Neural Process Lett (2013) 37:277–301
DOI 10.1007/s11063-012-9246-9

Parallel Chaos Search Based Incremental Extreme Learning Machine

Yimin Yang · Yaonan Wang · Xiaofang Yuan

Published online: 22 September 2012

Abstract Recently, a simple and efficient learning steps referred to as extreme learning machine (ELM), was proposed by Huang et al. , which has shown that compared to some conventional methods, the training time of neural networks can be reduced even by thousands of times. However, recent study showed that some of random hidden nodes may paly a very minion role in the network output and thus eventually increase the network complexity. This paper proposes a parallel chaos search based incremental extreme learning machine (PC-ELM) with additional steps to obtain a more compact network architecture. At each learning step, optimal parameters of hidden node that are selected by parallel chaos optimization algorithm will be added to exist network in order to minimize the residual error between target function and network output. The optimization method is proposed parallel chaos optimization method. We prove the convergence of PC-ELM both in increased network architecture and fixed network architecture. Then we apply this approach to several regression and classification problems. Experiment of 19 benchmark testing data sets are used to test the performance of PC-ELM. Simulation results demonstrate that the proposed method provides better generalization performance and more compact network architecture.

Keywords Extreme learning machine · Convergence rate · Chaos optimization algorithm · Random hidden nodes

1 Introduction

In the past decades feedforward neural networks (FNNs) has been investigated extensively from both theoretical and application aspects. According to [1], 95 % of neural networks (NNs) literatures are mainly on FNNs. Conventional NNs theories [2] showed that all the

Y. Yang (✉) · Y. Wang · X. Yuan
College of Electrical and Information Engineering,
Hunan University, Changsha 410082, China
e-mail: yangyi_min@126.com

parameters of FNNs need to be adjusted and there exists dependence among the parameters of different layers. However, tuning all the parameters of the networks may cause learning complicated and inefficient since tuning based learning may easily converge to local minima and/or may generally be very slow due to improper learning steps. Iterative learning algorithms especially descent techniques such as back propagation (BP) methods are often used in learning FNNs.

As a specific type of FNNs, the single-hidden-layer feedforward network (SLFNs) plays an important role in practical applications. Unlike conventional NNs theories, Huang and Chen [3,4] recently have proved that SLFNs with additive or RBF hidden nodes and with randomly generated hidden node parameter can work as universal approximators by only calculating the output weights linking the hidden layer to the output nodes. Thus iterative techniques are not required in adjusting parameters of SLFNs at all. Based on the universal approximation capability of SLFNs with random hidden nodes, Huang et al. in [3] and in [5] respectively proposed simple and efficient learning steps both with increased network architecture (I-ELM) and with fixed network architecture (ELM). This two methods make the selection of the weights of the hidden neurons very fast in the case of SLFNs. Hence, the overall computational time for model structure selection and actual training of the model is often reduced even by thousands of times, compared to some conventional methods. Based on incremental extreme learning machine (I-ELM) [3], resulting methods with the mechanism of growth hidden nodes are proposed such as enhanced incremental ELM (EI-ELM) [6], optimal pruned ELM (OP-ELM) [7], Convex incremental ELM (CI-ELM) [8] are proposed to obtain better performance. Whereas, based on extreme learning machine (ELM) [5], resulting methods are developed with fixed network architecture. Such as online sequential learning model ELM (OS-ELM) [9], error minimized ELM (EM-ELM) [1].

One major issues, however, still remain in ELM. As mentioned by Huang et al. [6], some of the hidden nodes may play a very minor role in the network output and may eventually increase the network complexity. For this reason, Huang et al. proposed EI-ELM to obtain compact network architecture. In EI-ELM, several parameters of hidden nodes at each learning step are randomly generated and among them, the hidden node parameter leading to the largest residual error decreasing will be added to the existing network. This research result implies that identification of hidden nodes parameters can be considered as a kind of optimization process. But, we notice that based on I-ELM, EI-ELM can only work for SLFNs with increased network architecture I-ELM. Further more, EI-ELM is acting like a global search, and its search mode is limited by random selection. Thus, if some optimization algorithms are used to find best hidden nodes parameters, we can obtain more higher searching efficacy, meanwhile, obtain compact network architecture.

This paper comes from our belief that some modified optimization algorithm should be used to find the best hidden node parameters. The best parameters of hidden node that leading to the largest residual error decreasing will be added to the existing network. This paper propose a learning algorithm referred to as parallel chaos search based incremental extreme learning machine (PC-ELM) that can handle either in creased network architecture or in fixed network architecture. PC-ELM can find the best input weights and biases at each learning step by proposed parallel chaos optimization algorithm (PCOA), which greatly improve the efficacy of ELM and obtain a more compact network architecture. Results show that compared with EI-ELM, I-ELM, ELM, C-ELM and EM-ELM, proposed PC-ELM can efficiently reduce the network complexity, meanwhile, obtain good generalization performance.

2 Incremental Extreme Learning Machine

The PC-ELM methodology is based on the original ELM algorithm from which it borrows the original SLFNs construction. In the following, the main concepts and theory of the ELM algorithm are shortly reviewed.

The theoretical proofs and a more thorough presentation of the ELM algorithm are detailed in the original paper [5]. The way to calculate the output weigh β from the knowledge of the hidden-layer output matrix $\mathbf{H}$ and target values is proposed with the use of a Moore-Penrose generlized inverse of the matrix $\mathbf{H}$. Overall, the detail ELM algorithm is shown as follow.

Algorithm 1 Pseudocode for I-ELM

Given a training set $(\mathscr{X}, \mathscr{T})$,$\mathscr{X} \in \mathbb{R}^m$, $\mathscr{T} \in \mathbb{R}^n$, an activation function $\mathbf{H}$, number L of hidden nodes, required learning accuracy η.
Step (1) **Initialization**: Let $L = 0$ and residual error $E = \mathscr{T}$.
Step (2) **Learning step**:
While $L < L_{max}$, $|E| > \eta$
increase by one the number of hidden nodes $L : L = L + 1$
(a) assign random input weight $\mathbf{w}$ and bias $\mathbb{b}$ for hidden nodes;
(b) Calculate the hidden-layer output matrix $\mathbf{H}$
(c) calculate the output weight β for the hidden nodes

$$\beta = \frac{\langle E, \mathbf{H}_L \rangle}{\| \mathbf{H}_L \|^2} \tag{1}$$

(d) calculate the residual error :$E = E - \beta_L \cdot \mathbf{H}_L$
Endwhile

3 Parameters Identification in ELM

Theorem 1 *[6][pp3462] Given an SLFNs with any nonconstant piecewise continuous hidden nodes $H(\boldsymbol{w}, \mathbb{b})$, if span$\{H(\boldsymbol{w}, \mathbb{b}) : (\boldsymbol{w}, \mathbb{b}) \in C^d \times C\}$ is dense in $\mathbb{R}^2$, for any continuous target function f and any randomly generated function sequence $\{\boldsymbol{H}_L\}$ and any positive integer k, $\lim_{L\to\infty} \| f - f_L^* \| = 0$ holds with probability one if*

$$\beta_L^* = \frac{\langle E_{L-1}^*, \boldsymbol{H}_L^* \rangle}{\| \boldsymbol{H}_L^* \|^2} \tag{2}$$

where $f_L^* = \sum_{i=1}^{L} \beta_i^* \boldsymbol{H}_i^*$, $E_L = f - f_L^*$, $\langle E, \boldsymbol{H} \rangle$ is the inner product and $\boldsymbol{H}_L^* = \{\boldsymbol{H}_i \mid min_{(L-1)k+1 \le i \le Lk} \| (f - f_{L-1}^*) - \beta_L H_i \|\}$.

The proof is shown in [6]. It can be noticed that although hidden nodes can be added randomly, some newly added hidden nodes may make residual error reduce less and some newly added hidden nodes may make residual error reduce more. Given a function sequence $\boldsymbol{H}_L$ which is randomly generated based on any continuous sampling distribution probability, for fixed k we can choose an element of the kth segment $\{\boldsymbol{H}_{(L-1)k+1}, \cdots, \boldsymbol{H}_{Lk}\}$ as $\boldsymbol{H}_L^*$ which minimizes $\| (f - f_{L-1}^*) - \beta_L \boldsymbol{H}_i \|$. In fact, because $\boldsymbol{H}_L = \boldsymbol{H}(\mathbf{w}_L, b_L)$, we can rewrite E_L in Theorem 1 as

$$E_L = f - (f_{L-1}^* + \beta_L^* \cdot \mathbf{H}_L^*) = f - f_{L-1}^* - \beta_L^* \cdot \mathbf{H}_L^*(\mathbf{w}_L^*, \mathbb{b}_L^*) \tag{3}$$

Consider (2), we obtain

$$E_L = f - f_{L-1}^* - \beta_L(\mathbf{w}_L^*, \mathbb{b}_L^*) \cdot \mathbf{H}_L(\mathbf{w}_L^*, \mathbb{b}_L^*) \tag{4}$$

Equation (4) indicates that if we want to minimize residual error E_L, we must find the optimum hidden node parameter $(\mathbf{w}_L^*, \mathbb{b}_L^*)$ at each learning step. By considering (4), we rewrite Theorem 1 as follow.

Theorem 2 *Given an SLFNs with any nonconstant piecewise continuous hidden nodes* $\boldsymbol{H}(\mathbb{Y})$, $\mathbb{Y} = (w, \mathbb{b})$, *for any continuous target function* f, *and any positive integer* k, $\lim_{L\to\infty} \parallel f - f_L^* \parallel = 0$ *holds with probability one if*

$$\beta_L^* = \frac{E_L \cdot \boldsymbol{H}_L^T(\mathbb{Y}_L^*)}{\boldsymbol{H}_L(\mathbb{Y}_L^*) \cdot \boldsymbol{H}_L^T(\mathbb{Y}_L^*)} \tag{5}$$

where $\mathbb{Y}_L^* = \{\mathbb{Y}_i \mid min_{(L-1)k+1 \le i \le Lk} \parallel (f - f_{L-1}^*) - \beta_L \boldsymbol{H}_L(\mathbb{Y}_i) \parallel\}$.

Proof The validity of the theorem is obvious according to Theorem 1. Further more, similar to [3], a consistent estimated of the weight β_L based on the training set is

$$\beta_L = \frac{E_L \cdot \mathbf{H}(\mathbb{Y})^T}{\mathbf{H}(\mathbb{Y}) \cdot \mathbf{H}(\mathbb{Y})^T} \tag{6}$$

□

Theorem 2 shows there are some input weights $(\mathbf{w}_L^*)$ and biases $(\mathbb{b}_L^*)$ at each learning step which make residual error reduce more quickly, i.e., $(\mathbb{Y}_L^* = \{\mathbb{Y}_i \mid min_{(L-1)k+1 \le i \le Lk} \parallel (f - f_{L-1}^*) - \beta_L(\mathbb{Y}_i)\, H_L(\mathbb{Y}_i) \parallel\}.)$. Finding these parameters, if we can, will greatly improve the learning effectiveness. Thus, the identification of $\mathbb{Y}^* = (\mathbf{w}^*, \mathbb{b}^*)$ in ELM is a kind of optimization process.

4 PC-ELM

The motivation for the work in this section comes from Theorem 2 that there are some input weights (w_L^*) and biases $(\mathbb{b}_L^*)$ at each learning step which make residual error reduce more. Therefore, our aim is to propose a simple improved implementation of ELM in order to find $\mathbb{Y}_L^*$, meanwhile, to achieve a more compact network architecture. In this section, we propose a parallel chaos search based incremental extreme learning machine(PC-ELM) in which the best input weights (w^*) and biases $(\mathbb{b}^*)$ at each learning step can be found based on proposed PCOA.

4.1 Conventional Chaos Optimization Algorithm

Chaos has properties of ergodicity, stochasticity, and the chaotic trajectories of the maps are useful to probe in wide ranges of the searching domain without being trapped into the local optimal. The chaos optimization algorithm (COA) is a novel optimization method developed by Li and Jiang [10]. The basic process of COA generally includes two major steps. Firstly, define a chaotic sequences generator based on the logistic map. Generate a sequence of the chaotic points and map it to a sequence of decision points in the original decision space (also called the first carrier wave). Then, calculate the objective functions with respect to the generated decision points and choose the point with the minimum objective function as the current

optimum. Secondly, the current optimum is assumed to be close to the global optimum after certain iterations, and it is viewed as the center with a little chaotic perturbation(also called the second carrier wave), and the global optimum is obtained through fine search. Repeat the above the two steps until some specified convergence criterion is satisfied, and the global optimum is obtained.

In the first carrier wave of COA(also called global search), amplify the ergodic areas of chaotic variables to the variance ranges of optimization variables as following:

$$x^{(t)} = (a - b) \cdot d \cdot \gamma \tag{7}$$

where a is upper bound and b is lower bound,t is iteration number, γ is the chaos variable vector and $\gamma \in [0, 1]$, d is amplification gain.

Although chaotic states have ergodicity in a certain area, it may take a long time to get some states. If the global optimum just appears at those states, it will take a long search time to get the optimum. The second carrier wave of COA is adopted to shorten the search time. In the conventional COA, the second carrier wave is usually defined as

$$x^{(t)} = x^{best} + \alpha(t) \cdot \gamma \tag{8}$$

where x^{best} is the so-far best decision solution vector and variable α is decreasing with the number of iterations t increased.

In this paper, optimal parameters of hidden nodes are searched by conventional COA and a method of chaos search based ELM (referred as to C-ELM) is proposed first. But unfortunately, although computational time of C-ELM is less than EI-ELM, performance of C-ELM is not much better than that of EI-ELM. This can also be verified from Figs. 8, 9, 10, 11, 12, and 13 in Sect. 5 which show average testing accuracy of EI-ELM and C-ELM for some classification data. There are two reasons. Firstly, further simulations have shown that conventional COA only effect in low dimension. But we notice in ELM, dimension of hodden node parameters ($\mathbb{Y}$) at each learning step is always high. Secondly, conventional COA is easy to fall into local minima especially in high-dimension optimization problem. In (8), the parameter $\alpha(t)$ determines the resolution with which regions are to be searched between the present position and the target position. If $\alpha(t)$ is too high, $x(t)$ might fly past good solutions. If $\alpha(t)$ is too small, $x(t)$ may not sufficiently explore beyond local solutions. Therefore, conventional COA should be adjusted to solve these two problems.

4.2 **Parallel Chaos Algorithm (PCOA)**

Recently, researchers are now trying to improve COA's efficiency by means of altering the model, configuration, control strategy, search style, et al., and thus various COA's have been proposed [11–13]. To overcome the two problems which mentioned above, in this section, we modify some processes and equations in second carrier wave of conventional COA in order to find optimal parameters of hidden node.

Step 1 First carrier wave of PCOA

Step 1.1: Set $t = 0$ and $F_j^* = +\infty$

step 1.2: Initialize the total number of the chaos search t_{max}, the number of the first chaos search m_1

Step 1.3: Map chaos variables r_{ij}^t into the variance range of the optimization variables by the following equation:

$$x_{ij}^t = a + (b-a) \cdot r_{ij}^t \tag{9}$$

$$x^{(t)} = \begin{bmatrix} x_{11}^{(t)} & x_{21}^{(t)} & \cdots & x_{n1}^{(t)} \\ x_{12}^{(t)} & x_{22}^{(t)} & \cdots & x_{n2}^{(t)} \\ \cdots & \cdots & \cdots & \cdots \\ x_{1p}^{(t)} & x_{2p}^{(t)} & \cdots & x_{np}^{(t)} \end{bmatrix} \tag{10}$$

where a is lower bound, b is upper bound, $i = 1, 2, \cdots, n$ represents each optimization variable, $j = 1, 2, \cdots, p$ represents each optimization variable mapped by multiple p chaos variables. Because the input weights w in neural network are ranged from $[-1, 1]$, we set $a = -1$ and $b = 1$ for input weights. Whereas, the biases $\mathbb{b}$ in network are ranged from [0, 1], we set $a = 0$ and $b = 1$ for bias.

Step 1.4: If $F_j(x^{(t)}) \leq F_j^*$ then $x^{best} = x_j^{(t)}$ and $F_j^* = F_j(x^{(t)})$; If $F_j(x^{(t)}) > F_j^*$ and x^{best}, F_j^* is maintained. This means that the search result is the optimal value of parallel multiple chaos variables.

Step 1.5: Generate next values of chaos variables by a chaotic map function (M):

$$r_{ij}^{(t+1)} = M(r_{ij}^{(t)}) \tag{11}$$

Step 1.6. If $t < m_1, t = t + 1$ and go to Step 1.3, else stop the first chaos search process.

As we know, the second carrier wave of conventional COA can affect the search time and the search precision. In the COA, the second carrier wave is shown in Eq. (8). This is to guarantee that the search process was performed in the small neighbor domain of the sub-optimal solutions, and to shorten the search time. But, this searching strategy is hard to use in ELM to find optimal solution ($\mathbb{Y}$). As for $x^{(t)} = x^{best} + \alpha(t) \cdot \gamma$, more times the iterations perform, the smaller the value of $\alpha(t)$ becomes and $\alpha(t) \cdot \gamma$ gets closer to zero. Obviously, when x^{best} fall into local minimum point, COA may lose the probability of finding global optimum. Thus we modified the second wave. And it is shown in Step 2.

Step 2: Chaos search by using the second carrier wave of proposed PCOA.

Step 2.1. Generation next values of chaos variables by the chaotic map function

$$r_{ij}^{(t+1)} = M(r_{ij}^{(t)}) \tag{12}$$

Step 2.2: $t = t + 1$

Step 2.3:

$$x_{ij}^{(t)} = x^{best} + \alpha(t) \cdot (\gamma_{ij}^{(t)} - 0.5), \qquad (j = 1, 2, \cdots, ceil(p/2)) \tag{13}$$

$$x_{ij}^{(t)} = a + \gamma_{ij}^{(t)} \cdot (b-a), \qquad (j = ceil(p/2) + 1, \cdots, p) \tag{14}$$

$$x^{(t)} = \begin{bmatrix} x_{11}^{(t)} & x_{21}^{(t)} & \cdots & x_{n1}^{(t)} \\ x_{12}^{(t)} & x_{22}^{(t)} & \cdots & x_{n2}^{(t)} \\ \cdots & \cdots & \cdots & \cdots \\ x_{1p}^{(t)} & x_{2p}^{(t)} & \cdots & x_{np}^{(t)} \end{bmatrix} \tag{15}$$

We can see that in 13, $x_{ij}^{(t)}$ is a randomly chosen element of the variables from the search space. This ensures that not all of $x^{(t)}$ in 14 are generated with small ergodic ranges around x^{best} when x^{best} fall into local minima.

Step 2.4 If $F_j(x^{(t)}) \leq F_j^*$ then $x^{best} = x_j^{(t)}$ and $F_j^* = F_j(x^{(t)})$; If $F_j(x^{(t)}) > F_j^*$ and x^{best}, F_j^* is maintained.

Step 2.5 if $t < (t_{max} - m_1)$ and go to step 2.1

4.3 Pseudo Code of PC-ELM

For parameters of hidden node ($\mathbb{Y}$), its identification problem can be treated as an optimization problem. In this paper, the optimal hidden-node parameters $\mathbb{Y}^*$ are selected by PCOA in order to minimize the residual network error between the target function and the network output. According to I-ELM [3], it could not be calculated since the exact functional form of f is unknown. Similar to [3], a consistent estimated of the weight β_L based on the training set is:

$$\beta_L = \frac{E_L \cdot \mathbf{H}(\mathbb{Y})^T}{\mathbf{H}(\mathbb{Y}) \cdot \mathbf{H}(\mathbb{Y})^T} \tag{16}$$

The pseudo code of proposed PC-ELM is shown in Algorithm 2.

4.4 Convergence Analysis

Lemma 1 *[5] Given any small positive value $\varepsilon > 0$ and activation function $\boldsymbol{H}$ which is infinitely differentiable in any interval, there exists q such that for q arbitrary distinct samples $(\mathscr{X}, \mathscr{T})$, where $\mathscr{X} \in \mathbb{R}^m$ and $\mathscr{T} \in \mathbb{R}^n$, for any $\boldsymbol{w}$ and b randomly chosen from any intervals of $\mathbb{R}^n$ and $\mathbb{R}$, respectively, according to any continuous probability distribution, then with probability one, $\| H_q\beta_q - \mathscr{T} \| < \varepsilon$.*

Remark 1 The proof is shown in [5] and based on this Lemma, the original batch learning mode extreme learning machine (ELM) [5] and their online sequential learning mode ELM (OS-ELM) [9] work for SLFNs with fixed network architectures.

Lemma 2 *[3] Given any bound nonconstant piecewise continuous function function for additive nodes or RBF nodes, for any continuous target function f and any randomly generated function sequence $\boldsymbol{H}_L$, $\lim_{L\to\infty} \| f - f_L \| = 0$ holds with probability one if*

$$\beta_L = \frac{\langle E_{L-1}, \boldsymbol{H}_L \rangle}{\| \boldsymbol{H}_L \|^2} \tag{20}$$

Remark 2 Different from Lemma 1, Lemma 2 is validity for the case where all the output weights of the existing hidden nodes are frozen when a new hidden node is nodded. Based on this lemma, Huang et al. proposed a method referred as incremental extreme learning machine (I-ELM). Unlike ELM, OS-ELM with a fixed network architecture, these incremental methods including EM-ELM, CI-ELM are proposed by Huang et al. with a incremental network architecture.

Theorem 3 *For a given set of distinct training samples $(\mathscr{X}, \mathscr{T})$, given an arbitrary positive value ε, there exists a positive integer q such that $\|E(w_q, b_q)\| = min\|\boldsymbol{H}_q\beta_q - \mathscr{T}\| < \epsilon$ using proposed PC-ELM.*

Proof (a) When $\mathbb{L} \neq L_{max}$, this is to say hidden nodes are added to hidden layer one by one or chunk by chunk.

Set total number of chaos search step k, the number of hidden nodes L. Set $L = 0$ and chaos search $k = 0$, $k < K$, the corresponding output error $||E_L(\mathbf{H}_i)|| > \varepsilon$, $L = 0, k = 0$. We

Algorithm 2 Pseudo-code for PC-ELM

Given a training set $(\mathscr{X}, \mathscr{T}), \mathscr{X} \in \mathbb{R}^n, \mathscr{T} \in \mathbb{R}$, an activation function $f : \mathbb{R} \to \mathbb{R}$, and the number of hidden nodes L, hidden node output function $H(\mathbf{Y}), \mathbf{Y} = [w, \mathbb{b}]$, maximum number L_{max} of hidden nodes, expected accuracy η, m_1, t, t_{max} have been defined in section 4.2.

Step (1) **Initialization**: Let $L = 0, t = 0$ and residual error $E_0 = \mathscr{T}$.

Step (2) **Learning step**:

while do $L < L_{max}, |E| > \eta$

$$L = L + \mathbb{L} \tag{17}$$

where $\mathbb{L} = 1, \cdots, L_{max}$. If $\mathbb{L} = 1, \cdots, L_{max} - 1$, it means the hidden nodes are added to hidden layer one by one or group by group. Thus PC-ELM is developed for SLFNs with increased network architecture. If $\mathbb{L} = L_{max}$, PC-ELM is developed for SLFNs with fixed network architecture, meanwhile, its learning mode is as the same as that of ELM.

while do $t < t_{max}$

$t = t + 1$

for do $j = 1 : p$

if then$t < m_1$

generate $\mathbb{L}$ hidden nodes parameters $\mathbf{Y}^j_{Ft}(F = L + 1 - \mathbb{L}, \cdots, L - 1, L; F > 0)$ by (9)-(10)

end if

if then$m_1 < t < t_{max}$

generate $\mathbb{L}$ hidden nodes parameters $\mathbf{Y}^j_{Ft}(F = L + 1 - \mathbb{L}, \cdots, L - 1, L; F > 0)$ by (13)-(15)

end if

(a) calculate the output weight β^j_{Ft} for the new hidden nodes:

$$\beta^j_{Ft} = \frac{E_{((L-\mathbb{L}))} \cdot (H(\mathbf{Y}^j_{Ft})^T}{H(\mathbf{Y}^j_{Ft}) \cdot (H(\mathbf{Y}^j_{Ft}))^T} \tag{18}$$

(b) calculate the residual error after adding the new hidden node $\mathbb{L}$:

$$E^j_{Lt} = E_{((L-\mathbb{L})t)} - \beta^j_{Ft} \cdot H(\mathbf{Y}^j_{Ft}) \tag{19}$$

end for

(c) Let $j^* = \{j | min_{1 \le j \le p} \parallel E^j_{Lt} \parallel\}, E_{Lt} = E^{j^*}_{Lt}, \quad \mathbf{Y}_{Ft} = \mathbf{Y}^{j^*}_{Ft}, \beta_{Ft} = \beta^{j^*}_{Ft}$

end while

(d) Let $t^* = \{t | min_{1 \le t \le t_{max}} \parallel E_{Lt} \parallel\}, E_L = E_{Lt^*}, \mathbf{Y}_F = \mathbf{Y}_{Ft^*}, \beta_F = \beta_{Ft^*}$

end while

keep on adding chaos search steps till $k = K$ and we get the new corresponding output error $||E_L(\mathbf{H}_K)|| \le ||E_L(\mathbf{H}_0)||, L = 0$. Then, set $L = 1$ and chaos search $k = 0$. According to [1], we can get the conclusion that the new corresponding output error $||E_1(\mathbf{H}_0)|| \le ||E_0(\mathbf{H}_K)||$. Thus, when $k = K$ and $L = N$, we get:$||E_N(\mathbf{H}_K)|| \le |E_N(\mathbf{H}_0)|| \le \cdots \le ||E_1(\mathbf{H}_0)|| \le ||E_0(\mathbf{H}_K)|| \le ||E_0(\mathbf{H}_0)||$. On the other hand, by Huang et al. [5], if $\sum_{L=1}^{q} L > N$, $\mathbf{H}_q$ is invertible with probability one and thus $E_q(\mathbf{H}_0) \ge \cdots \ge E_q(\mathbf{H}_K) = 0$. Therefore, there exist $q < N$ and $E(w_q, b_q) = min||\mathbf{H}_q \beta_q - T|| < \epsilon$.

(b) When $\mathbb{L} = L_{max}$, it means PC-ELM works for SLFNs with a fixed network architecture.

Suppose $L_{max} = q_1 (q_1 = 1, 2, \cdots)$, chaos search step $k = 1$, we get $\mathbb{L} = L_{max} = q_1 < N$, and the corresponding output error $||E_{q_1}(\mathbf{H}_k)|| > \varepsilon, k = 1$. we keep adding chaos search steps till $k = K$ and we get the new corresponding output error $||E_{q_1}(\mathbf{H}_K)|| \le$

$||E_{q_1}(\mathbf{H}_{K-1})|| \leq \cdots \leq ||E_{q_1}(\mathbf{H}_1)||$. According to Lemma 1, there exist q_2 and $q_1 < q_2 < N$ that make corresponding $||E_{q_2}(\mathbf{H}_1)|| \leq \epsilon$, then we keep adding chaos search steps till $k = K$ and we get $\epsilon \leq ||E_{q_2}(\mathbf{H}_K)|| \leq ||E_{q_2}(\mathbf{H}_{K-1})|| \leq \cdots \leq ||E_{q_2}(\mathbf{H}_1)||$. Therefore, there exist $q < N$ and $E(w_q, b_q) = min||H_q\beta_q - T|| < \epsilon$. □

Remark 3 Different from Lemma's 1 and 2, Theorem 3 proves that by PC-ELM, SLFNs with q hidden nodes can exactly learn N distinct observations both with increased network architecture (hidden nodes are added to hidden layer one by one, group by group) and with fixed network architecture. PC-ELM can obtain more compact network architecture.

5 Performance Verification

To examine the performance of our proposed algorithm (PC-ELM), in this section, we test them on some benchmark regression and classification problems from UCI database. The simulations are conducted in Matlab 2009a running on the same Windows 7 machine with at 2 GB of memory and i5-430 (2.33 GHZ) processor. Neural networks are tested in ELM, EM-ELM, I-ELM, EI-ELM, OP-ELM, proposed PC-ELM and conventional COA base ELM (C-ELM). And it should be noted that C-ELM is a specific case of PC-ELM when p equals to 1 and replacing Eqs. (13–15) with Eq. (8). For these methods, the sigmoid additive activation function are chosen as the output function of the hidden nodes: $H(\mathbf{w}, \mathbb{b}, \mathscr{X}) = 1/(1 + exp(-(\mathbf{w} \cdot \mathscr{X} + \mathbb{b})))$; the chaotic map function are chosen as: $x^{(t)} = sin(2/x^{(t-1)})$. The input data are normalized into $[-1, 1]$ while the output data for regression are normalized into the range $[0, 1]$. For comparing the generalization performance of these methods (PC-ELM, C-ELM and EI-ELM), these methods should be run in the same total searching steps. Here we set: for C-ELM and EI-ELM, the number of total searching steps N is defined as $N = k$; for PC-ELM, the number of searching steps N is defined as $N = p \times t$. If $k = p \times t$, these three methods are run under the same condition.

The performance comparison has been conducted in some real benchmark regression and classification problems shown in Table 1 which includes nine regression problems (i.e., Concrete, Auto MPG, Machine CPU, Bank, Servo, Friedman, Puma, California Housing and Parkinsons) and ten classification problems (i.e., Sgement, Diabetes, Iris, Heart, breast cancer, Landsat satellite image, Hill valley, Movement libras, Semeion handwritten digit, Musk I and Waveform Database II). The experimental results between PC-ELM and some other ELM algorithms on regression and classification cases are given in Tables 2–6. In Tables 2–6, the close results obtained by different algorithms are underlined and the apparent better results are shown in boldface. In Sect. 5.1, we compare the generalization performance of PC-ELM with some incremental methods including I-ELM, C-ELM and EI-ELM. In Sect. 5.2, we compare the generalization performance of PC-ELM with some other methods including ELM, EM-ELM, OP-ELM and C-ELM. All these results in this section are obtained over 50 trials for all cases.

5.1 Performance Comparison of Incremental ELM Methods

In this subsection, the initial SLFNs are given 1 hidden nodes and then new random hidden nodes will be added one by one. The experimental results between PC-ELM and some other incremental algorithms (I-ELM, EI-ELM, C-ELM) on regression and classification cases are given in Tables 2, 3 and Figs. 1, 2, 3, 4, 5, 6, 7, and 8.

Table 1 Specification of 15 benchmark data sets

Datasets	# Attri	Class	Type	# Train	# Test
Concrete	9	–	Regression	500	530
Auto MPG	8	–	Regression	200	192
Machine CPU	6	–	Regression	100	109
Bank	8	–	Regression	3,000	1,499
Servo	4	–	Regression	100	67
Friedman	11	–	Regression	20,768	20,000
Puma	9	–	Regression	4,500	3,692
Parkinsons	18	–	Regression	3,000	2,875
California housing	8	–	Regression	8,000	12,460
Segment	19	7	Classification	1,500	810
Diabetes	8	2	Classification	576	192
Iris	4	3	Classification	80	70
Heart	22	2	Classification	80	187
Breast cancer	30	2	Classification	369	200
Waveform II	40	3	Classification	2,500	2,500
Musk I	168	2	Classification	276	200
Landsat satellite image	36	7	Classification	3,217	3,218
Hill valley	101	2	Classification	606	606
Movement libras	90	16	Classification	180	180

5.1.1 Performance Comparison of Incremental ELM Methods with the Same Number of Hidden Nodes

We first compare the generalization performance of I-ELM, EI-ELM, C-ELM and PC-ELM with the same hidden-node numbers. In order to test these methods under the same condition, we set $k = p \times t$. Figs. 1, 2, 3, and 4 show that the performance results of these methods with the same number of searching steps N will become better when the number of hidden nodes increases.

Datasets Concrete, Machine CPU, Auto MPG, Fried, Puma, and Parkinsons are belong to regression problems. Table 2 shows the performance of PC-ELM, C-ELM, I-ELM and EI-ELM with different hidden-node numbers, respectively. For Parkinsons problem, in Fig. 1, the PC-ELM provide a higher test RMSE rate than other methods thanks to its better regression capability. It reaches approximately 0.11 when around 20 hidden nodes are added, whereas other methods offer about 0.15. Seen from Table 2 and Fig. 2, for all regression cases, the testing RMSE of PC-ELM are generally much smaller than that of I-ELM, C-ELM and EI-ELM. It shows that PC-ELM achieves faster convergence rate than I-ELM, C-ELM and EI-ELM.

5.1.2 Sensitivity of Parameters P and t for PC-ELM

Figures 5, 6, 7, and 8 shows that the generalization performance of EI-ELM, C-ELM and PC-ELM with 50 sigmoid hidden nodes will become better when the number of searching steps N increase. As observed from Figs. 5, 6, 7, and 8, the average rate of testing accuracy

Table 2 Performance comparison for benchmark problems

Datasets	EI-ELM				C-ELM				I-ELM			PC-ELM				
	Mean	Time (s)	# Nodes	k	Mean	Time (s)	# Nodes	k	Mean	Time (s)	# Nodes	Mean	Time (s)	# Nodes	P	t
Concrete	0.0630	0.4593	20	20	0.0586	0.3943	20	20	0.1884	0.0643	20	**0.0501**	0.4010	20	2	10
Auto MPG	0.1048	0.2907	10	50	0.1045	0.2072	10	50	0.1830	0.0775	10	0.1042	0.2071	10	5	10
Machine CPU	0.0566	0.2884	20	20	0.0530	0.2197	20	20	0.0903	0.0081	20	**0.0486**	0.2009	20	5	4
Fried	0.0869	15.9375	20	100	0.0897	13.5250	20	100	0.1014	0.0206	20	0.0886	13.9422	20	5	20
Puma	0.1896	1.9715	100	20	0.1902	2.0952	100	20	0.2056	0.1962	100	0.1891	1.6189	100	2	10
Parkinsons	0.1281	6.3905	200	40	0.1176	4.9827	200	40	0.1349	0.3526	200	**0.1115**	4.1016	200	4	10
Segment	86.79 %	56.14	50	500	87.11 %	29.9302	50	500	73.19 %	0.1507	50	**89.34 %**	31.07	50	5	100
Satellite	77.96 %	107.9995	50	500	82.39 %	42.3260	50	500	73.06 %	0.7453	50	**89.34 %**	31.07	50	5	100
Diabetes	74.64 %	22.6905	50	500	75.67 %	17.5442	50	500	65.54 %	0.4740	50	**76.63 %**	16.5308	50	5	100
Iris	86.00 %	6.5630	50	500	89.99 %	6.9872	50	500	78.57 %	0.0419	50	**91.43 %**	5.9126	50	5	100
Heart	80.70 %	7.1620	50	500	80.99 %	6.8453	50	500	78.77%	0.0281	50	81.23 %	6.6800	50	5	100
Waveform	84.15 %	77.3484	50	500	84.65 %	50.1095	50	500	73.88 %	0.2465	50	84.55 %	51.7299	50	5	100
Hill valley	57.15 %	21.9462	200	500	59.91 %	76.4197	200	500	49.52 %	0.4819	200	**60.01 %**	79.4293	200	5	100
Musk I	85.03 %	92.1654	200	500	88.64 %	80.2934	200	500	77.35 %	0.4964	200	**89.59 %**	73.7254	200	5	100
Movement	61.78 %	114.6710	200	500	62.69 %	60.6411	200	500	49.35 %	0.3513	200	**67.15 %**	57.4602	200	5	100

Accuracy testing accuracy, *time* training time, *mean* testing RMSE for regression problems and mean testing accuracy for classification problems, # *node* number of hidden nodes

Table 3 Performance comparison with 10 hidden nodes on classification application

Datasets	EI-ELM		EI-ELM		C-ELM		PC-ELM		I-ELM	
Classification	$k = 1000$		$k = 100$		$K = 100$		$t = 100$, p = 10			
	Accuracy (%)	Time (s)	Accuracy (%)	Time (s)	Accuracy (%)	Time (s)	accuracy (%)	Time (s)	Accuracy (%)	Time (s)
Segment	75.86	45.3573	70.98	1.4742	81.81	1.2854	**86.51**	10.1229	61.20	0.0842
Landsat satellite image	74.08	35.3498	73.03	1.7397	73.59	1.0216	**76.89**	9.5317	59.82	0.0905
Diabetes	70.78	17.8713	67.79	0.9173	66.67	0.6871	**72.05**	6.0681	64.51	0.0125
Iris	76.11	3.7688	75.08	0.2683	79.77	0.2249	**86.56**	2.1450	69.17	0.0069
Heart	79.20	3.5599	79.09	0.2792	78.07	0.3994	**80.63**	2.2995	75.93	0.0109
Waveform database II	77.02	92.8362	76.58	2.7131	75.95	2.0265	**80.84**	17.7529	57.19%	0.0759

New hidden nodes are added one by one
Accuracy testing accuracy, *time* training time

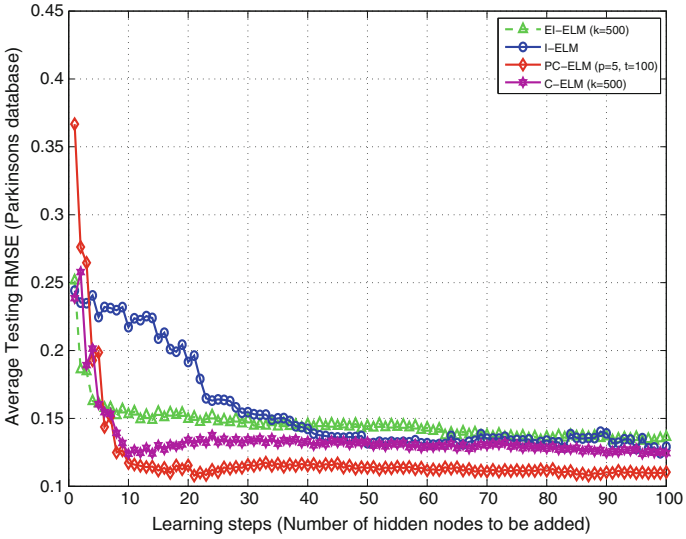

Fig. 1 Average test RMSE of different methods in Parkinsons case

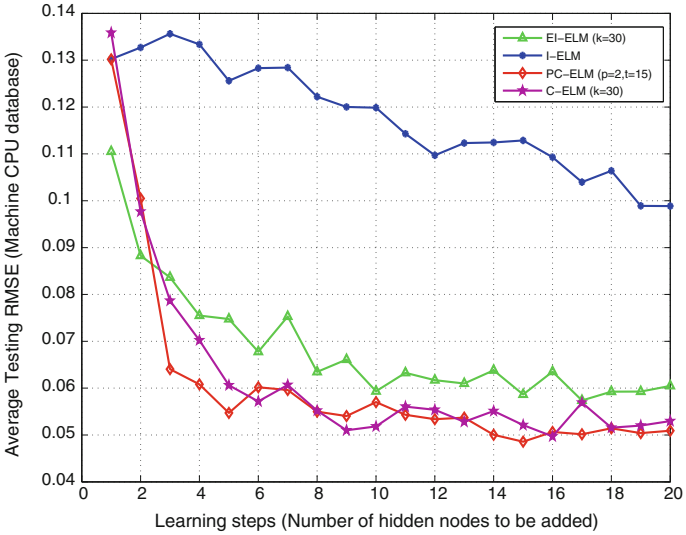

Fig. 2 Average test RMSE of different methods in machine CPU case

obtained by PC-ELM are generally better than those obtained by C-ELM and EI-ELM when both the number of hidden nodes and the number of searching steps are the same. Further more, from Figs. 5, 6, 7, and 8, we find that the testing results obtained by PC-ELM with $P_1 = 2$, $P_2 = 3$, $P_3 = 4$ and $P_4 = 5$ are approximately equal. It implies that as long as PC-ELM has the same total searching steps N, i.e., $(P_1 \times t_1 = P_2 \times t_2 = P_3 \times t_3 = \cdots = N)$, the generalization performance of PC-ELM does not change whatever parameter P or t changes. In all, the generalization performance obtained by PC-ELM can be generally better than those

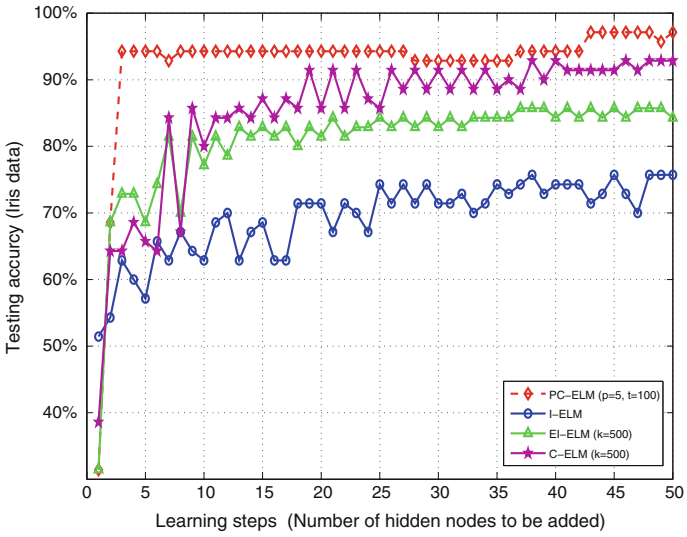

Fig. 3 Average testing accuracy of different algorithms in iris case

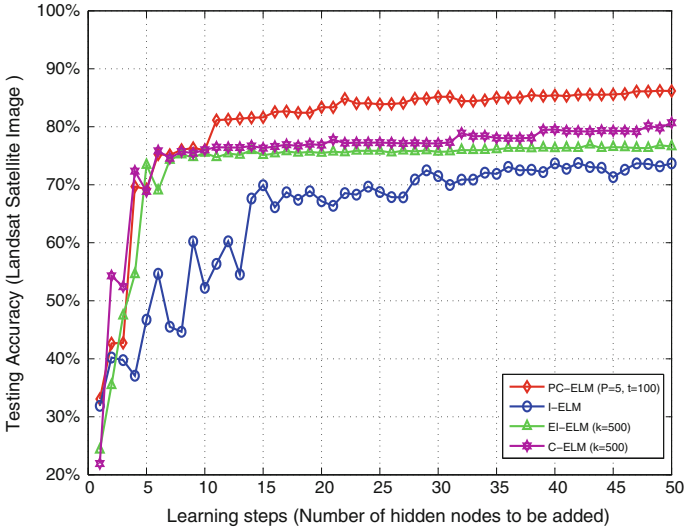

Fig. 4 Average testing accuracy of different algorithms in landsat satellite image case

results obtained by EI-ELM and C-ELM when $N = k = p \times t$). The above observations are true for the rest cases as well.

Segment, Satellite, Diabetes, Iris, Heart, Waveform, Hill Valley, Musk I and Movement are belong to classification problem. For these problems, also in Table 2, the results in term of avg.test accuracy of the PC-ELM are much better than those of the other methods. Also, in Figs. 3 and 4, the PC-ELM displays a higher test accuracy rate than other methods thanks to its better classification ability. In Fig. 3, the avg.test accuracy of PC-ELM reaches approximately 94 % with 10 hidden nodes, whereas other methods offer about 82 % and 70 %. For other classification cases, the performance of the PC-ELM is also better than that of the other

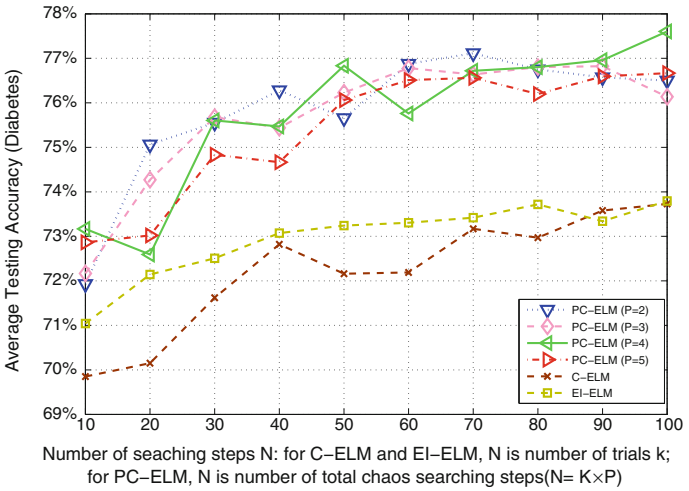

Fig. 5 Average testing accuracy of different algorithms in diabetes case

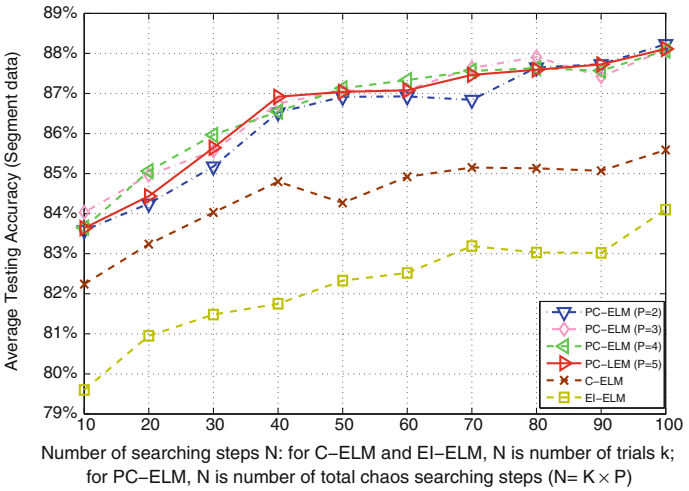

Fig. 6 Testing accuracy of different selecting parameters of different methods in segment case

methods. In short, PC-ELM generally produces better performance for these benchmark problems in term of testing accuracy for classification and testing RMSE for regression.

5.1.3 Performance Comparison of Incremental ELM Methods with 10 Hidden Nodes

During our simulations, we also study the generalization performance of these methods with 10 hidden nodes. The comparison results are shown in Table 3. Here we set $t = 100$, $p = 10$ for PC-ELM and $K = 1000$ for EI-ELM. Table 3 demonstrates that PC-ELM with only a few hidden nodes can achieve much better generalization performance as compared to other incremental ELM methods. For example, Table 2 shows the testing accuracy of Waveform database II obtained by EI-ELM and PC-ELM is 84.15 and 84.55 % respectively, with 50

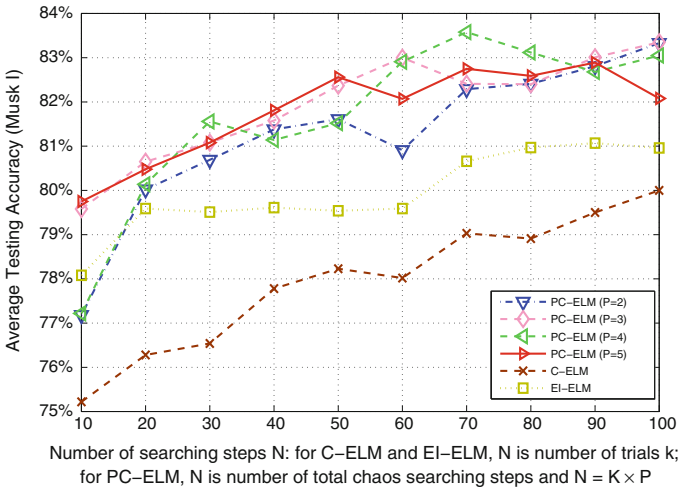

Fig. 7 Testing accuracy of different selecting parameters of different methods in musk I case

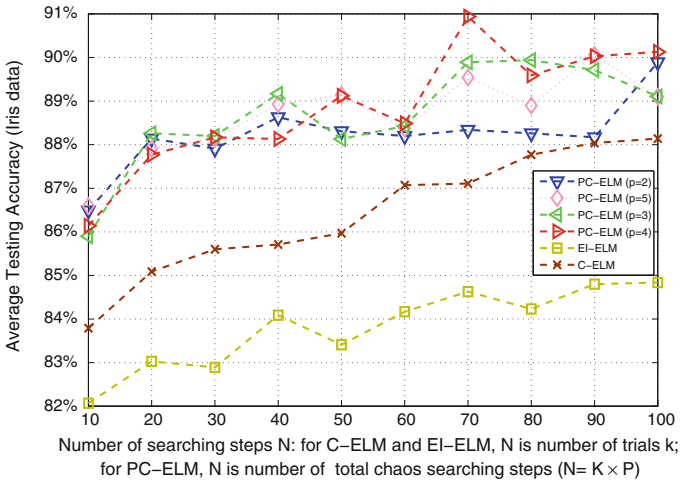

Fig. 8 Testing accuracy of different selecting parameters of different methods in iris case

hidden nodes are used. In Table 3, however, we notice the testing accuracy of this database obtained by EI-ELM and PC-ELM is 77.02 and 80.84 % respectively, even with the 10 hidden nodes.

5.2 Performance Comparison of Fixed ELM Methods

In this subsection, we study the generalization performance of ELM methods with fixed network architecture. These ELM methods are EM-ELM, ELM, OP-ELM, and our proposed PC-ELM. For PC-ELM, the initial SLFNs are given 0 hidden nodes and then L_{max} hidden nodes will be added to hidden layer. For EM-ELM, this algorithm can be seen as a specially fixed network architecture. Because the first learning step of EM-ELM is as the same as ELM (with fixed network architecture). In the second learning step, EM-ELM recalculates all the

Table 4 Performance comparison with 10 hidden nodes for benchmark problems

Datasets	EM-ELM			C-ELM				ELM			PC-ELM					OP-ELM		
	dev	RMSE	Time (s)	dev	RMSE	Time (s)	k	dev	RMSE	Time (s)	dev	RMSE	Time (s)	P	t	dev	RMSE	Time (s)
Bank	0.0217	0.1188	0.7145	0.0141	0.0821	0.7634	200	0.0517	0.1131	0.0761	0.0122	**0.0641**	0.6941	5	40	0.1287	0.4735	0.0333
Fried	0.0031	0.1161	0.4528	0.0015	0.0827	0.4210	200	0.0016	0.1156	0.0353	0.0010	**0.0823**	0.4053	5	40	0.0065	0.1232	1.6914
Servo	0.0262	0.1651	0.1884	0.0050	0.1290	0.1917	200	0.0310	0.1414	0.0206	0.0036	**0.1217**	0.1613	5	40	0.0440	0.1621	0.3785
California house	0.0320	0.1495	0.4593	0.0102	0.1421	0.3943	50	0.0294	0.1501	0.0643	0.0103	**0.1401**	0.4010	5	10	0.0402	0.1531	0.2587

dev testing standard deviation, *accuracy* testing RMSE for regression problems and mean testing accuracy for classification problems

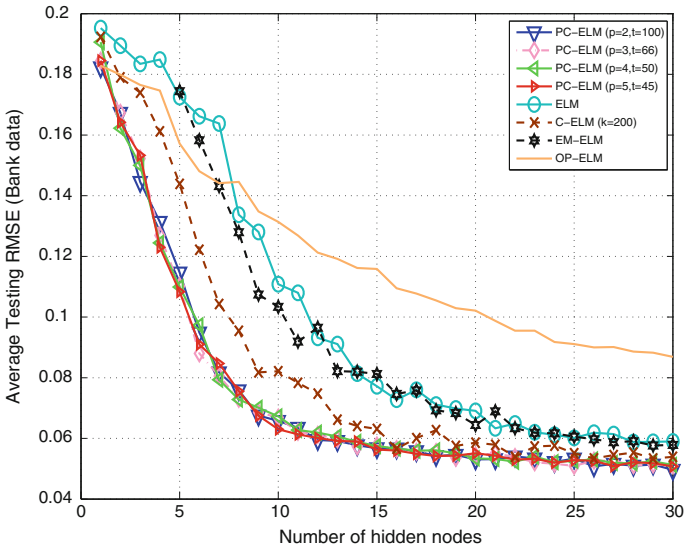

Fig. 9 Average testing accuracy of different algorithms in bank case

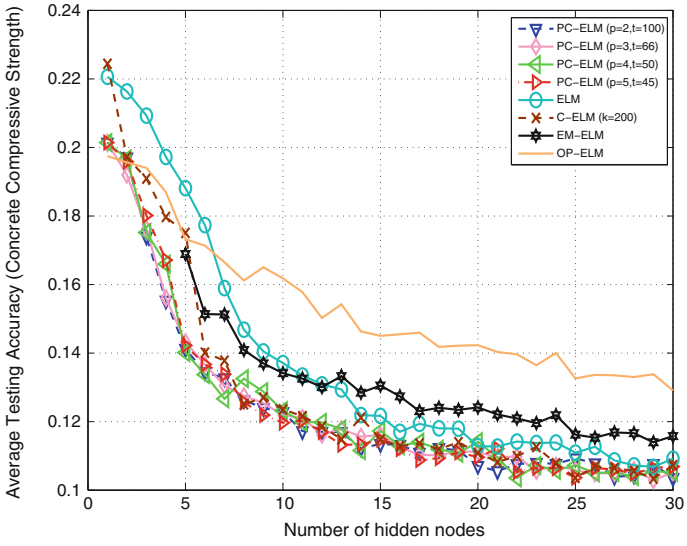

Fig. 10 Average testing accuracy of different algorithms in concrete case

output weights like ELM but hidden nodes is/are added one by one or group by group to hidden layer. For EM-ELM, in this subsection, the initial SLFNs are given 5 hidden nodes and then new hidden nodes will be added one by one. Thus, the experimental results between PC-ELM and other fixed ELM methods(ELM,C-ELM, EM-ELM) on regression and classification cases are given in Table 4 and Figs. 9, 10, 11, and 12.

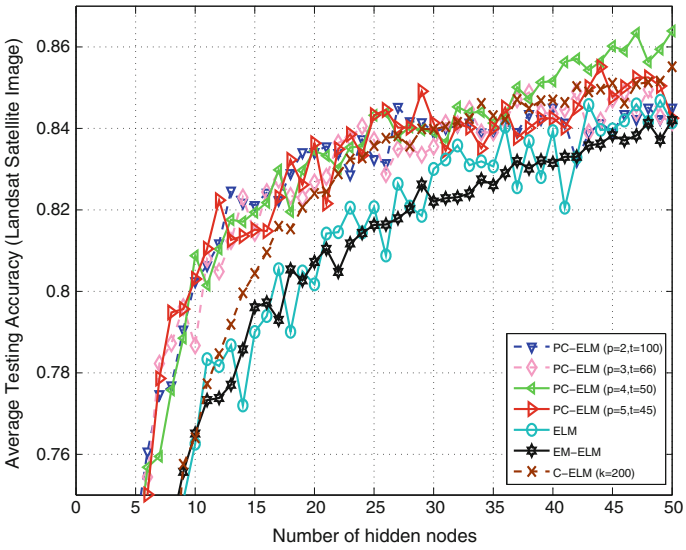

Fig. 11 Average testing accuracy of different algorithms in landsat satellite image case

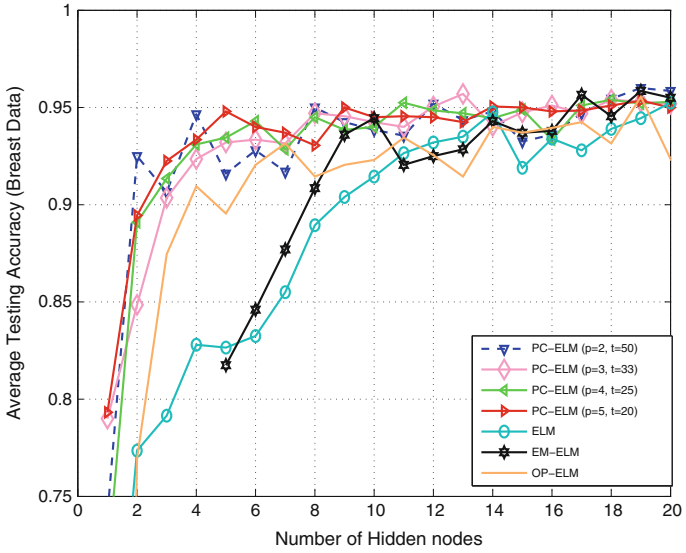

Fig. 12 Average testing accuracy of different algorithms in breast case

5.2.1 Performance Comparison of Fixed ELM Methods with the Same Number of Hidden Nodes

The performance comparison between PC-ELM and some other ELM algorithm (C-ELM, EM-ELM, ELM) on regression and classification cases is given in Figs. 9, 10, 11, and 12. These results show under the condition of the same number of hidden nodes, PC-ELM generally tends to provide more better generalization performance than C-ELM, EM-ELM and

Table 5 Number of hidden nodes between PC-ELM and other ELM methods on Regression application

Datasets (stop accuracy)	PC-ELM		C-ELM		ELM		EM-ELM		OP-ELM	
	$t = 50, p = 2$		$k = 100$		$K = 100$					
	# Nodes	Time (s)	# Nodes	Time (s)	# Nodes	Time (s)	# Nodes	Time (s)	# Nodes	Time (s)
Auto_MPG (0.05)	**39.5**	0.3714	45.7	0.6012	60.2	0.0574	69.7	0.0725	46.9	0.7217
Servo (0.080)	**19.4**	0.1231	23.6	0.2001	26.5	0.0156	26.1	0.0076	44.5	0.5276
Concrete (0.09)	**38.6**	1.1232	41.6	1.3012	55.2	0.0312	63.1	0.0121	60.4	2.8383
Fired (0.08)	**24.1**	10.1346	31.6	18.997	48.7	0.5814	48.1	0.3174	149.4	67.0784
Bank (0.055)	**16.6**	1.6125	21.3	2.2351	29.5	0.0390	30.2	0.0401	97.4	2.5781
California house (1.35)	**21.1**	0.9765	23.0	1.1236	32.1	0.1248	30.2	1.2948	38.3	1.3432
Breast (0.95 %)	**2.1**	0.2194	5.7	0.6012	5.9	0.0211	5.7	0.0201	6.3	0.1298
Waveform II (0.75 %)	**11.6**	2.4243	15.4	3.1467	25.2	0.0312	22.3	0.1810	31.9	0.7038
Landsat satellite image (0.84 %)	**22.8**	6.8094	31.6	1.8997	36.34	0.7436	37.1	0.4343	41.9	2.8930

All the methods stop training when the expected accuracy is reached

Accuracy training RMSE for regression problem and mean training accuracy for classification problem, # *node* number of hidden nodes

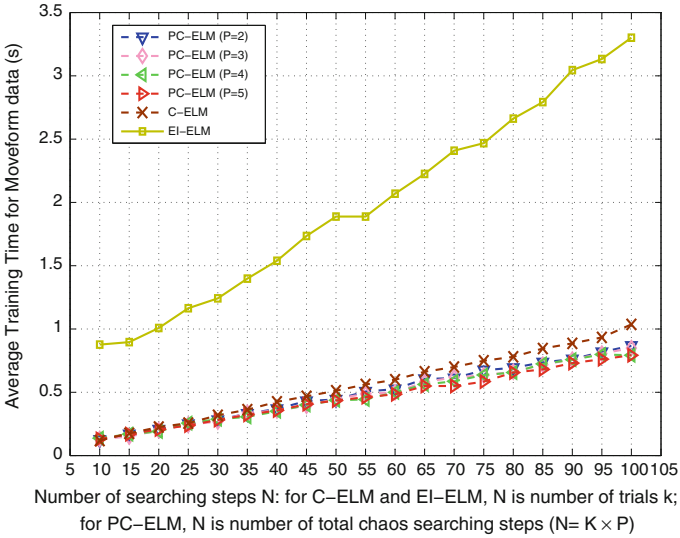

Fig. 13 Testing accuracy of different selecting parameters of different methods in waveform II case

ELM. Further more, we find that the average testing accuracy of PC-ELM with $P_1 = 2$, $P_2 = 3$, $P_3 = 4$ and $P_4 = 5$ is approximately equal. This indicates that as long as it has the same searching steps N, i.e., $(P_1 \times t_1 = P_2 \times t_2 = P_3 \times t_3 = \cdots = N)$, the generalization performance of PC-ELM does not change whatever parameter P changes. This means in PC-ELM, the parameter P, t can choose randomly at the beginning by user, which do not affect the generalization performance in learning process at all.

As observed from performance comparison between PC-ELM and C-ELM in Sect. 5.1 and 5.2, we can found that the performance gaps between PC-ELM and C-ELM with the mechanism of growing hidden nodes are larger than those of comparison performance with fixed network architecture. This because when $\mathbb{L} = L_{max}$, too many hidden parameters should be optimized. In this connection, the dimension of hidden node parameters ($\mathbb{Y}$) is so large that optimization method will be easy to fall into local minima. But in short, the performance of PC-ELM is generally better than C-ELM.

5.2.2 Comparison Performance of Hidden-Node Numbers

In this subsection, the hidden-node numbers among PC-ELM, C-ELM, EM-ELM and ELM are comprised. For regression problem, the target training RMSE are set as: 0.05 for Auto_MPG, 0.08 for Servo, 0.09 for Concrete, 0.08 for Friedman, 0.055 for Bank, 1.35 for California House. For the classification problem, the target training accuracy are set as: 95 % for Breast, 75 % for Waveform, 84 % for Landsat Satellite Image. Table 5 display the hidden-node numbers obtained by these methods including ELM, C-ELM, EM-ELM and PC-ELM for regression and classification cases. As seen from the simulation results given in these Tables, the hidden-node numbers obtained by our proposed method PC-ELM are much smaller than those of other methods. This means PC-ELM working for SLFNs with fixed NN architecture can give more compact network architecture than C-ELM, ELM and EM-ELM.

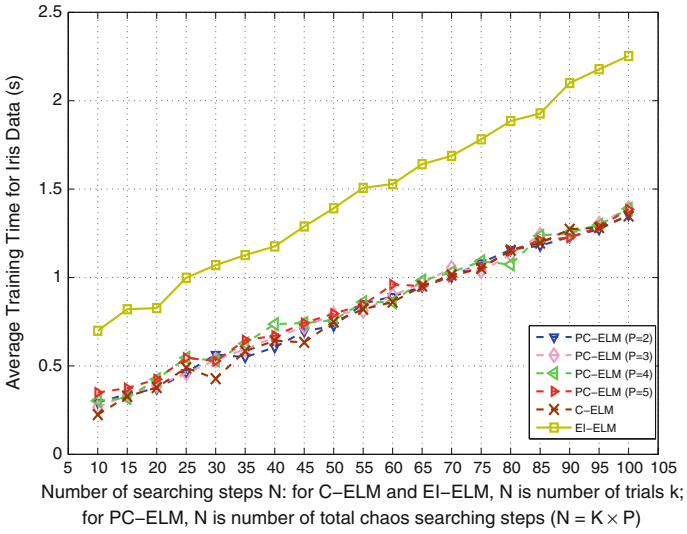

Fig. 14 Testing accuracy of different selecting parameters of different methods in iris case

Huang et al. [3,5] have systematically investigated the performance of ELM, I-ELM, SVR and BP for most data sets tested in this work. It is found that ELM and I-ELM obtain the same generalization performance as SVR but in much simpler and faster way. This is to say, the network complexity of SVR is higher than ELM and I-ELM. As observed from our testing results, the proposed PC-ELM obtains more compact network architecture than ELM and I-ELM. Thus, the network complexity of PC-ELM is much lower than SVR. Further more, it also found in [3,5] that all the two methods ELM, I-ELM usually obtain better generalization performance than BP under the same number of hidden nodes. Interestingly, from our testing results, the proposed PC-ELM obtains better generalization performance than ELM and I-ELM at much higher testing accuracy or testing RMSE. Thus, the proposed PC-ELM performs better than BP as well.

5.3 Training Time Analysis

The training time curves for several cases are shown in Figs. 13 and 14 when 50 hidden nodes are used. From these figures, the spent learning time of C-ELM, EI-ELM and PC-ELM are still approximately linearly increasing with the number of searching steps. In theory the training time of PC-ELM and C-ELM linearly increases with the searching steps increased. It means the training time of PC-ELM with searching steps $N_1 (N_1 = t \times p)$ should be around $\frac{N_1}{N_2}$ times that of PC-ELM with searching steps N_2 under the condition of the same hidden-node numbers. However, simulations results which show in Figs. 13 and 14 do not coincide with above analysis. For example in Fig. 14, average execute time of PC-ELM with 10 searching steps reaches approximately 0.25 s , whereas PC-ELM with 100 searching steps offers about 1.35 s, not 2.5 s. This is because for all the applications, we need to store some temp variables, which normally use large memory. When memory reach computer's choke-point, it will disturb CPU's computation remarkably. This is also why these methods with less number of searching steps need more running time. So the relationship between number of searching steps and training time should be that the training time of PC-ELM with

Table 6 Sensitivity of N for PC-ELM

Datasets	Network architecture	$N = 40$			$N = 100$			$N = 200$			$N = 1{,}000$		
		dev	Mean (%)	Time (s)	dev	Mean (%)	Time (s)	dev	Mean (%)	Time (s)	dev	Mean (%)	Time (s)
Segment	Increased	0.0250	78.53	2.6099	0.0175	80.42	6.2073	0.0101	81.15	11.5519	0.0180	**82.18**	55.70
Diabetes	Increased	0.0390	75.89	1.2808	0.0246	77.34	2.9890	0.0354	77.60	7.7766	0.0173	77.97	35.8880
Iris	Increased	0.0417	90.00	0.3822	0.0301	91.14	0.7909	0.0297	90.86	1.6365	0.0328	91.86	7.9179
Breast cancer	Increased	0.0147	94.70	1.1614	0.0154	94.52	2.5319	0.0089	94.53	4.9983	0.0139	95.27	30.1589
Heart	Increased	0.0092	82.56	0.5140	0.0076	82.64	1.0211	0.0103	83.54	2.1701	0.0117	82.94	9.0651
Satellite	Increased	0.0214	75.64	5.1184	0.0174	75.94	8.7782	0.0102	76.05	16.6250	0.0125	**77.46**	104.2588
Segment	Fixed	0.0124	89.90	0.0926	0.0071	90.29	2.2449	0.0084	89.88	4.4601	0.0117	90.27	21.7621
Diabetes	Fixed	0.0360	74.92	0.4020	0.0424	75.36	0.9906	0.0380	75.21	1.7285	0.0493	75.42	9.1463
Iris	Fixed	0.0275	91.26	0.1747	0.0101	93.06	0.3666	0.0163	92.66	0.6724	0.0226	92.37	3.6317
Movement libras	Fixed	0.0097	65.12	0.5288	0.0071	66.18	1.1893	0.0107	66.09	2.3559	0.0090	65.38	15.0474

dev testing standard deviation, *mean* testing RMSE for regression problem or mean testing accuracy for classification problem, *time* training time

searching steps N_1 should be around $\frac{N_1+\sigma}{N_2+\sigma}$ times that of PC-ELM with searching steps N_2, where σ is impact factor determined by computing environment. The above observations are true for the rest cases as well.

5.4 Sensitivity of the Total Searching Steps N for PC-ELM

In this experiment, the parameters P are selected randomly from positive integer numbers, but the benchmark problems are tested by using $N = 40, 100, 200, 1000$. The experiment results are shown in Table 6. In some cases, the parameter N is not very sensitive, such as breast cancer, heart, diabetes and iris. However, in some cases, such as problem Segment and Satellite, the value of the parameter N is sensitive to the performance of PC-ELM. Based on the conclusion in subsect. 5.3, we know that the training time of PC-ELM will increase with N increased. Thus there is not necessary to choose too large value of N. With the results in Table 6, we can see that PC-ELM with $N = 100$ or $N = 40$ gives a good compromise between training time and testing accuracy. And we also suggest fixing $N \in [40, 100]$.

6 Conclusion

In this paper, we proposed parallel chaos based extreme learning machine. A performance comparison of PC-ELM with other learning algorithms has been carried out on benchmark problems in the areas of regression and classification.

This paper shows that compared with I-ELM, EM-ELM, ELM, OP-ELM, C-ELM and EI-ELM, proposed PC-ELM can achieve faster convergence rate and much more compact network architectures. Unlike the other incremental/sequential/growing learning algorithm such as I-ELM, EI-ELM, which add hidden nodes one by one, or unlike some ELM learning methods with fixed network architecture such as ELM, EM-ELM and OP-ELM, our new approach can work both in fixed network architecture and incremental network architecture. The main goal in this paper was not to show that the PC-ELM is either the best in the term of testing accuracy or the computational time. The main goal is to prove that it is very good compromise among the number of hidden nodes, the training time and the testing accuracy/RMSE.

In PC-ELM, other type of hidden nodes can be chosen, such as RBF hidden nodes, multiplicative nodes, fuzzy rules, fully complex nodes, wavelets, etc. Meanwhile, some other chaos map functions could also be used in our proposed method. The performance of our method on other type of hidden nodes or on other type of chaos map functions will be reported in the future work due to the limited space in this paper. Further more, this paper proved that parameter of hidden nodes in SLFNs can be optimized by any optimization algorithm, thus the performance of PC-ELM with other types of optimization methods is worth investigating. Reasonably speaking, PC-ELM could also be applied to OS-ELM, the performance of PC-ELM with these algorithms is also worth investigating.

Acknowledgments The authors would like to thank Dr Guang-Bin Huang, Nanyang Technological University, Singapore, for constructive suggestions. The authors would like to thank the editor and anonymous reviewers for their invaluable suggestions, which has been incorporated to improve the quality of this paper dramatically. This study was supported by National Natural Science Foundation of China (60835004, 61175075, 61104088, 61173108), and National Technology Support Project (2008BAF36B01).

References

1. Feng G, Huang G, Lin Q, Gay R (2009) Error minimized extreme learning machine with growth of hidden nodes and incremental learning. IEEE Trans Neural Netw 20(8):1352–1357
2. Hornik K (1991) Approximation capabilities of multilayer feedforward networks. IEEE Trans Neural Netw 4:251–257
3. Huang G, Chen L, Siew C (2006) Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans Neural Netw 17(4):879–892
4. Chen L, Zhou L, Pung H (2008) Universal approximation and QoS violation application of extreme learning machine. Neural Process Lett 28(2):81–95
5. Huang G, Zhu Q, Siew C (2006) Extreme learning machine: theory and applications. Neurocomputing 70:489–501
6. Huang G, Chen L (2008) Enhanced random search based incremental extreme learning machine. Neurocomputing 71:3460–3468
7. Miche Y, Sorjamaa A, Bas P, Simula O, Jutten C, Lendasse A (2010) Optimal pruned extreme learning machine. IEEE Trans Neural Netw 21(1):158–162
8. Huang G, Chen L (2007) Convex incremental extreme learning machine. Neurocomputing 70:3056–3062
9. Liang N, Huang G, Paramasivan S, Narasimhan S (2006) A fast and accurate online sequential learning algorithm for feedforward networks. IEEE Trans Neural Netw 17(6):1411–1423
10. Li B, Jiang W (1998) Optimizing complex functions by chaos search. Cybern Syst 29(4):409–419
11. B Cheng, Z Guo, Z Bai, B Cao (2006) Parallel chaos immune evolutionary programming. AI 2006: Advances in Artificial Intelligence, Proceedings 4304:224–232
12. Li L, Yang Y, Peng H, Wang X (2006) An optimization method inspired by "chaotic" ant behavior. Int J Bifurcation Chaos 16(8):2351–2364
13. Peng H, Li L, Yang Y, Liu F (2010) Parameter estimation of dynamical systems via a chaotic ant swarm. Phys Rev E 81(1):016207