	0.0209	0.0001	30
	SVR	0.0945	0.0098	5.2988	22.75
Housing	SaE-ELM	**0.0904**	0.0132	12.7141	26
	LM	0.1042	0.0144	0.3837	2
	CFWNN-ELM	0.1067	0.0114	0.0109	60
	SVR	0.1102	0.0069	13.399	57.24
CPU	SaE-ELM	**0.0396**	0.0083	1.5422	8
	LM	0.0826	0.0715	0.2055	10
	CFWNN-ELM	0.0910	0.0326	0.0016	10
	SVR	0.0811	0.0180	1.0018	7.8
Servo	SaE-ELM	**0.0884**	0.0161	2.7266	30
	LM	0.1276	0.0475	0.2247	10
	CFWNN-ELM	0.1577	0.0214	0.0023	25
	SVR	0.1177	0.0185	2.2245	22.4
Cancer	SaE-ELM	**0.2561**	0.0217	3.0250	12
	LM	0.3155	0.0962	0.3266	5
	CFWNN-ELM	0.2781	0.0128	$< 10^{-4}$	5
	SVR	0.2643	0.0151	3.4245	74.3

is most suitable for the Price dataset while the eighth combination is more suitable for the Machine CPU dataset.

Table 5 shows the comparisons between the proposed SaE-ELM with several existing algorithms. All these results are collected by averaging multiple trials. The number of hidden nodes for LM and CFWNN-ELM is gradually increased and the one with the best generalization performance is reported here. From this table, we find that SaE-ELM wins the lowest testing RMSEs in all 6 datasets among all these four algorithms. For some applications, the performances have been dramatically improved. For example, the testing RMSE of Machine CPU using SaE-ELM is 0.0396. However, for the other three algorithms, the testing RMSEs of this dataset are all higher than 0.0500. CFWNN-ELM has the shortest training time among all these algorithms. The training time used by SaE-ELM is longer than LM and CFWNN-ELM due to its incorporating the differential evolution into network parameter optimization.

4.2 Evaluation on Classification Problems

In this section, the performance of SaE-ELM is evaluated on 5 real world datasets classification problems while the first 4 datasets are from the UCI database, namely the Pima Indians Diabetes dataset (Diabetes), the Heart Disease dataset (Disease), the Iris data set (Iris) and the Italy Wine dataset (Wine), and the last one is from the Protein Information Resource (PIR[2]) center. The following lists a detail description of these 5 real-world datasets.

[2] http://pir.georgetown.edu/.

Table 6 Specifications of real-world classification datasets

Datasets	Data		Attributes	Classes
	Training	Testing		
Diabetes	500	268	8	2
Disease	100	170	13	2
Iris	100	50	4	3
Wine	100	78	13	3
Protein Sequences	949	533	56	10

Diabetes: In this dataset, the diagnostic, binary-valued variable investigated is whether the patient shows signs of diabetes according to World Health Organization criteria.
Disease: This dataset concerns the presence of heart disease in the patient by using 13 attributes.
Iris: The dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant.
Wine: These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.
Protein Sequences: This dataset concerns the protein sequence classification problem by using a 56-dimensional feature signal to 10 classes.

The specification of these 5 datasets are shown in Table 6. All the attributes (inputs) have been normalized to the range of $[-1, 1]$ in our simulations and for each trial of simulations, the training set and testing set are randomly generated from the whole dataset with the partition number shown in Table 6. In our simulations, 30 % of the training dataset is used as the validation dataset for all these algorithms.

The averaging classification results of multiple trials for all these 5 real world datasets are shown in Table 7. For SaE-ELM and E-ELM, NP is set to be 20 for all datasets. For DE-LM, NP is set to be 600 and 80 for the first two datasets while for the last three datasets, NP is set to be 20. For these three algorithms, the number of hidden nodes is gradually increased and the one with the best testing rate is reported in this table. We can easily find that SaE-ELM achieves the highest success testing rates in all 5 datasets among these three algorithms. With manual determining generation strategies and control parameters in E-ELM and DE-LM, different combinations lead to different generalization performances for different datasets. Among all these eight combinations for E-ELM, the first combination performs best in Disease and Iris, the second combination has the highest testing rate in Diabetes, the third combination is most suitable for Wine and the last combination is most suitable for Protein Sequences. However, for DE-LM, the first combination is most suitable for the dataset Iris, the second combination has the best performance on Diabetes, the third combination is most effective in classifying the datasets Disease and Protein Sequences while the sixth combination is most suitable for the dataset Wine. For all these datasets, the training time by SaE-ELM is slightly longer than E-ELM. However, SaE-ELM learns much faster than DE-LM in general.

We also compare the classification performances of SaE-ELM with three existing algorithms: LM, CFWNN-ELM and SVM. Table 8 shows the averaging results of all these four approaches on these 5 real world datasets. The one with the best testing rate is shown in boldface. It is easy to find that SaE-ELM is able to achieve the highest success testing rates among

Table 7 Results comparisons for real-world data classifications

Data	NNs	Strategy	Testing		Training time (s)	Nodes
			Rate	Dev		
Diabetes	SaE-ELM		**79.55**	2.68	5.6422	20
	E-ELM	C1	77.24	3.53	1.1244	22
		C2	78.09	3.53	1.4750	30
		C3	77.01	3.12	1.0494	22
		C4	76.94	3.20	1.0706	22
		C5	77.02	3.64	1.6521	22
		C6	77.25	3.42	1.2152	22
		C7	76.06	3.52	1.1754	22
		C8	76.42	3.21	1.2310	22
	DE-LM	C1	76.90	1.78	4.7258	2
		C2	77.50	2.08	4.6297	2
		C3	76.75	1.91	4.4586	2
		C4	76.60	2.16	8.5188	4
		C5	76.21	2.83	4.5263	2
		C6	76.01	2.45	4.8856	2
		C7	76.44	3.12	4.2265	2
		C8	75.94	2.97	4.3526	2
Disease	SaE-ELM		**82.53**	3.86	0.7086	18
	E-ELM	C1	81.62	4.29	0.2247	16
		C2	81.46	3.42	0.2434	18
		C3	80.59	3.66	0.2191	16
		C4	80.96	3.71	0.2062	14
		C5	81.09	3.45	0.2563	14
		C6	81.21	3.76	0.3265	14
		C7	80.05	3.44	0.2588	14
		C8	81.06	3.64	0.2051	14
	DE-LM	C1	77.56	3.51	2.8375	2
		C2	77.29	3.13	2.7469	2
		C3	79.06	2.28	2.8203	2
		C4	78.56	2.54	2.8148	2
		C5	78.22	3.53	2.9874	2
		C6	77.43	3.12	2.8755	2
		C7	77.94	3.20	2.7956	2
		C8	77.88	3.20	2.8564	2
Iris	SaE-ELM		**97.20**	4.12	0.2289	18
	E-ELM	C1	96.48	3.67	0.1522	16
		C2	96.28	3.39	0.1600	16
		C3	96.40	3.15	0.1758	20
		C4	95.80	3.78	0.1500	16
		C5	96.09	3.32	0.1423	16
		C6	96.25	3.34	0.1522	16

Table 7 continued

Data	NNs	Strategy	Testing		Training time (s)	Nodes
			Rate	Dev		
		C7	95.96	3.77	0.1788	16
		C8	96.12	3.45	0.1652	16
	DE-LM	C1	96.80	2.71	1.6133	8
		C2	96.10	2.63	1.3789	6
		C3	96.40	1.79	1.1164	4
		C4	96.10	2.55	1.1820	4
		C5	96.09	2.88	1.3521	4
		C6	96.01	2.12	1.2877	4
		C7	95.87	2.20	1.3566	6
		C8	96.94	2.21	1.2546	6
Wine	SaE-ELM		**97.95**	2.54	0.3050	22
	E-ELM	C1	97.18	2.44	0.2100	20
		C2	96.97	2.82	0.2728	26
		C3	97.28	2.71	0.2319	22
		C4	96.97	2.63	0.2516	22
		C5	96.09	2.88	0.3214	30
		C6	97.01	2.76	0.2588	22
		C7	96.43	2.45	0.2745	22
		C8	96.85	2.54	0.2653	22
	DE-LM	C1	96.28	1.76	1.6461	4
		C2	96.73	1.88	1.6891	4
		C3	96.03	1.99	2.2023	6
		C4	96.22	2.14	1.7109	4
		C5	96.59	2.15	1.8532	4
		C6	97.05	1.87	1.9865	4
		C7	96.68	2.20	2.0706	4
		C8	97.11	1.65	1.9985	4
Protein Sequences	SaE-ELM		**97.53**	0.47	43.6344	80
	E-ELM	C1	97.15	0.47	11.9469	105
		C2	97.08	0.77	10.4125	100
		C3	97.23	0.47	12.2406	110
		C4	97.27	0.59	8.8266	90
		C5	96.22	0.53	11.8852	100
		C6	97.05	0.63	11.2345	100
		C7	96.52	0.57	11.6578	100
		C8	96.43	0.46	11.2245	100
	DE-LM	C1	88.78	1.51	312.0344	25
		C2	88.54	1.92	299.1984	25
		C3	89.92	1.19	385.2250	25
		C4	87.88	1.14	319.0953	25

Table 7 continued

Data	NNs	Strategy	Testing		Training time (s)	Nodes
			Rate	Dev		
		C5	88.56	1.28	308.0789	25
		C6	87.98	1.34	311.0024	25
		C7	89.52	1.16	316.0452	25
		C8	88.87	1.22	313.0215	25

Table 8 Results comparisons for real-world data classifications with several existing methods

Data	Methods	Testing (%)		Training time (s)	Nodes/SVs
		Rate	Dev		
Diabetes	SaE-ELM	**79.55**	2.68	5.6422	20
	LM	74.73	3.2	2.8978	20
	CFWNN-ELM	78.02	2.56	0.0102	35
	SVM	77.31	2.73	364.6	62.28
Disease	SaE-ELM	**82.53**	3.86	0.7086	18
	LM	71.75	6.67	0.3066	20
	CFWNN-ELM	76.85	3.43	0.0016	30
	SVM	76.10	3.46	6.7094	81.6
Iris	SaE-ELM	**97.20**	4.12	0.2289	18
	LM	95.40	3.19	0.3641	10
	CFWNN-ELM	95.92	3.05	0.0009	20
	SVM	94.36	2.76	4.5488	23.3
Wine	SaE-ELM	**97.95**	2.54	0.3050	22
	LM	92.97	4.36	0.4988	20
	CFWNN-ELM	95.15	3.13	0.0028	30
	SVM	97.48	1.57	5.8941	47.3
Protein Sequences	SaE-ELM	**97.53**	0.47	43.6344	80
	LM	87.45	2.13	1326	60
	CFWNN-ELM	87.48	2.29	0.0859	130
	SVM	90.63	0.01	274.9672	261

these four algorithms in all cases. CFWNN-ELM learns much faster than other approaches. However, the training time by SaE-ELM is shorter than the one used by SVM in general.

5 Conclusion

In this paper, we have developed a novel learning algorithm named self-adaptive evolutionary extreme learning machine for single hidden layer feedforward networks[3]. By incorporating the self-adaptive differential evolution algorithm to optimize the network hidden node param-

[3] The source code for SaE-ELM is available at http://www.extreme-learning-machines.org.

eters and employing the extreme learning machine to derived the network output weights, SaE-ELM has shown several salient features. SaE-ELM not only avoided limitations existed in E-ELM and DE-LM, where both algorithms suffered from manually selecting the trial vector generation strategies and control parameters, but also improved the generalization performance. Comparisons with E-ELM and DE-LM using eight different combinations of trial vector generation strategies and control parameters on several experiments in regression and classification have demonstrated that SaE-ELM did outperform these two algorithms in general. Comparisons with three existing approaches LM, CFWNN-ELM and SVM have also shown that SaE-ELM can efficiently improve the network generalization performance. In the future work, we will try to find a more efficient way to reduce the training time by the proposed SaE-ELM algorithm.

Acknowledgements The authors would like to thank the anonymous reviewers whose insightful and helpful comments greatly improved this paper.

References

1. Huang G-B, Zhu Q-Y, Siew C-K (2004) Extreme learning machine: a new learning scheme of feedforward neural networks. In: Proceedings of international joint conference on neural networks, Budapest, Hungary, pp 985–990
2. Huang G-B, Zhu Q-Y, Siew C-K (2006) Extreme learning machine: theory and applications. Neurocomputing 70(1-3):489–501
3. Cao JW, Lin ZP, Huang G-B (2011) Composite function wavelet neural networks with differential evolution and extreme learning machine. Neural Process Lett 33(3):251–265
4. Cao JW, Lin ZP, Huang G-B (2012) Voting based extreme learning machine. Inf Sci 185(1):66–77
5. Balabin RM, Safieva RZ, Lomakina EI (2010) Gasoline classification using near infrared (NIR) spectroscopy data: comparison of multivariate techniques. Anal Chim Acta 671:27–35
6. Balabin RM, Safieva RZ (2011) Biodiesel classification by base stock type (vegetable oil) using near infrared (NIR) spectroscopy data. Anal Chim Acta 689(2):190–197
7. Balabin RM, Lomakina EI (2011) Support vector machine regression (LS-SVM)—an alternative to artificial neural networks (ANNs) for the analysis of quantum chemistry data?. Phys Chem Chem Phys 13(24):11710–11718
8. Balabin RM, Lomakina EI (2011) Support vector machine regression (SVR/LS-SVM) - an alternative to neural networks (ANN) for analytical chemistry? Comparison of nonlinear methods on near infrared (NIR) spectroscopy data. Analyst 136(8):1703–1712
9. Hsu C-W, Lin C-J (2002) A comparison of methods for multiclass support vector machines. IEEE Trans Neural Netw 13(2):415–425
10. Levenberg K (1944) A method for the solution of certain non-linear problems in least squares. Q Appl Math 2:164–168
11. Hagan MT, Menhaj MB (1994) Training feedforward networks with the marquardt algorithm. IEEE Trans Neural Netw 5(6):989–993
12. Huang G-B, Chen L, Siew CK (2006) Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans Neural Netw 17(4):879–892
13. Liang N-Y, Huang G-B, Saratchandran P, Sundararajan N (2006) A fast and accurate on-line sequential learning algorithm for feedforward networks. IEEE Trans Neural Netw 17(6):1411–1423
14. Huang G-B, Chen L (2007) Convex incremental extreme learning machine. Neurocomputing 70(16-18):3056–3062
15. Huang G-B, Chen L (2008) Enhanced random search based incremental extreme learning machine. Neurocomputing 71(16-18):3460–3468
16. Rong H-J, Huang G-B, Saratchandran P, Sundararajan N (2009) On-line sequential fuzzy extreme learning machine for function approximation and classification problems. IEEE Trans Syst Man Cybern B 39(4):1067–1072
17. Lan Y, Soh YC, Huang G-B (2009) Ensemble of online sequential extreme learning machine. Neurocomputing 72(13-15):3391–3395

18. Storn R, Price K (2004) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11(4):341–359
19. Ilonen J, Kamarainen JI, Lampinen J (2003) Differential evolution training algorithm for feedforward neural networks. Neural Process Lett 17:93–105
20. Subudhi B, Jena D (2008) Differential evolution and levenberg marquardt trained neural network scheme for nonlinear system identification. Neural Process Lett 27:285–296
21. Zhu Q-Y, Qin A-K, Suganthan P-N, Huang G-B (2005) Evolutionary extreme learning machine. Pattern Recog 38(10):1759–1763
22. Qin A-K, Huang V-L, Suganthan P-N (2009) Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans Evol Comput 13(2):398–417
23. Bartlett PL (1998) The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. IEEE Trans Inf Theory 44(2):525–536
24. Cao JW, Lin ZP, Huang G-B (2010) Composite function wavelet neural networks with extreme learning machine. Neurocomputing 73(7-9):1405–1416
25. G*ä*mperle R, M*ü*ller SD, Koumoutsakos P (2002) A parameter study for differential evolution. In: Proceedings of WSEAS international conference on advances in intelligent systems, fuzzy systems, evolutionary computation. Interlaken, Switzerland, pp 293–298
26. Ronkkonen J, Kukkonen S, Price KV (2005) Real-parameter optimization with differential evolution. In: Proceedings of IEEE congress evolutionary computation, Edinburgh, Scotland, pp 506–513