Extreme Learning Machine for Function Approximation – Interval Problem of Input Weights and Biases

Grzegorz Dudek
Department of Electrical Engineering
Czestochowa University of Technology
42-200 Czestochowa, Al. Armii Krajowej 17, Poland
dudek@el.pcz.czest.pl

***Abstract*—In this article the approximation capability of the extreme learning machine is studied. Specifically the impact of the range from which the input weights and biases are randomly generated on the fitted curve complexity is analyzed. The guidance for how to generate the input weights and biases to get good performance in approximation of the functions of one variable is provided.**

Keywords—extreme learning machine; function approximation; feedforward neural networks

I. INTRODUCTION

Due to the excellent approximation properties feedforward neural networks (FNNs) are widely used in regression and classification problems. Their adaptive nature allows them for learning from observed data and generalize well in unseen examples. As universal approximators FNNs are able to model any given function and nonlinear relationships. The training algorithms used in FNNs usually employ some form of gradient descent method. It is known that gradient descent-based learning is generally slow and converges to local minima.

Recently, a new learning algorithm has been proposed for training single-hidden-layer FNNs named the extreme learning machine (ELM) [1]. The novelty in ELM is that the input weights (linking the inputs with hidden layer) and hidden neuron biases need not to be adjusted. They are randomly initiated according to any continuous sampling distribution and fixed without iteratively tuning. The only parameters need to be learned are the output weights linking the hidden and output layers. For this reason, ELM can be simply considered as a linear system in which the output weights can be analytically determined through simple generalized inverse operation of the hidden layer output matrices. The learning speed of ELM can be thousands of times faster than traditional gradient descent-based learning.

As theoretical studies have shown, even with randomly generated hidden nodes, ELM with wide type of activation functions (AFs) maintains the universal approximation capability [2] and it can classify any disjoint regions [3]. In the opinion of the authors of [1] ELM, different from traditional learning algorithms, tends to reach the smallest training error and the smallest norm of weights, which implies good generalization performance.

In this work we look inside ELM and analyze its weights and biases. We study how the ranges from which input weights and biases are randomly generated affect the approximation ability of ELM. The aim of this work is to provide guidance for how to generate the input weights and biases to get good performance in approximation of the functions of one variable. To achieve this, we perform experiments using the target function of varying complexity. The single variable target function enables us to visualize results and look at how the activation functions of neurons compose the fitting curve. This gives us tips on how to generate weights.

II. BASIC EXTREME LEARNING MACHINE

ELM was originally proposed by Huang et al. [1] as a new learning algorithm for the single-hidden-layer FNNs. The main concepts behind the ELM are that: (i) the weights and biases of the hidden nodes are generated randomly and are not adjusted, and (ii) the output weights are determined analytically.

The output function of ELM is of the form (one output case):

$$f_L(\mathbf{x}) = \sum_{i=1}^{L} f_i(\mathbf{x}) = \sum_{i=1}^{L} \beta_i h_i(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta}, \qquad (1)$$

where L is a number of hidden neurons, $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), ..., h_L(\mathbf{x})]$ is the output vector of the hidden layer with respect to the input $\mathbf{x}$, and $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_L]^T$ is the vector of the output weights (between the hidden layer and the output node).

$h_i(\mathbf{x})$ is the output of the i-th hidden node (or its AF), while $f_i(\mathbf{x})$ is its weighted output. The output function $f_L(\mathbf{x})$ is the linear combination of AFs $h_i(\mathbf{x})$. Hidden neurons maps the data from d-dimensional input space to the L-dimensional feature space H, and thus, $\mathbf{h}(\mathbf{x})$ is a feature mapping. This mapping is nonlinear if it is formed by any nonlinear piecewise continuous AFs. Different AFs may be used in different

978-1-4799-8322-3/15/$31.00 ©2015 IEEE

hidden neurons. The most popular AFs are: sigmoid, Gaussian, multiquadric, step, triangular and sine functions. The sigmoid AF is of the form:

$$h_i(\mathbf{x}) = \frac{1}{1+\exp(-(\mathbf{a}_i \cdot \mathbf{x} + b_i))}, \quad (2)$$

where $\mathbf{a}_i = [a_{i,1}, a_{i,2}, ..., a_{i,d}]^T$ is the vector of the input weights of the i-th hidden neuron, b_i is its bias, and $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of $\mathbf{a}_i$ and $\mathbf{x}$.

Characteristically for ELM the hidden nodes parameters, $\mathbf{a}_i$ and b_i, are randomly generated according to any continuous probability distribution instead of being explicitly trained. This process is independent of the training data.

The output weights β_i are solved by minimizing the approximation error:

$$\min\|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\|^2, \quad (3)$$

where $\mathbf{H}$ is the hidden layer output matrix and $\mathbf{T}$ is the training data output matrix.

The optimal solution to (3) is:

$$\boldsymbol{\beta}^* = \mathbf{H}^+\mathbf{T}, \quad (4)$$

where $\mathbf{H}^+$ is the Moore–Penrose generalized inverse of matrix $\mathbf{H}$.

To improve generalization performance of ELM its regularized version was proposed [3]. Recent developments in ELM including improving its stability and compactness, learning on online sequential data, imbalanced data, noisy and missing data, and incremental ELM are described in [4].

III. Approximation Properties of ELM Depending on the Intervals of Input Weight and Biases

In this section we examine ELM in the function approximation problem. To visualize results we limit to the real-valued functions of a single variable. In the first experiment we use the SinC function, which was also used in [1] as benchmark for ELM performance evaluation:

$$g(x) = \begin{cases} \sin(x)/x, & \text{for } x \neq 0, \\ 1, & \text{for } x = 0, \end{cases} \quad (5)$$

For brevity, we use the following acronyms:

TF: target function $g(x)$,
FC: fitted curve $f_L(x)$,
AF: activation function $h_i(x)$,
II: input interval, i.e. the interval to which inputs are normalized.

All the simulations are carried out in MATLAB R2010b environment. For ELM we use Matlab function `elm` created by the authors of ELM algorithm (downloaded from http://www.ntu.edu.sg/home/egbhuang/elm_random_hidden_nodes.html). In this implementation the input weights and biases are generated randomly from the uniform distribution: weights from the range [–1, 1] and biases form the range [0, 1]. In our simulations we use sigmoid AFs.

In the first experiment we repeat simulations performed in [1]. The training dataset contains 5000 points (x_i, y_i), where x_i are uniformly randomly distributed on the interval (–10, 10) and y_i are distorted by adding the uniform noise distributed in [–0.2, 0.2]. The testing set is generated similarly but without noise. Thus the test points represent the TF. According to [1] the inputs are normalized into the range [0, 1], while the outputs are normalized into the range [–1, 1]. The ELM is composed of 20 hidden neurons with sigmoid AFs.

The FC in Fig. 1 is shown. This curve unexpectedly differs from that one shown in Fig. 1 in [1]. In our case the FC demonstrates substantial deviations from the TF, while in [1] it almost coincides with the TF. Adding neurons do not improve the FC. Note that the deviations between FC and TF are not due to overfitting to the training points. In case of such a large number of training points as in our experiment the overfitting should not have happened, because the training points cover evenly every part of the TF in excess.

We observe that if the inputs are not preprocessed by normalization to the range [0, 1] the FC almost perfectly approximates the TF. Thus the problem is in the II, i.e. the range of x. The inputs of ELM are processed by the AFs of neurons. The outputs of our 20 neurons for inputs x normalized to [0, 1] and without normalization, i.e. from the interval (–10, 10), are shown in Fig. 2. It can be seen from this figures that ELM operates in these two cases on different fragments of AFs. These AFs are unchanged in ELM during learning. The AF fragments in IIs are the basis functions forming the fitting curve by linear combination. The components of this combination, i.e. AFs multiplied by the output weights β_i, are shown in Fig. 3. The sum of these curves gives FC.

The slope of each individual AF is more or less the same along the interval [0, 1], whereas in the interval (–10, 10) the AF slope changes significantly. In the latter case the larger variability of the AFs is in the middle of the II, whereas at the borders of II flat parts of many AFs can be observed. A measure of the AF variability in x is its slope in this point, i.e. its derivative $dh_i(x)/dx$. Let us define the slope function of the set of AFs as the average slope of individual AFs:

$$s_h(x) = \frac{1}{L}\sum_{i=1}^{L}\left|\frac{dh_i(x)}{dx}\right|, \quad (6)$$

and express it as a percentage of the maximum value of s_h in the II:

$$s_{h\%}(x) = 100 \cdot \frac{s_h(x)}{\max\limits_{x \in \mathrm{II}} s_h(x)}. \quad (7)$$

The same can be defined for the weighted AFs:

$$s_f(x) = \frac{1}{L}\sum_{i=1}^{L}\left|\frac{df_i(x)}{dx}\right| = \frac{1}{L}\sum_{i=1}^{L}\left|\beta_i \frac{dh_i(x)}{dx}\right|, \quad (8)$$

$$s_{f\%}(x) = 100 \cdot \frac{s_f(x)}{\max\limits_{x\in \mathrm{II}} s_f(x)}. \quad (9)$$

As we can see from (8) the slopes of AFs are modified in the output layer by the output weights β_i to get FC that has the best fit to data points.

The plots of $s_{\%}(x)$ shows the variability of the set of AFs or weighted AFs in II. We can observe how the variability changes along II. The AF variability for our examples of [0, 1] and (–10, 10) intervals is illustrated in Fig. 4. In this figure the variability of the TF is also shown as the percentage slope function defined as follows:

$$s_{g\%}(x) = 100 \cdot \left|\frac{dg(x)}{dx}\right| \cdot \left(\max_{x\in \mathrm{II}}\left|\frac{dg(x)}{dx}\right|\right)^{-1}. \quad (10)$$

In (–10, 10) interval the largest variability of the AFs is in the middle part of II, where there is the largest variability of TF. And the less variability of AFs at the borders corresponds to the less variability of TF in these regions. In the case of [0, 1] interval, fitting the curve to TF in the middle part of II requires the adequate slopes of AFs, which are adjusted by the output weights. But because individual AF has similar slope in the whole II, increasing the slopes by output weights for a good fitting in the steep parts of TF causes the same increasing in the flat regions of TF at the borders. This results in too much variability of ELM at the borders. This problem would not arise in the case of TF of the same variability in II, e. g. the pure sine function. If the variability of the TF changes in II the slopes of the separate AFs should changes as well. Moreover, the most variable regions of AFs should correspond to the most variable regions of TF. In the case of multi-layer perceptron (MLP) the variable regions of AFs are matched to TF by adaptation of biases b_i. But in ELM biases are not adapted, so their initial values generated by random are very important. They should be matched to II.

To illustrate this better let us consider approximation of another TF of the form:

$$g(x) = \sin(20 \cdot e^x) \cdot x^2, \quad (11)$$

for $x \in [0, 1]$. The complexity of this function increases along the interval of [0, 1]. At the left border of this interval TF is flat, while at the right border its variability is the highest. The training set includes 5000 points (x_i, y_i), where x_i are uniformly randomly distributed on [0, 1] and y_i are distorted by adding the uniform noise distributed in [–0.2, 0.2]. The testing set is created similarly but without noise. The outputs are normalized into the range [–1, 1]. The ELM is composed of 20 hidden neurons with sigmoid AFs. The input weights and biases are generated randomly from the default intervals ([–1, 1] for weights and [0, 1] for biases).

In the first example the inputs are normalized to the range of [0, 10]. In this II the most variability of AFs is at the left border, whereas the most variability of TF is at the right border. Fig. 5 shows FC, AFs and $s_{\%}(x)$ in this case. FC fluctuates in the flat part of TF and is underfitted in the complex part. More neurons in the hidden layer (up to 1000) does not improve the result. When the variability region of AFs correspond to the variability region of TS the fit is much better. This occurs when II, to which inputs are normalized, is [–10, 0]. This case is shown in Fig. 6. Here flat parts of AFs correspond to flat part of TF, and complex part of TF is modeled using steeper parts of AFs.

For comparison in Fig. 7 the results for MLP are shown. The number of hidden neurons and their AF type were the

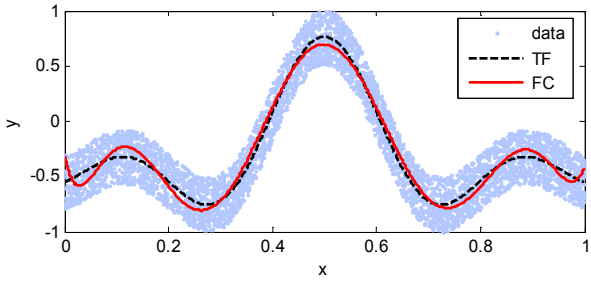

Fig. 1. Results of function (5) approximation in II of [0, 1].

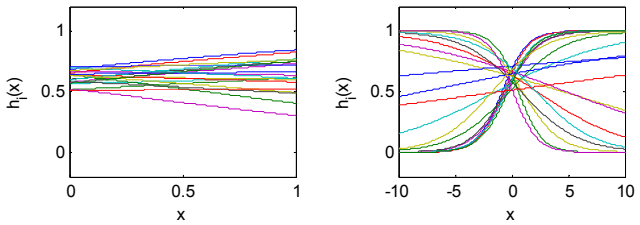

Fig. 2. The fragments of AFs in the IIs: [0,1] and (–10, 10).

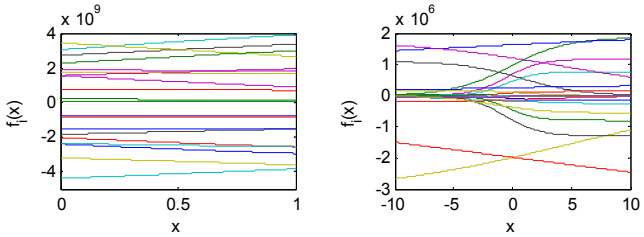

Fig. 3. The fragments of the weighted AFs in the IIs: [0,1] and (–10, 10).

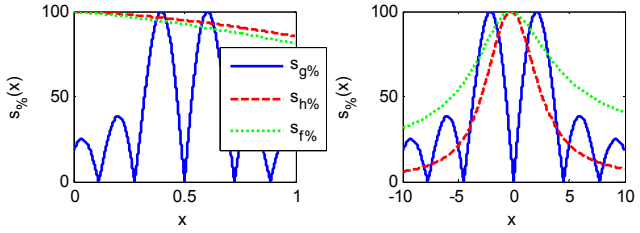

Fig. 4. The percentage slope functions in IIs: [0,1] and (–10, 10).

same as for ELM. The output neuron was linear. The MLP was learned using the Levenberg-Marquardt algorithm (Matlab implementation was used).

As can be seen from Fig. 7 the fit is better than for ELM. The weighted AF variability $s_{f\%}(x)$, which is adjusted by learning input and output weights and biases, correspond to the TF variability. The resulting AFs are evenly distributed in the II. When we changed the II, this distribution and final FC were similar. The biases of the hidden neurons adapt to the II. All AFs presented in Fig. 7 have the common feature, they rapidly pass from 0 to 1. This is independent on II and even on the TF, if the number of neurons is high. The pattern of AF distribution in II differs from these ones for ELM (compare $h_i(x)$ plots in Fig. 2, 5, 6 and 7). Let us copy this pattern into ELM. Let us assume that:

- II is constant: [0, 1],
- the input weights are constant: even-numbered neurons have $a_i = +10$ and odd-numbered ones have $a_i = -10$,
- AFs (sigmoids) are evenly distributed in II.

To distribute AFs in II we assume for a moment that all input weights are +10 and that the value of the first AFs in $x = 0$ (left border) is equal to $q = 0.99$:

$$h_1(0) = \frac{1}{1+\exp(-(a_1 \cdot 0 + b_1))} = q = 0.99 . \qquad (12)$$

From this equation we get $b_1 \approx 4.6$. Then we assume that the value of the last AFs in $x = 1$ (right border) is equal to $1 - q = 0.01$:

$$h_L(1) = \frac{1}{1+\exp(-(a_L \cdot 1 + b_L))} = 1 - q = 0.01 . \qquad (13)$$

From (13) we get $b_L \approx -4.6 - a_L = -14.6$. The biases of the successive neurons are evenly distributed in the interval $[b_L, b_1]$. Thus the i-th neuron bias is:

$$b_i = b_1 + (i-1)\frac{b_L - b_1}{L-1} . \qquad (14)$$

Now we put minus before biases of the odd-numbered neurons having negative input weights. The resulting distribution of AFs in Fig. 8 is shown. This AF pattern is used in ELM for function (11) approximation. The variability of AFs is high and uniform in the whole II ($s_{h\%}(x)$ = const = 100%). This allows ELM to approximate the complex functions. But in the case of our TF (11) having changing complexity, in the flat part the fluctuations of the FC are observed. This excessive variability of ELM at the left border was not reduced by output weights as in the case of MLP (compare $s_{f\%}(x)$ functions in Fig. 8 and 7). But when the number of hidden neurons increases, the FC improves. In Fig. 9 results for 100 neurons are presented.

Based on the experiences of the above described simulations, in the last experiment we try to select the intervals for random input weights and biases in ELM. First let us assume that II is constant: [0, 1]. So for each approximation problem the inputs are normalized into this range. To ensure high variability of AFs in II let us assume that the input weights a_i are generated randomly within the range [–20, 20] from the uni-

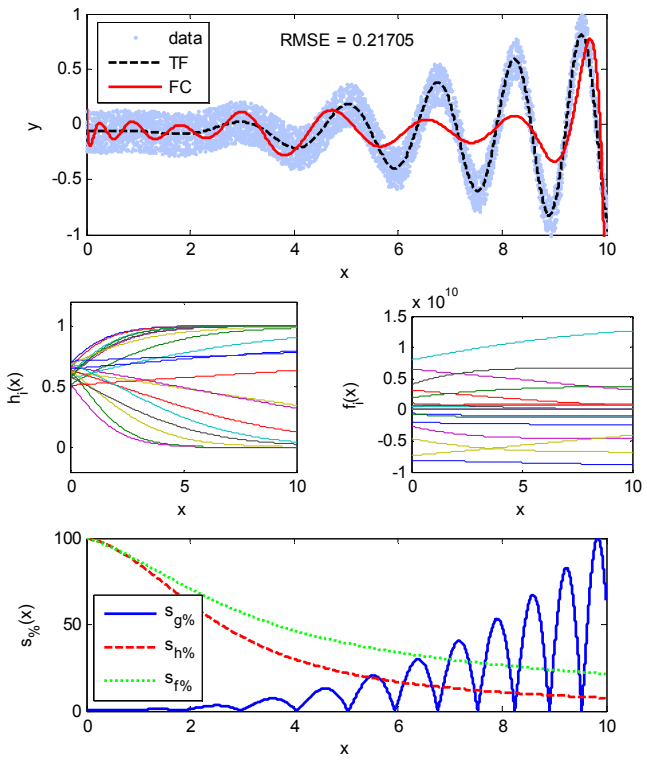

Fig. 5. Results of function (11) approximation in II of [0,10].

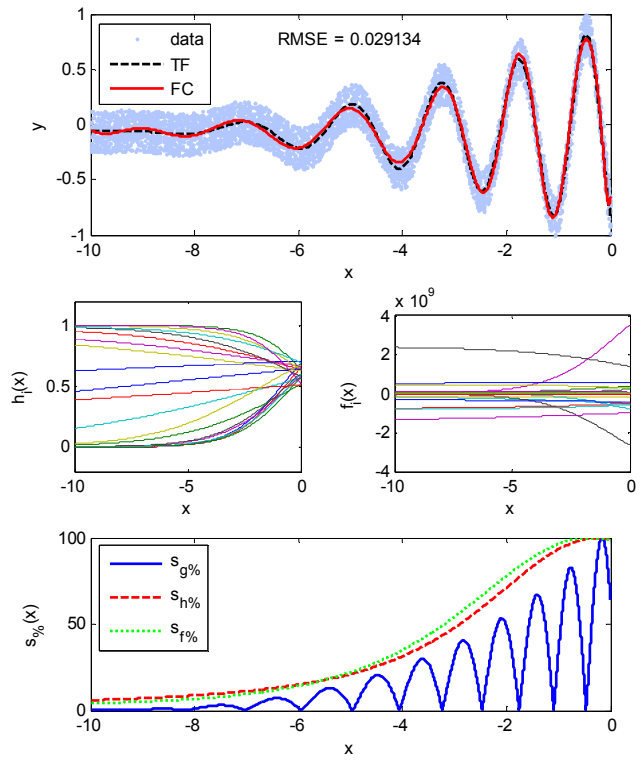

Fig. 6. Results of function (11) approximation in II of [–10, 0].

form distribution. (Larger absolute values of weights means greater AF slopes; for $a_i = 0$ we get the completely flat AF.) Biases are generated randomly from uniform distribution in such a way to ensure variability of each AF in II. Therefore, we assume that the minimum value of the sigmoid AF should be not greater than $q \in [0.5, 1]$, and its maximum value should be no less than $1 - q$:

$$\min_{x\in \mathrm{II}}(h_i(x)) \le q \text{ and } \max_{x\in \mathrm{II}}(h_i(x)) \ge 1-q, i = 1,2,...,L \,. \quad (15)$$

For sigmoids with positive weights a_i, the minimum is in the left border of II, and maximum is in the right border, so:

$$h_i(0) = \frac{1}{1+\exp(-(a_i \cdot 0 + b_i))} \le q \,, \quad (16)$$

$$h_i(1) = \frac{1}{1+\exp(-(a_i \cdot 1 + b_i))} \ge 1-q \,. \quad (17)$$

From the first inequality we get the upper limit of the range from which the biases are randomly generated. It is the same for all neurons. The lower limit obtained from the second inequality is dependent on the neuron input weight. The range of the bias for the i-th neuron having the positive weight a_i is:

$$b_i \in \left[-\ln\left(\frac{q}{1-q}\right) - a_i, -\ln\left(\frac{1-q}{q}\right) \right], \text{if } a_i \ge 0 \,. \quad (18)$$

For sigmoids with negative weights a_i the minimum is in the right border, and maximum is in the left border of II. Similar considerations as above lead to the range of the bias value for the i-th neuron having the negative weight:

$$b_i \in \left[-\ln\left(\frac{q}{1-q}\right), -\ln\left(\frac{1-q}{q}\right) - a_i \right], \text{if } a_i < 0 \,. \quad (19)$$

The lower value of the threshold q reduces the introduction of the flat fragments of AFs into II. The closer q-value to 1, the more flat parts of AFs is introduced into II. This is at the expense of smaller number of the steep fragments. This is illustrated in Fig. 10, where the slope function (6) for different q is shown. Note that for smaller q the AF slopes at the borders of II are much lower than in its central part.

The distribution of twenty AFs generated randomly according to the above rules for $q = 0.9$ in Fig. 11 is shown. Note higher variability of AFs in II than outside this interval.

Results of function (11) approximation for randomly generated input weights from the range [–20, 20], and biases from the ranges (18) and (19) in Fig. 12 are shown. These results, FC and RMSE, are very similar to the case presented in Fig. 9, where input weights and biases were initiated deterministically. The answer to the question which method of the input weights and biases is better requires further study.

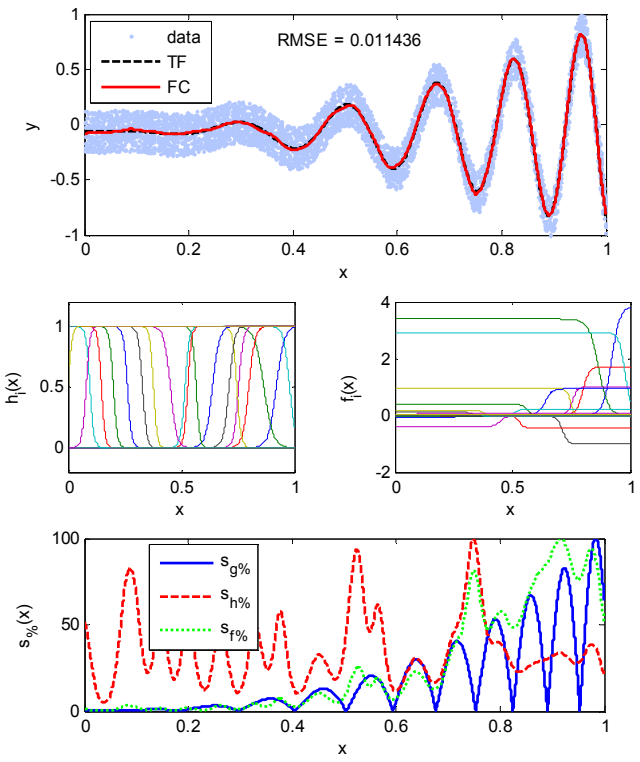

Fig. 7. Results of function (11) approximation by MLP.

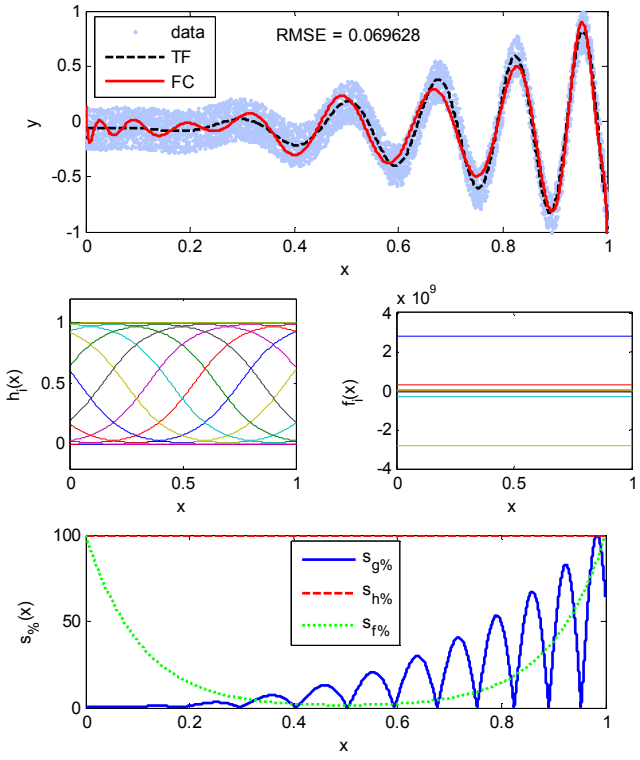

Fig. 8. Results of function (11) approximation by ELM with 20 evenly distributed neurons.

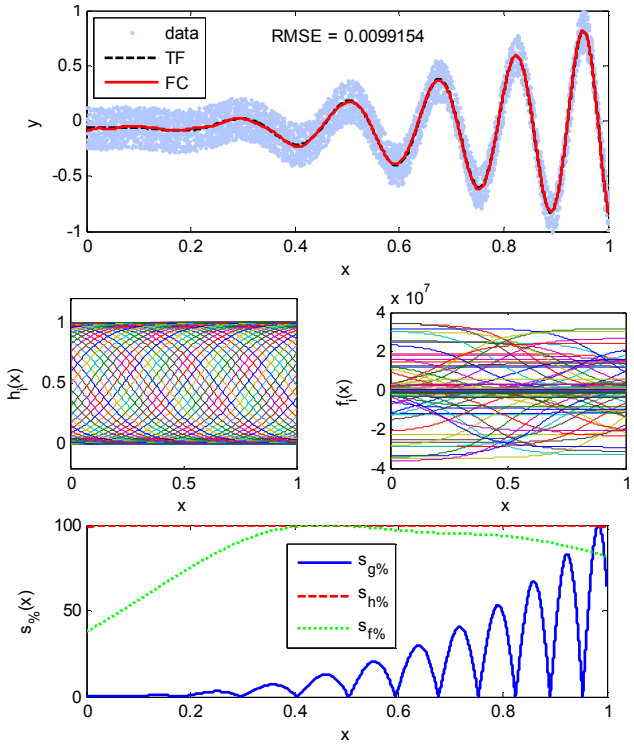

Fig. 9. Results of function (11) approximation by ELM with 100 evenly distributed neurons.

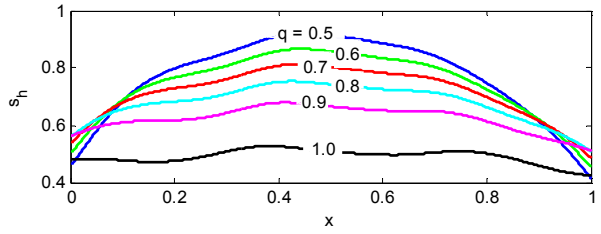

Fig. 10. The slop functions (6) for different values of q.

IV. Conclusions

The performance of ELM is sensitive to the localization of the AF variability region with respect to the input interval. This is because the AFs are fixed and cannot be modified by adaptation of input weights and biases as in the case of MLP. In ELM when the variability of the AFs does not correspond to the complexity of the target function adverse effects can be seen such as fluctuation of the fitted curve in the flat parts of the target function or underfitting in its complex parts.

When we have no information about the localization of the flat and complex parts of the target function or this function is of the same complexity in the whole range, we should provide the AFs which variability cover the input interval, i.e. the mean variability of AFs is similar in the whole interval. In such a case the AFs (working as the basis functions) are able to construct by linear combination (this is performed by the output neuron) the output function which approximates the target function with acceptable error.

In this work we recommend the ranges from which the input weights and biases should be randomly generated for one

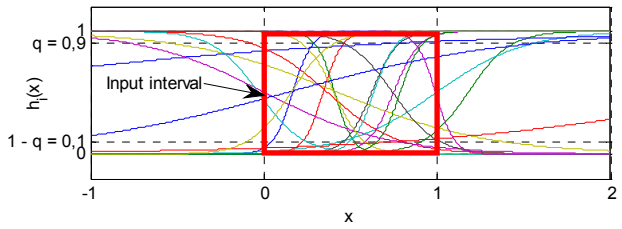

Fig. 11. The distribution of 20 AFs for q = 0.9.

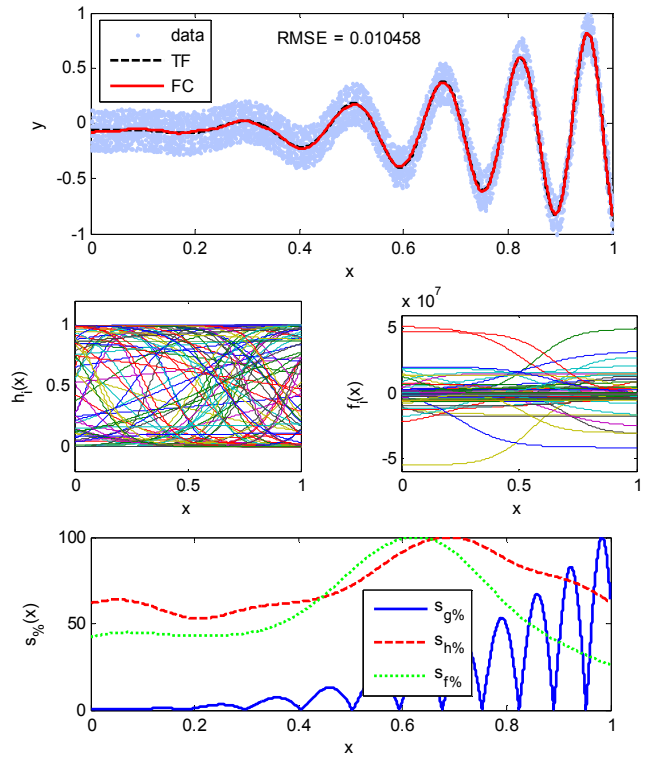

Fig. 12. Results of function (11) approximation by ELM with 100 neurons randomly distributed according to the proposed rules.

variable function approximation. The proposed range for input weights ensures different slopes of AFs and the ranges for biases (determined individually for each neuron) ensure appropriate shifts of AFs, so that the variability of AFs in the input interval was the highest.

References

[1] G.-B. Huang, Q.-Y. Zhu and C.-K. Siew, "Extreme Learning Machine: Theory and Applications", Neurocomputing, vol. 70, pp. 489-501, 2006.

[2] G.-B. Huang, L. Chen and C.-K. Siew, "Universal Approximation Using Incremental Constructive Feedforward Networks with Random Hidden Nodes", IEEE Transactions on Neural Networks, vol. 17, no. 4, pp. 879-892, 2006.

[3] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme Learning Machine for Regression and Multiclass Classification," IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, vol. 42, no. 2, pp. 513-529, 2012.

[4] G. Huang, G.-B. Huang, S. Song, and K. You, "Trends in Extreme Learning Machines: A Review," Neural Networks, vol. 61, no. 1, pp. 32-48, 2015.