Soft Comput (2012) 16:1465–1475
DOI 10.1007/s00500-012-0829-1

FOCUS

A study on random weights between input and hidden layers in extreme learning machine

Ran Wang · Sam Kwong · Xizhao Wang

Published online: 12 February 2012

Abstract Extreme learning machine (ELM), as an emergent technique for training feed-forward neural networks, has shown good performances on various learning domains. This paper investigates the impact of random weights during the training of ELM. It focuses on the randomness of weights between input and hidden layers, and the dimension change from input layer to hidden layer. The direct motivation is to verify as to whether during the training of ELM, the randomly assigned weights exert some positive effects. Experimentally we show that for many classification and regression problems, the dimension increase caused by random weights in ELM has a performance better than the dimension increase caused by some kernel mappings. We assume that via the random transformation, output-samples are more concentrate than input-samples which will make the learning more efficient.

Keywords Extreme learning machine · Random weights · Dimension change

1 Introduction

Extreme learning machine (ELM), which was first proposed by Huang et al. (2004), is a three-layer feed-forward neural network. The training of ELM mainly contains three fundamental steps. The first step is to randomly assign weights between input and hidden layers, the second step is to calculate the hidden layer output matrix, and the third step is to acquire the weights between hidden and output layers via computing the pseudo-inverse of the hidden layer output matrix. Unlike the training of usual feed-forward neural networks in which the weights are iteratively adjusted by using back-propagation (BP) algorithm (Rumelhart et al. 1986), ELM does not include any iteration. Due to the fast training speed and high generalization, it has been applied to various learning domains, such as online learning (Rong et al. 2009), classification (Chacko et al. 2011; Huang et al. 2010; Wang and Huang 2005), and regression (Tang and Han 2009).

The ELM technique got wide developments during the recent years (Jun et al. 2011; Wang et al. 2011). Huang and Siew (2004) proposed the RBF networks case of ELM. Li et al. (2005) extended ELM from the real domain to the complex domain. Huang et al. (2006) focused on showing that single-hidden-layer feedforward networks (SLFNs) with randomly generated hidden nodes can work as universal approximators which results in Incremental ELM (I-ELM), and in the work of Huang and Chen (2007) they further improved I-ELM based on a convex optimization method. Besides, Miche et al. (2010) proposed the optimally pruned ELM which is more robust and generic. It is worth noting that for all the ELM works, the random weights given in the first training-step is the most important part (Zhu et al. 2005), and the impact of random weights on the performance is the essence of ELM. Although all the referred works have respectively exhibited good results on kinds of problems, the random mechanism has not obtained enough justifications so far. In this study, our motivation is to verify as to whether the random weights between input

R. Wang (✉) · S. Kwong
Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong
e-mail: ranwang3@student.cityu.edu.hk

S. Kwong
e-mail: CSSAMK@cityu.edu.hk

X. Wang
Department of Mathematics and Computer Science, Hebei University, Baoding, Hebei 071002, People's Republic of China

and hidden layers exert some positive effects on the performance.

Theoretically Huang (in Huang et al. 2006) proved that, as the number of hidden nodes going to infinity, the ELM with random weights between input and hidden layers can uniformly approximate any continuous function. Here the impact of random weights is to guarantee that the space of functions with ELM forms can be large enough, so that the approximation holds well. It is not clear that, if the weights between input and hidden layers are fixed when the number of hidden nodes goes to infinity, the universal approximation still holds. Practically we cannot design an ELM such that the number of hidden nodes is large enough (Feng et al. 2009; Huang and Chen 2008). However, what is the impact of the random weights when we fix the number of hidden nodes in designing an ELM? This paper also makes an initial attempt to investigate this impact.

Basically, the randomly assigned weights is to make some transformations of the input samples. Obviously, this transformation could be: (1) dimension increase, (2) dimension non-change, and (3) dimension decrease. In most cases, the number of hidden nodes is bigger than the number of input nodes, which means that this transformation usually refers to increase (at least non-decrease) the input dimension. We call this random-based dimension increase. It is similar to the kernel-based technique in support vector machine (SVM) (Vapnik 2000) which maps a dataset from the input space (low dimension) to a feature space (high dimension), such that the dataset in the feature space is linearly separable, where the concrete form of the mapping is usually unknown. We call this kernel-based dimension increase. In order to validate the advantage of the random mechanism, this paper further makes a comparison between the random-based dimension increase and kernel-based dimension increase in ELM for classification and regression problems.

The rest of this paper is organized as follows: in Sect. 2, we make a brief review on ELM technique; in Sect. 3, we consider a special case that the input and hidden layers are with the same dimension, and explore the impact of random weights under this condition; in Sect. 4, we make a discussion on the kernel technique and propose the kernel-based dimension increase for ELM; in Sect. 5, we conduct some experiments to compare the performance of the random-based dimension increase and kernel-based dimension increase; conclusions are given in Sect. 6.

2 Extreme learning machine

Single-hidden layer feed-forward neural networks (Haykin 1999; Nigrin 1993), which have the advantages of simple structure and high learning capability, have been applied to different learning domains, such as pattern recognition, data mining, and regression, etc. The basic structure of SLFNs is shown in Fig. 1.

Given a training set $\{(\mathbf{x}_j, \mathbf{t}_j)|\mathbf{x}_j \in \mathbf{R}^n, \mathbf{t}_j \in \mathbf{R}^m, j = 1, \ldots, N\}$ which contains N distinct examples, the SLFNs with $\tilde{N}$ hidden nodes and activation function $g(\mathbf{x})$ could be formulated as

$$\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{o}_j, \quad j = 1, \ldots, N, \tag{1}$$

where $\mathbf{w}_i$ is the weight vector connecting the input nodes and the i-th hidden node, b_i is the bias of the i-th hidden node, β_i is the weight vector connecting the i-th hidden node and the output nodes, $\mathbf{x}_j$ is the j-th training example, $\mathbf{o}_j$ is the corresponding output of $\mathbf{x}_j$ in the network, and "$\cdot$" denotes the dot product of two vectors.

The standard SLNFs can approximate the N training examples with zero error, i.e., there exists $\mathbf{w}_i$, b_i, and β_i, such that:

$$\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{t}_j, \quad j = 1, \ldots, N. \tag{2}$$

Equation 2 could be written into the following matrix form

$$\mathbf{H}\beta = \mathbf{T}, \tag{3}$$

where

$$\mathbf{H}(\mathbf{w}_i, b_i, \mathbf{x}_j) = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_1 + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_N + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}}, \tag{4}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\tilde{N}}^T \end{bmatrix}_{\tilde{N} \times m}, \quad \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix}_{N \times m}, \tag{5}$$

and $\mathbf{H}$ is called the hidden layer output matrix.

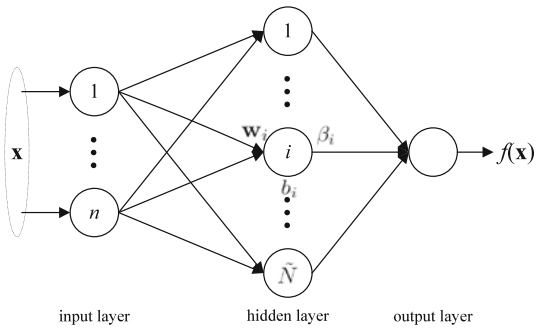


Fig. 1 Structure of single-hidden layer feed-forward neural networks (SLFNs)

In the traditional feed-forward network training methods, such as the BP algorithm (Rumelhart et al. 1986), all the parameters, including the weight vectors and bias values, need to be tuned iteratively, thus the training speed may be very slow (Michell 1997). What is more, they also suffer from other problems, such as local minima, overfitting, etc.

Extreme learning machine is a new network training method which could be treated as a learning model without iterative tuning. Huang (in Huang et al. 2006) has theoretically proved that the input weights and hidden layer biases of SLFNs, i.e., $\mathbf{w}_i$ and b_i, could be randomly assigned if the activation function of the hidden layer is infinitely differentiable. In this case, SLFNs could be considered as a linear system, and the output weights, i.e., β_i, are then determined by the hidden layer output matrix. The ELM algorithm is described as follows:

ELM algorithm: Given a training set $\{(\mathbf{x}_j, \mathbf{t}_j)|\mathbf{x}_j \in \mathbf{R}^n, \mathbf{t}_j \in \mathbf{R}^m, j = 1, \ldots, N\}$, activation function $g(\mathbf{x})$, and hidden node number $\tilde{N}$,

Step 1: Randomly assign input weight $\mathbf{w}_i$ and bias b_i where $i = 1, \ldots, \tilde{N}$.

Step 2: Calculate the hidden layer output matrix $\mathbf{H}$ by (4).

Step 3: Calculate the output weight β

$$\beta = \mathbf{H}^{\dagger}\mathbf{T} \tag{6}$$

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$, and $\mathbf{T} = [t_1, \ldots, t_N]^T$.

The ELM algorithm looks much simpler than most of the existing SLFNs training methods, and is easier to be applied to various application domains (Huang et al. 2010, 2011). In the ELM framework, the second and third steps are easy to understand, where the second step calculates the hidden layer output matrix, and the third step gets the minimum norm least-square solution of a linear system. The most difficult-understanding part lies in the first step.

In ELM, the random weights assignment could be treated as a random mapping from input layer to hidden layer, and this mapping is named as ELM mapping. In the works of Huang et al. (2010), Huang has investigated the relationship between ELM mapping and SVM kernel mapping. However, the impact of this random mapping is still not clear. As aforementioned, the random weights is to increase, at least non-decrease, the input dimension, which could be expressed as $\tilde{N} \geq n$. Obviously, $\tilde{N} = n$ is the minimum setting of the hidden nodes, in which the dimensionality of the input space is same as the ELM feature space; whereas in most cases, it has $\tilde{N} > n$, in which a random-based dimension increase process is performed. Thus, in this paper, we focus on two parts of studies: (1) the impact of random weights when $\tilde{N} = n$; and (2) the effect of the random mechanism when $\tilde{N} > n$.

3 ELM with same dimensions of input and hidden layers

In this section, we consider a special type of ELM in which the dimension of input layer is same as the dimension of hidden layer, which means that the number of hidden nodes is same as the size of input feature vector.

We observe that whether there exists a difference between two network structures, where the first one is the typical ELM with randomly assigned parameters under the condition that the input and hidden layers have the same dimension, and the second one has just 2 layers with directly inputting the training data into the hidden layer. In detail, given that $\mathbf{x}_j = \{x_{j1}, \ldots, x_{jn}\} \in \mathbf{R}^n$ is the j-th training example where $j = 1, \ldots, N$, the hidden layer output matrix of the first structure is calculated as Eq. 4, while the hidden layer output matrix of the second one could be calculated as

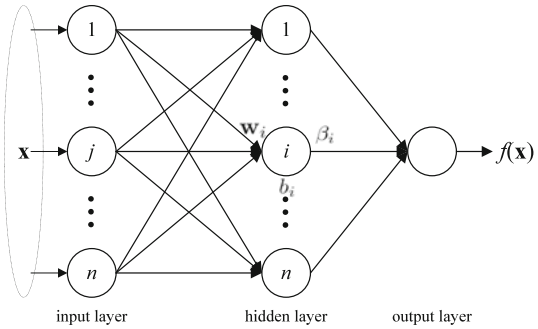
$$\mathbf{H}(b_i, \mathbf{x}_j) = \begin{bmatrix} g(x_{11} + b_1) & \ldots & g(x_{1n} + b_n) \\ \vdots & \ldots & \vdots \\ g(x_{N1} + b_1) & \ldots & g(x_{Nn} + b_n) \end{bmatrix}_{N \times n}. \tag{7}$$

These two structures could be illustrated as in Fig. 2. Except these two, we also investigate a third one in which the training data are directly input into the hidden layer and all the biases of the hidden nodes, i.e., b_i, are fixed as 0.

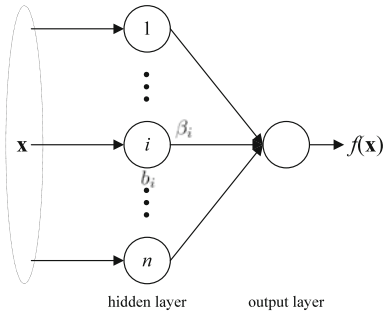
Fourteen classification datasets and six regression datasets are selected to evaluate the impact of random weights and biases. The detail of the datasets are listed in Table 1.

We normalize all the attribute values into interval $[-1,+1]$, and adopt a simple sigmoid activation function $g(\mathbf{x}) = 1/(1 + \exp(\mathbf{x}))$, then conduct the following steps on each dataset:

1. Divide the dataset into 10 folds, each time use 9 folds to conduct the training, and the left fold to do the testing.
2. Use the training data to train the three different structures. The first one is the ELM with random weights and biases, the second one is the network with direct inputs and random biases, and the third one is the network with direct inputs and zero biases.
3. For classification tasks, get the testing accuracy of the learning structures on the testing data; and for regression tasks, get the mean square error (MSE).
4. Conduct the above 10-fold cross validation 10 times, finally, get 100 results and their average value.



(a) ELM with randomly assigned weights



(b) Network with direct input

Fig. 2 Two network structures. **a** ELM with randomly assigned weights where the input and hidden layers have the same dimension. **b** Network with direct input

Table 1 Datasets for classification

Datasets	Types	No. of examples	No. of attributes	No. of classes
Iris	Classification	150	4	3
Abalone	Classification	4,177	8	29
Wdbc	Classification	569	30	2
Car evaluation	Classification	1,728	6	4
Diabetes	Classification	768	8	2
Satimage	Classification	6,435	36	7
Segment	Classification	2,310	19	7
SPECT heart	Classification	267	44	2
Wine	Classification	178	13	3
Chart	Classification	600	60	6
Glass	Classification	214	10	6
Libras	Classification	360	90	15
Yeast	Classification	1,484	8	10
Shuttle	Classification	58,000	9	7
Sinc	Regression	1,500	1	–
Forest fires	Regression	517	12	–
Concrete	Regression	1,030	8	–
Servo	Regression	167	4	–
Wine quality-red	Regression	1,599	11	–
Wine quality-white	Regression	4,898	11	–

Table 2 Classification and regression results

Datasets Classification datasets	ELM-random weights Testing Acc	Direct input (random b_i) Testing Acc	Direct input ($b_i = 0$) Testing Acc
Iris	**0.81267**	0.75667	0.73333
Abalone	**0.25338**	0.24728	0.24827
Wdbc	0.95219	**0.95251**	0.94038
Car evaluation	0.75475	**0.77394**	0.77336
Diabetes	**0.77049**	0.70060	0.69993
Satimage	**0.83586**	0.76467	0.75393
Segment	0.87727	0.89087	**0.89437**
SPECT heart	**0.80429**	0.78899	0.77080
Wine	0.95831	0.96395	**0.96532**
Chart	**0.93450**	0.78817	0.74167
Glass	**0.71838**	0.70498	0.68590
Libras	**0.84895**	0.69877	0.62313
Yeast	0.53259	0.53544	**0.53801**
Shuttle	**0.90714**	0.88545	0.90141
Regression datasets	MSE	MSE	MSE
Sinc	**0.35198**	0.35292	0.35799
Forest fires	**0.05229**	0.07116	0.09057
Concrete	0.28970	**0.25942**	0.26498
Servo	**0.18787**	0.21147	0.21972
Wine quality-red	0.26725	**0.26136**	0.26172
Wine quality-white	0.25497	**0.25103**	0.25108

The average values over the 100 results are listed in Table 2. We see from Table 2 that, for classification problems, the ELM with randomly assigned parameters derives higher testing accuracy on nine datasets out of fourteen, and for regression problems, it derives lower MSE on three datasets out of six. Although in this level, we cannot make a conclusion on which method is definitely better, some analysis could be given. By comparing the direct input methods with zero biases and randomly assigned biases, we see that the randomly assigned b_i have no evident influence on most of the selected datasets. While with the randomly assigned input weights $\mathbf{w}_i$, there appears some obvious differences of the learning performance, and for more than half of the selected datasets, these differences are shown as positive.

Some further investigations are conducted. Suppose that the 100 results of the three learning structures are recorded in $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$. We use Wilcoxon ranksum test to do some statistical analysis between $\mathbf{A}_1$ and $\mathbf{A}_2$, or $\mathbf{A}_1$ and $\mathbf{A}_3$. Wilcoxon ranksum test (Demšar 2006) is a famous non-parametric statistical hypothesis test for accessing that whether

Table 3 Wilcoxon rank-sum test on classification and regression results

Dataset	$[p,h] = ranksum(\mathbf{A}_1, \mathbf{A}_2)$		$[p,h] = ranksum(\mathbf{A}_1, \mathbf{A}_3)$	
	p value	h value	p value	h value
Iris	0.00097	1	1.58e−06	1
Abalone	0.19360	0	0.52569	0
Wdbc	0.56488	0	0.09554	0
Car evaluation	1.27e−05	1	1.95e−05	1
Diabetes	2.84e−19	1	6.24e−23	1
Satimage	2.52e−34	1	2.32e−34	1
Segment	0.00045	1	2.16e−05	1
SPECT heart	0.02975	1	1.96e−07	1
Wine	0.66941	0	0.81720	0
Chart	7.78e−34	1	1.20e−34	1
Glass	0.17329	0	0.00587	1
Libras	9.25e−29	1	3.93e–34	1
Yeast	0.83159	0	0.73294	0
Shuttle	9.58e−18	1	0.00362	1
Sinc	0.27421	0	0.00149	1
Forest fires	0.01172	1	1.66e−06	1
Concrete	4.48e−19	1	8.94e−14	1
Servo	0.00123	1	2.11e−05	1
Wine quality-red	0.00014	1	0.00047	1
Wine quality-white	1.72e−07	1	2.50e−07	1

there exists significant difference between the elements of two sets, or whether one of two groups of independent observations tends to have larger values than the other. With an unknown distribution, the Wilcoxon test is safer and more sensible than the t-test. We use the MATLAB function "[p,h]=ranksum(x,y)" to realize the test where the two parameters, i.e., x and y, are respectively set as $\mathbf{A}_1$ and $\mathbf{A}_2$, or $\mathbf{A}_1$ and $\mathbf{A}_3$. With the significance level α, when the returned p-value is smaller than α, there exists a significant difference between the elements of the two referred sets, and the corresponding h-value is 1; otherwise, there is no significant difference in statistical meaning, and the corresponding h-value is 0. As in (Demšar 2006) mentioned, $\alpha = 0.05$ is probably what we are interested in, and under this setting, the test will not reject the null-hypothesis unless one algorithm almost always outperforms the other. Thus in our tests, the significance level α is always set as 0.05.

The Wilcoxon ranksum test results are listed in Table 3. It is easy to observe that, for both classification and regression problems, there exists some significant differences between the ELMs with random weights and direct inputs, and this observation confirms that the random mapping indeed exerts its effects. In many cases, the ELM with random weights has a performance better than the one without random weights. We assume that the samples are always spread over the input space with some irregular distributions. Via the random transformations, although the dimension has not been changed, the distribution is different which makes the samples more concentrate; thus, the performance is improved.

4 Kernel technique and kernel-based dimension increase for ELM

After verifying the impact of random weights in the dimension non-change case, we need to further verify its impact in the dimension increase case. The direct way is to make some comparisons between the random mapping and some fixed mappings with the same number of hidden nodes. However, optionally fixed mappings have no practical meaning and are not expected to be used in ELM. Fortunately, we can adopt the kernel-based technique which is often employed in SVM.

In this section, we make some discussions on the kernel technique and the kernel-based dimension increase for ELM.

4.1 Two examples

When we use SVM to solve classification problems, the data may not be linearly separable in the input space. In this case, SVM will map the data from the input space into a higher dimensional feature space which makes them linearly separable via a mapping function $\phi : \mathbf{x} \rightarrow \phi(\mathbf{x})$. Since the solution in feature space only involves inner products of mapped points, one can obtain the optimal hyperplane by kernel trick (Chen et al. 2011; Schölkopf and Smola 2002), which expresses the inner product of feature space by a kernel function $k : \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = k(\mathbf{x}_i, \mathbf{x}_j)$.

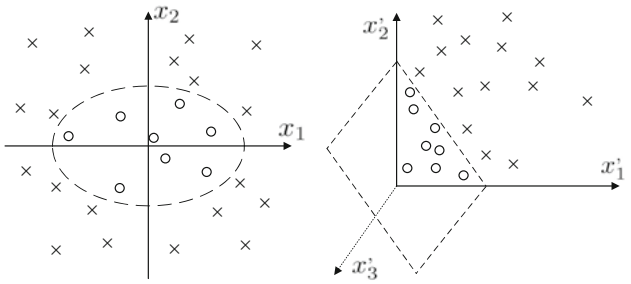
The key problem in SVM is to select proper transformation and its corresponding feature space. When kernel technique is introduced, the problem will be changed to select proper kernel function. In order to illustrate the kernel technique in SVM, we first consider two examples in this part.

Example 1 Homogeneous polynomial kernel with degree 2.

Given a binary classification problem in 2-dimensional space on the training set $\mathbf{X} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_1}$, where $\mathbf{x}_i = (x_{i1}, x_{i2}) \in \mathbf{R}^2$ is the i-th example and $y_i \in \{+1, -1\}$ is the label of $\mathbf{x}_i$. This problem is shown as in Fig. 3a, where the two classes of examples could be separated by an ellipse which takes the origin as its center, and this ellipse is denoted by:

$$w_1 x_1^2 + w_2 x_2^2 + \sqrt{2} w_3 x_1 x_2 + b = 0 \tag{8}$$

where w_1, w_2, w_3, and b are real values.



(a) Original space **(b)** Feature space

Fig. 3 A 2-dimensional classification problem. **a** The problem in 2-dimensional original space where the classifier is an ellipse. **b** The problem in 3-dimensional feature space where the classifier is a hyperplane

It is obvious that this problem is non-linearly separable in the original space. We thus introduce a non-linear mapping ϕ which maps the examples of $\mathbf{X}$ into a feature space $\mathbf{F}$:

$$\phi\colon \mathbf{X} \subseteq \mathbf{R}^2 \mapsto \mathbf{F} = \mathbf{R}^3$$
$$\mathbf{x} = (x_1, x_2) \mapsto \phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2) = (x_1', x_2', x_3'). \quad (9)$$

With this mapping, the 2-dimensional input examples become 3-dimensional mapped points, and the ellipse in the original space becomes a hyperplane in the feature space which is denoted by:

$$w_1x_1' + w_2x_2' + w_3x_3' + b = 0. \quad (10)$$

As shown in Fig. 3b, in the 3-dimensional feature space, we can use SVM technique to get the separating hyperplane, thus to obtain the ellipse of the original space. Furthermore, the inner product of two mapped points, i.e., $\phi(\mathbf{x_i})$ and $\phi(\mathbf{x}_j)$, is calculated as:

$$\begin{aligned}\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle &= \langle (x_{i1}^2, x_{i2}^2, \sqrt{2}x_{i1}x_{i2}), (x_{j1}^2, x_{j2}^2, \sqrt{2}x_{j1}x_{j2})\rangle \\ &= x_{i1}^2x_{j1}^2 + x_{i2}^2x_{j2}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} \\ &= (x_{i1}x_{j1} + x_{i2}x_{j2})^2 \\ &= \langle \mathbf{x}_i, \mathbf{x}_j\rangle^2. \end{aligned} \quad (11)$$

Thus in order to to reduce the computations, the inner products of ϕ are further replaced by the following kernel function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle = \langle \mathbf{x}_i, \mathbf{x}_j\rangle^2, \quad (12)$$

and this kernel is the homogeneous polynomial kernel with degree 2. The mapping for homogeneous polynomial kernel with degree d includes every d-degree elements of the data.

It is worth noting that there exists a similar mapping ϕ_2:

$$\phi_2 : \mathbf{X} \subseteq \mathbf{R}^2 \mapsto \mathbf{F} = \mathbf{R}^4$$
$$\mathbf{x} = (x_1, x_2) \mapsto \phi_2(\mathbf{x}) = (x_1^2, x_2^2, x_1x_2, x_2x_1). \quad (13)$$

This mapping maps the examples into a different feature space, but has the same kernel function with ϕ, i.e., $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi_2(\mathbf{x}_i), \phi_2(\mathbf{x}_j)\rangle = \langle \mathbf{x}_i, \mathbf{x}_j\rangle^2$.

Example 2 Inhomogeneous polynomial kernel with degree 2.

Given another binary classification problem in 2-dimensional space on the training set $\mathbf{X} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_2}$, where $\mathbf{x}_i = (x_{i1}, x_{i2}) \in \mathbf{R}^2$ and $y_i \in \{+1, -1\}$. Suppose that this problem could be solved by a quadratic curve in the plane, which is denoted by:

$$w_1 + 2w_2x_1 + 2w_3x_2 + 2w_4x_1x_2 + w_5x_1^2 + w_6x_2^2 + b = 0 \quad (14)$$

where $w_1, \ldots, w_6$ and b are real values. We consider to map the input points from the 2-dimensional space to the 6-dimensional space by the mapping ϕ where:

$$\begin{aligned}\phi(\mathbf{x}) &= (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2) \\ &= (x_1', x_2', x_3', x_4', x_5', x_6'). \end{aligned} \quad (15)$$

Thus, the quadratic curve in the 2-dimensional space is mapped as a hyperplane in the 6-dimensional space, and this hyperplane is:

$$w_1 + \sqrt{2}w_2x_1' + \sqrt{2}w_3x_2' + \sqrt{2}w_4x_3' + w_5x_5' + w_6x_6' + b = 0. \quad (16)$$

Similarly, the inner product of ϕ is calculated as:

$$\begin{aligned}\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle &= 1 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + 2x_{i1}x_{j1}x_{i2}x_{j2} \\ &\quad + x_{i1}^2x_{j1}^2 + x_{i2}^2x_{j2}^2 \\ &= (\langle \mathbf{x}_i, \mathbf{x}_j\rangle + 1)^2. \end{aligned} \quad (17)$$

Thus the kernel function is expressed as:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle = (\langle \mathbf{x}_i, \mathbf{x}_j\rangle + 1)^2, \quad (18)$$

and this kernel is the inhomogeneous polynomial kernel with degree 2. The mapping for inhomogeneous polynomial kernel with degree d includes every elements of the data up to d-degree.

4.2 Kernel functions

In SVM technique, similarities are measured by inner products. These inner products are strongly dependent on the selected kernel functions. Once the mapping is fixed, the kernel function is then determined which could be used in the construction of SVM (Schölkopf and Smola 2002).

Basically there are two kinds of kernels. The first kind is known as translation invariant kernel which is denoted by $k(\mathbf{x}_i, \mathbf{x}_j) = f(d(\mathbf{x}_i, \mathbf{x}_j))$ where $d(\mathbf{x}_i, \mathbf{x}_j) = ||\mathbf{x}_i - \mathbf{x}_j|| = \sqrt{\langle \mathbf{x}_i - \mathbf{x}_j, \mathbf{x}_i - \mathbf{x}_j\rangle}$. This kind of kernel is determined by

the difference between two input examples and independent of the absolute position of each individual one. And the second kind is known as dot product kernel which is denoted by $k(\mathbf{x}_i, \mathbf{x}_j) = f(\langle \mathbf{x}_i, \mathbf{x}_j \rangle)$. This kind of kernel is directly expressed as the form of inner product. The basic definitions and determinant of kernel functions could be found in "Appendix". Several most well-known kernel functions are listed here:

1. Homogeneous polynomial kernel with degree d: $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle^d$.
2. Inhomogeneous polynomial kernel with degree d: $k(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + 1)^d$.
3. Gaussian radial basis function (rbf) kernel: $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-||\mathbf{x}_i - \mathbf{x}_j||^2 / 2\sigma^2)$.
4. Sigmoid kernel: $k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(v\langle \mathbf{x}_i, \mathbf{x}_j \rangle + c)$.

Once the kernel functions are determined, it is more convenient to map the train data into higher dimensional feature space, thus to obtain a proper SVM classifier.

4.3 Kernel-based dimension increase for ELM

As it is aforementioned, the first step of ELM algorithm is a dimension change process, and this change is always referred to as dimension increase, at least non-decrease. In the basic framework of ELM, this dimension increase is realized by some randomly assigned weights, and we call this method random-based dimension increase.

We could also apply the kernel mappings to realize the dimension increase process, and this method is called kernel-based dimension increase. Note that for many kernels, the mapping function is hard to obtain. While the solution of the ELM could not be replaced by inner products of the input space, thus we need to select some kernels whose associated mappings are available to use. Under this condition, the polynomial kernels will be some good choices.

Still given the training set $\{(\mathbf{x}_j, \mathbf{t}_j) | \mathbf{x}_j \in \mathbf{R}^n, \mathbf{t}_j \in \mathbf{R}^m, j = 1, \ldots, N\}$, and activation function $g(\mathbf{x})$. We adopt the kernel mapping $\phi(\mathbf{x}_j) \rightarrow (x'_{j1} \ldots, x'_{j\tilde{N}})$, where $\tilde{N} > n$. Thus the hidden layer output matrix will be calculated as:

$$\mathbf{H}(\phi, b_i, \mathbf{x}_j) = \begin{bmatrix} g(x'_{11} + b_1) & \cdots & g(x'_{1\tilde{N}} + b_{\tilde{N}}) \\ \vdots & \ldots & \vdots \\ g(x'_{N1} + b_1) & \cdots & g(x'_{N\tilde{N}} + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}}, \tag{19}$$

and the later step is the same as the basic ELM algorithm.

Kernel-based dimension increase ELM algorithm: Given a training set $\{(\mathbf{x}_j, \mathbf{t}_j) | \mathbf{x}_j \in \mathbf{R}^n, \mathbf{t}_j \in \mathbf{R}^m, j = 1, \ldots, N\}$, and kernel mapping function $\phi(\mathbf{x})$,

Step 1: Realize the mapping $\phi(\mathbf{x}_j) \rightarrow (x'_{j1}, \ldots, x'_{j\tilde{N}})$ where $j = 1, \ldots, N$.

Step 2: Calculate the hidden layer output matrix $\mathbf{H}$ by (19).

Step 3: Calculate the output weight β

$$\beta = \mathbf{H}^{\dagger}\mathbf{T} \tag{20}$$

where $\mathbf{H}^{\dagger}$ is the Moore–Penrose generalized inverse of matrix $\mathbf{H}$, and $\mathbf{T} = [t_1, \ldots, t_N]^T$.

5 Experimental comparison between random-based and kernel-based dimension increase in ELM training

In this section, we conduct some experimental comparisons between the random-based dimension increase and kernel-based dimension increase in training ELM. The goal is to verify that when the number of hidden nodes is fixed, whether the random mapping mechanism can exert some positive effects on the learning and outperform a fixed mapping. It is well known that for a given sample set and a specific kernel, the feature space is unique (Schölkopf and Smola 2002). But for many kernels, the concrete form of the mapping is hard to produce. Thus, in order to realize the kernel-based dimension increase, we need to select some kernels whose associated mapping is easy to obtain. In this case, both homogenous polynomial kernel and inhomogenous polynomial kernel are considered.

For comparison purpose, we need to guarantee that the random-based and kernel-based methods map the data into the feature spaces with same dimension. However, when the dimension of the feature space is very high, the finite examples may become scattered, under this condition, it is meaningless to make any investigation. Thus, a comparatively lower dimensional feature space is preferred. In fact, for 10 input features, the 2-degree, 3-degree, and 4-degree homogenous polynomial kernel mapping will lead to a 55-dimensional, 220-dimensional, and 715-dimensional feature space; the 2-degree, 3-degree, and 4-degree inhomogenous polynomial kernel mapping will lead to a 66-dimensional, 286-dimensional, and 1001-dimensional feature space. For some datasets, the number of input features is even higher, and usually, the computer will be out of memory when calculating such a high-dimensional mapping. Thus in our experiments, we just adopt the 2-degree polynomial kernel mappings. Once the kernel mapping is fixed, the number of hidden nodes is then fixed, and we can perform the random-based dimension increase with this specific setting.

The same classification and regression datasets listed in Table 1 are used again in this section. We normalize all the

attribute values into [−1,+1], adopt a simple sigmoid activation function $g(\mathbf{x}) = 1/(1 + \exp(\mathbf{x}))$, and conduct 10-fold cross-validation 10 times on each datasets. We perform the experiments under MATLAB 7.9.0, which are executed on a computer with a 3.16-GHz Intel Core 2 Duo CPU, a maximum 4.00-GB memory, and 64-bit windows 7 system. For classification tasks, we report the averaged testing accuracy; for regression tasks, we report the averaged MSE; and for each problem, we also report the corresponding number of hidden nodes, the averaged training and testing time. The final result for 2-degree homogenous polynomial kernel mapping and 2-degree inhomogenous polynomial kernel mapping are respectively listed in Tables 4 and 5.

Compared with 2-degree homogenous polynomial kernel mapping, we see from Table 4 that for both classification and regression problems, the random input weights derive a performance better than the kernel mapping on most of the selected datasets. This result verifies the positive impact of the random input weights, which means that by doing some random transformations on the input samples (usually some dimension increasing transformations), the SLFNs can possibly achieve a better learning performance. We assume that, by some random transformations, the samples will be more concentrate in the feature space which makes the learning much easier.

Compared with the 2-degree inhomogenous polynomial kernel mapping, we observe from Table 5 that the random input weights get a better performance on more than half of the selected datasets. In fact the difference between these two methods are quite trivial in many cases. This is possibly caused by the high dimensionality of the feature space.

Regarding the CPU time evaluation, it is easy to observe from Tables 4 and 5 that both the training and testing procedures of the random-based dimension increase are much faster than the kernel-dimension increase. Usually the computational expenses for calculating kernel mappings are quite high, thus the time advantages of random-based ELM are obvious.

6 Concluding remarks

In this paper, we make a study on the impact of random weights between input and hidden layers in ELM. The study is mainly constructed by two part: (1) the effect of random weights when the input and hidden layers have the same dimension, and (2) the impact of the random-based dimension increase in ELM. Experimentally we show that the randomly assigned input weights do have some effects on the learning, and for many classification or regression tasks, these effects are shown as positive. Compared with some fixed mappings such as the polynomial kernel mapping, the random one can get a higher performance in most cases. The reason is initially explained as that, after the random transformation, the output-samples become more concentrate in the feature space which makes the learning much easier.

Acknowledgments This paper is partly supported by City University Strategic Research Grant (SRG) 7002680.

Appendix

A Kernel function and its determinant

A.1 Definition of Kernel function

Definition 1 (*Kernel function*) Suppose that $\mathbf{X}$ is a subset of $\mathbf{R}^n$. The function $k(\mathbf{x}_i, \mathbf{x}_j)$ defined on $\mathbf{X} \times \mathbf{X}$ is called a kernel function if there exists a mapping $\phi : \mathbf{X} \to \mathbb{H}$, $\mathbf{x} \to \phi(\mathbf{x})$, such that $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, where $\mathbb{H}$ denotes a Hilbert space, and $\langle , \rangle$ denotes the inner product of $\mathbb{H}$.

A.2 Mercer Theorem

Theorem 1 (Mercer Theorem) *Suppose that χ is a compact set of $\mathbf{R}^n$. For any symmetric function $k(\mathbf{x}_i, \mathbf{x}_j)$ defined on $\chi \times \chi$, the necessary and sufficient condition for it being the inner product of a feature space is that: for any $\phi(\mathbf{x}) \neq 0$ and $\int \phi^2(\mathbf{x}) d\mathbf{x} < \infty$, there has $\int \int k(\mathbf{x}, \mathbf{x}_i) \phi(\mathbf{x}) \phi(\mathbf{x}_i) d\mathbf{x} d\mathbf{x}_i > 0$.*

A.3 Definition of Gram matrix (Kernel matrix)

Definition 2 Given a function $k : \chi \times \chi \to \mathbb{K}$ (where χ is a compact set of $\mathbf{R}^n$, $\mathbb{K}$ is the mapped set) and patterns $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \chi$, the $N \times N$ matrix $\mathbf{K}$ with elements $\mathbf{K}_{i,j} := k(\mathbf{x}_i, \mathbf{x}_j)$ is called the Gram matrix (or Kernel matrix) of k with respect to $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

A.4 Property of Kernel function

Theorem 2 *Suppose that χ is a compact set of $\mathbf{R}^n$. For the continuous and symmetric function $k(\mathbf{x}_i, \mathbf{x}_j)$ defined on $\chi \times \chi$, the necessary and sufficient condition for it being a kernel function is that: the Gram Matrix of k with respect to any $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \chi$ is positive semi-definite.*

Table 4 Random-based dimension increase versus 2-degree homogenous polynomial kernel-based dimension increase

Dataset	Random-based		Kernel-based		No. hidden nodes
Classification dataset	Accuracy	Train sec. Test sec.	Accuracy	Train sec. Test sec.	
Iris	**0.94333**	0.00078 0.00031	0.79733	0.00437 0.00172	10
Abalone	**0.26367**	0.03604 0.00203	0.26200	0.25569 0.03994	36
Wdbc	**0.74387**	1.02087 0.00452	0.73758	2.56996 0.80465	465
SPECTF	**0.64205**	0.25335 0.00530	0.61892	4.38909 3.21549	990
Car	**0.80674**	0.00718 0.00156	0.71813	0.05928 0.00967	21
Diabetes	0.77121	0.00936 0.00047	**0.77580**	0.05569 0.01466	36
Satimage	**0.89958**	5.60886 0.05288	0.87447	22.03764 3.08820	666
Segment	**0.95091**	0.24835 0.00452	0.94818	1.28420 0.30763	190
Wine	**0.94768**	0.01638 0.00094	0.94547	0.10218 0.05320	91
Chart	**0.98483**	1.95516 0.01498	0.88200	20.12569 12.44779	1,830
Glass	**0.88239**	0.00733 0.00031	0.87016	0.04867 0.02184	55
Libras	**0.87792**	1.08779 0.01966	0.73890	91.55012 78.53028	4,095
Yeast	0.59524	0.00920 0.00343	**0.60272**	0.10702 0.01763	36
Shuttle	**0.97730**	1.31493 0.03167	0.96879	3.77585 0.41013	45
Regression dataset	MSE	Train sec. Test sec.	MSE	Train sec. Test sec.	No. hidden nodes
Sinc	**0.36737**	0.00094 0.00062	0.37494	0.00125 0.00047	1
Forest fires	**0.10262**	0.01841 0.00000	0.13103	0.14024 0.04618	78
Concrete	0.21774	0.01045 0.00031	**0.20693**	0.08159 0.01669	36
Servo	**0.13935**	0.00031 0.00031	0.17309	0.00359 0.00187	10
Wine-red	**0.26110**	0.02870 0.00156	0.26124	0.25912 0.05304	66
Wine-white	**0.24411**	0.09266 0.00312	0.24782	0.59468 0.10468	66

Table 5 Random-based dimension increase versus 2-degree inhomogenous polynomial kernel-based dimension increase

Dataset	Random-based		Kernel-based		No. hidden nodes
Classification dataset	Accuracy	Train sec. Test sec.	Accuracy	Train sec. Test sec.	
Iris	0.96200	0.00109 0.00062	**0.96733**	0.00374 0.00250	15
Abalone	**0.26336**	0.05132 0.00312	0.25748	0.32947 0.05444	45
Wdbc	0.65638	1.18639 0.00406	**0.65789**	2.86964 0.90340	496
SPECTF	**0.64080**	0.26208 0.00359	0.61615	4.67847 3.44591	1,035
Car	0.82245	0.00764 0.00187	**0.88154**	0.08455 0.01482	28
Diabetes	0.76887	0.00858 0.00094	**0.77423**	0.08112 0.01966	45
Satimage	**0.89936**	6.61616 0.05320	0.87719	24.41556 3.40800	703
Segment	**0.95567**	0.28938 0.00593	0.95134	1.46329 0.34960	210
Wine	**0.93558**	0.02028 0.00125	0.93005	0.13307 0.06536	105
Chart	**0.98500**	2.11179 0.01498	0.90517	21.25373 13.03622	1,891
Glass	0.86414	0.00827 0.00062	**0.87312**	0.06833 0.02980	66
Libras	**0.87790**	1.05129 0.02340	0.73240	95.47355 83.06585	4,186
Yeast	0.59635	0.01825 0.00156	**0.59868**	0.14477 0.02792	45
Shuttle	**0.98118**	1.88808 0.03978	0.97303	5.16129 0.51278	55
Regression dataset	MSE	Train sec. Test sec.	MSE	Train sec. Test sec.	No. hidden nodes
Sinc	0.30279	0.00109 0.00062	**0.20690**	0.00406 0.00187	3
Forest fires	**0.11278**	0.02168 0.00156	0.97458	0.17379 0.06006	91
Concrete	0.20844	0.01404 0.00062	**0.19892**	0.10717 0.02231	45
Servo	0.12599	0.00094 0.00062	**0.10864**	0.00359 0.00218	15
Wine-red	**0.26225**	0.04040 0.00250	0.26467	0.31684 0.07036	78
Wine-white	**0.24389**	0.12948 0.00406	0.26134	0.73991 0.14430	78

References

Chacko B, Vimal Krishnan V, Raju G, Babu Anto P (2011) Handwritten character recognition using wavelet energy and extreme learning machine. Int J Mach Learn Cybern. doi: 10.1007/s13042-011-0049-5

Chen C, Zhang J, He X, Zhou Z (2011) Non-parametric kernel leanring with robust pairwise constraints. Int J Mach Learn Cybern. doi:10.1007/s13042-011-0048-6

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

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

Haykin S (1999) Neural networks: a comprehensive foundation. Prentice hall, New Jersey

Huang G, Chen L (2007) Convex incremental extreme learning machine. Neurocomputing 70(16–18):3056–3062

Huang G, Chen L (2008) Enhanced random search based incremental extreme learning machine. Neurocomputing 71(16):3460–3468

Huang G, Chen L, Siew C (2006) Universal approximation using incremental constructive feedforward networks with random hidden nodes. Neural Netw IEEE Transact 17(4):879–892

Huang G-B, Ding X, Zhou H (2010) Optimization method based extreme learning machine for classification. Neurocomputing 74(1-3):155–163

Huang G, Siew C (2004) Extreme learning machine: Rbf network case. In: Eighth IEEE Control, Automation, Robotics and Vision Conference (ICARCV 2004), vol 2, pp 1029–1036

Huang G, Wang D, Lan Y (2011) Extreme learning machines: a survey. Int J Mach Learn Cybern 2(2):107–122

Huang G, Zhou H, Ding X, Zhang R (2010) Extreme learning machine for regression and multiclass classification. Syst Man Cybern Part B Cybern IEEE Transact 99:1–17

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

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

Jun W, Shitong W, Chung F (2011) Positive and negative fuzzy rule system, extreme learning machine and image classification. Int J Mach Learn Cybern 2(4):261–271

Li M, Huang G, Saratchandran P, Sundararajan N (2005) Fully complex extreme learning machine. Neurocomputing 68:306–314

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

Michell T (1997) Machine learning. McGrawHill, USA

Nigrin A (1993) Neural networks for pattern recognition. The MIT press, Cambridge

Rong H, Huang G, Sundararajan N, Saratchandran P (2009) Online sequential fuzzy extreme learning machine for function approximation and classification problems. Syst Man Cybern Part B Cybern IEEE Transact 39(4):1067–1072

Rumelhart D, Hintont G, Williams R (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536

Schölkopf B, Smola A (2002) Learning with kernels. The MIT Press, Cambridge

Tang X, Han M (2009) Partial lanczos extreme learning machine for single-output regression problems. Neurocomputing 72(13): 3066–3076

Vapnik V (2000) The nature of statistical learning theory. Springer, Berlin

Wang D, Huang G (2005) Protein sequence classification using extreme learning machine. In: Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN'05). vol 3, pp 1406–1411

Wang W, Chen A, Feng H (2011) Upper integral network with extreme learning mechanism. Neurocomputing 74(16):2520–2525

Zhu Q, Qin A, Suganthan P, Huang G (2005) Evolutionary extreme learning machine. Pattern Recognit 38(10):1759–1763