Extreme Learning Machine as A Function Approximator: Initialization of Input Weights and Biases

Grzegorz Dudek

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

Abstract. Extreme learning machine is a new scheme for learning the feedforward neural network, where the input weights and biases determining the nonlinear feature mapping are initiated randomly and are not learned. In this work we analyse approximation ability of the extreme learning machine depending on the activation function type and ranges from which input weights and biases are randomly generated. The studies are performed on the example of approximation of one variable function with varying complexity. The ranges of input weights and biases are determined for ensuring the sufficient flexibility of the set of activation functions to approximate the target function in the input interval.

Keywords: extreme learning machine, function approximation, activation functions, feedforward neural networks.

1 Introduction

Feeforward neural networks (FNNs) have been successfully applied to solve many complex and diverse tasks. They are widely used in regression and classification problems due to their adaptive nature and excellent approximation properties (FFN is an universal approximator, i.e. it is capable of approximating any nonlinear function). As a learning machine FNN can learn from observed data and generalize well in unseen examples. All inner parameters of the networks (weights and biases) are adjustable. Due to the layered structure of FNN the learning process is complicated, inefficient and requires the activation functions of neurons to be differentiable. The training usually employ some form of gradient descent method, which is generally time-consuming and converges to local minima. Moreover some parameters, such as number of hidden neurons or learning algorithm parameters, have to be tuned manually.

The Extreme Learning Machine (ELM) is an alternative learning algorithm proposed for training single-hidden-layer FNNs [1]. The learning process does not require iterative tuning of weights. The input weights (linking the inputs with hidden layer) and biases of hidden neurons need not to be adjusted. They are randomly initiated according to any continuous sampling distribution without the knowledge of the training data. The only parameters need to be learned

are the output weights (between the hidden and output layers). Thus 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. As theoretical studies have shown, even with randomly generated hidden nodes, ELM with wide type of activation functions can work as an universal approximator. Numerous experiments and applications have demonstrated that ELM and its variants are efficient, accurate and easy to implement. The learning speed of ELM can be thousands of times faster than traditional gradient descent-based learning.

In this work we analyze approximation ability of ELM depending on the activation function type and ranges from which input weights and biases are randomly generated. To visualize results the studies are performed on the example of approximation of one variable function with varying complexity. The ranges of input weights and biases are determined for ensuring the sufficient flexibility of ELM in the input interval.

2 Basic Extreme Learning Machine

ELM originally proposed by Huang et al. [1] learns in three steps. Given a training set $\Phi = \{(\mathbf{x}_k, t_k) \mid (\mathbf{x}_k \in \mathbb{R}^n, t_k \in \mathbb{R}, k = 1, 2, ..., N\}$, hidden node activation function type $h(\mathbf{x})$, and the number of hidden nodes L,

1. Randomly initiate according to any continuous sampling distribution hidden node parameters, i.e. input weights and biases: $\mathbf{a}_i = [a_{i,1}, a_{i,2}, ..., a_{i,n}]^T$ and $b_i, i = 1, 2, ..., L$. Usually uniform distribution is used for this: $a_{i,j} \sim \mathrm{U}(a_{\min\, i,j}, a_{\max\, i,j}), b_i \sim \mathrm{U}(b_{\min\, i,j}, b_{\max\, i,j})$. (The ranges from which weights and biases are generated, $a_{\min}, a_{\max}, b_{\min}$ and $b_{\max}$, are the main subject of this work.)
2. Calculate the hidden layer output matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & \cdots & h_L(\mathbf{x}_1) \\ \vdots & \vdots & \vdots \\ h_1(\mathbf{x}_N) & \cdots & h_L(\mathbf{x}_N) \end{bmatrix}, \tag{1}$$

where $h_i(\mathbf{x})$ is an activation function of the i-th neuron, which is nonlinear piecewise continuous function, e.g. the sigmoid:

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

$\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of $\mathbf{a}_i$ and $\mathbf{x}$.
The i-th column of $\mathbf{H}$ is the i-th hidden neuron output vector with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$. Hidden neurons maps the data from n-dimensional input space to the L-dimensional feature space H, and thus, $\mathbf{h}(\mathbf{x})$ is a nonlinear feature mapping. The most popular activations functions are: sigmoid, Gaussian, multiquadric, hard-limit, triangular and sine functions. Different activation functions can be used in different hidden neurons.

The output matrix $\mathbf{H}$ remains unchanged because parameters of the activation functions, $\mathbf{a}_i$ and b_i, are fixed.

3. Calculate the output weights β_i:

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

where $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_L]^T$ is the vector of the output weights, $\mathbf{T} = [t_1, t_2, ..., t_N]^T$ is the training data output matrix, and $\mathbf{H}^{+}$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$.

The above equation for $\boldsymbol{\beta}$ results from the minimizing the approximation error:

$$\min \|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\| . \tag{4}$$

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

$$f_L(\mathbf{x}) = \sum_{i=1}^{L} f_i(\mathbf{x}) = \sum_{i=1}^{L} \beta_i h_i(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta}, \tag{5}$$

where $f_i(\mathbf{x}) = \beta_i h_i(\mathbf{x})$ is the weighted output of the i-th hidden node.

The output function $f_L(\mathbf{x})$ is a linear combination of the activation functions $h_i(\mathbf{x})$. Characteristically for ELM the hidden nodes parameters, $\mathbf{a}_i$ and b_i, are randomly generated instead of being explicitly trained. This process is independent of the training data and provide random feature mapping.

To improve generalization performance of ELM its regularized version was proposed [2]. Recent developments in theoretical studies and applications of ELM are reported in [3].

3 Approximation Capability of ELM: Simulation Study

In this section we analyze the approximation capability of ELM depending on the activation function types and the way of initialization of the input weights and biases. 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.

The simulation tests were performed in MATLAB R2010b environment. We used Matlab implementation of ELM: function `elm` created by the authors of ELM algorithm (downloaded from `http://www.ntu.edu.sg/home/egbhuang/elm\_random\_hidden\_nodes.html`). The input weights and biases in this implementation are generated randomly from the uniform distribution: weights from the range of $[-1, 1]$ and biases form the range of $[0, 1]$. There are five types of AFs in `elm` to choose from. Each AF gets linear combination of ELM inputs:

$\mathbf{a} \cdot \mathbf{x} + b$ as an argument. The coefficients of this combination are input weights and bias of the i-th neuron. The types of AFs implemented in `elm` function are shown in Table 1 in the single input version.

In [4] we analyse the impact of ranges from which the input weights and biases are randomly generated on the fitted curve complexity when sigmoid AFs are used. In this work we consider ELM with other AFs. To illustrate results the single variable TF is used of the form:

$$g(x) = \sin(20 \cdot e^x) \cdot x^2, \tag{6}$$

where $x \in [0, 1]$.

The complexity of function (6) increases along the interval $[0, 1]$. TF is flat at the left border of the interval, while at the right border its variability is the highest. To express TF variability we use the percentage slope function [4]:

$$s_{g\%}(x) = 100 \cdot \left| \frac{dg(x)}{dx} \right| \cdot \left(\max_{x \in \mathrm{II}} \left| \frac{dg(x)}{dx} \right| \right)^{-1}. \tag{7}$$

The training set includes 5000 points (x_k, y_k), where x_k 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 $[-1, 1]$. These settings are the same as in [1], where ELM performance was evaluated on the SinC function benchmark.

In Fig. 1 the results of approximation using ELM with 20 hidden neurons with Gaussian AFs are shown. As can be seen from this figure the 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. If we look at the fragments of AFs, $h_i(x)$, in the input window (II), we notice their low variability, which does not correspond to the variability of TF. The AF fragments in II compose the set of basis functions from which the FC is constructed by their linear combination. The components of this combination, $f_i(x)$, i.e. AFs weighted by the output weights β_i, are also shown in Fig. 1. To measure the variability of the set of AFs we use the percentage slope functions of AFs, $s_{h\%}(x)$, and the weighted AFs, $s_{f\%}(x)$, defined as follows [4]:

$$s_{h\%}(x) = \frac{100 \cdot s_h(x)}{\max_{x \in \mathrm{II}} s_h(x)}, \text{ where } s_h(x) = \frac{1}{L} \sum_{i=1}^{L} \left| \frac{dh_i(x)}{dx} \right|, \tag{8}$$

$$s_{f\%}(x) = \frac{100 \cdot s_f(x)}{\max_{x \in \mathrm{II}} s_f(x)}, \text{ where } s_f(x) = \frac{1}{L} \sum_{i=1}^{L} \left| \frac{df_i(x)}{dx} \right| = \frac{1}{L} \sum_{i=1}^{L} \left| \beta_i \frac{dh_i(x)}{dx} \right| \tag{9}$$

The plots of these functions, expressing the AF set variability along II, in Fig. 1 are shown, together with function (7) expressing TF variability. Note that the highest variability of the AF set is at the left border of II, whereas the highest variability of TF is at the right border.

To improve approximation capability of ELM let us increase the AF set variability in the II of $[0, 1]$. This will be reached in two ways:

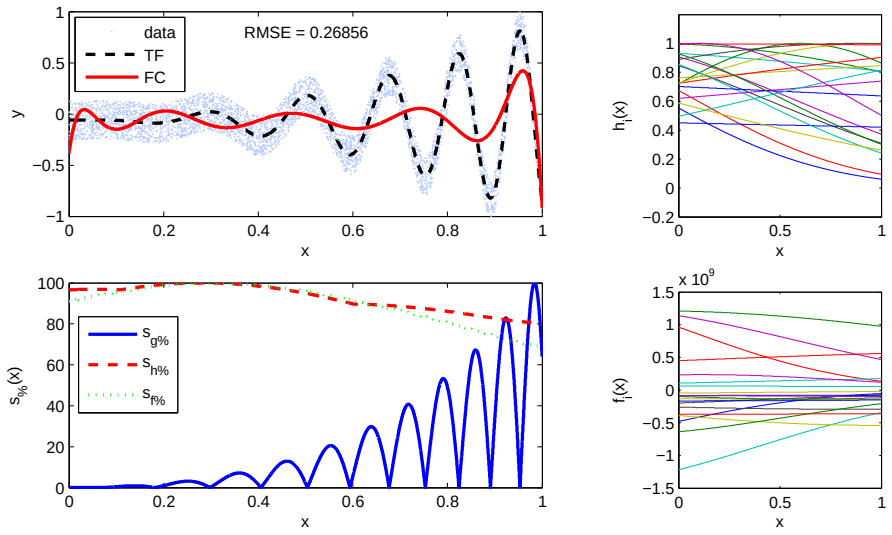

Fig. 1. Results of approximation using ELM with 20 Gaussian neurons generated from the default intervals

- by increasing the input weights determining the slopes of the Gaussian AFs and
- by adjusting the biases to the II so that the maxima of AFs are inside the II.

The first requirement is not very important because the AF slopes are regulated by output weights (see (9)). But to avoid large output weights necessary for providing steep weighted AFs to model the steep TF fragments, let us assume that $a_i \in [0, 10]$. (For one input case a_i can be limited to the positive values.)

According to the second requirement the maximum of AF should be for $x \in [0, 1]$. When the maximum is in the left border of our II we get:

$$h_i(0) = \exp(-(a_i \cdot 0 + b_i)^2) = 1 \ \rightarrow \ b_i = 0, \tag{10}$$

and when it is in the right border we get:

$$h_i(1) = \exp(-(a_i \cdot 1 + b_i)^2) = 1 \ \rightarrow \ b_i = -a_i. \tag{11}$$

Thus the bias of the i-th neuron should be randomly generated within the range:

$$b_i \in [-a_i, 0], \text{where } a_i \geq 0. \tag{12}$$

For the lowest value of the input weight $a_i = 0, b_i = 0$ and AF is a constant function: $h_i(x) = 1$. The higher the value of a_i, the steeper the AF is.

The results of approximation for weights and biases randomly generated in the intervals proposed above in Fig. 2 are presented. Here 100 Gaussian neurons are used in the hidden layer (RMSE for 20 neurons was 0.088). Note higher variability of AFs in the II than outside this interval. The slope function $s_{f\%}(x)$

corresponds better to the variability of TF than in the previous example. Similar results were achieved when input weights were all set to constant values $a_i = 5$, and biases were uniformly distributed in the interval $[-a_i, 0]$. This is illustrated in Fig. 3.

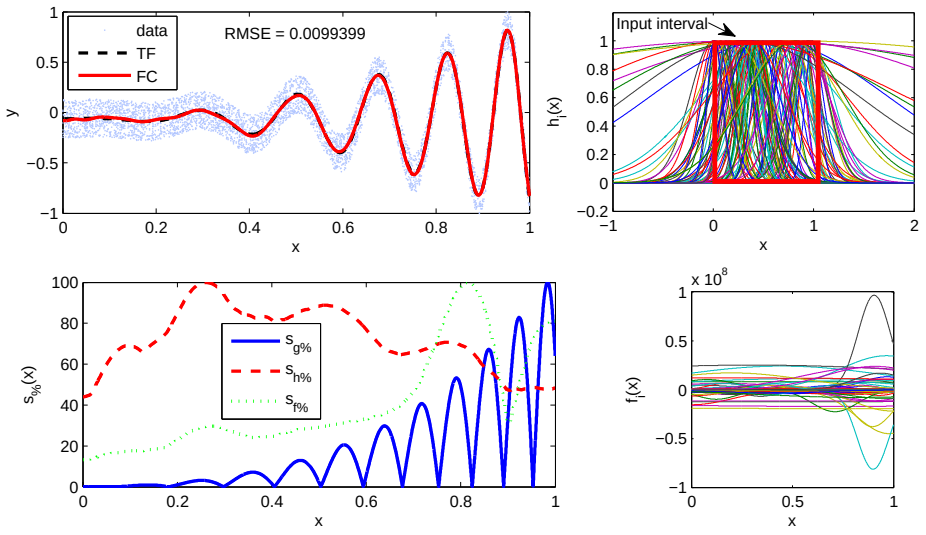

Fig. 2. Results of approximation using ELM with 100 Gaussian neurons generated from the proposed intervals

In the next experiment we replace Gaussian AFs by triangular AFs. When using default intervals for random generation of input weights and biases ($[-1, 1]$ and $[0, 1]$, respectively) the results are not satisfactory. The problem of underfitting appears. To improve approximation capability in this case we use the same approach as for Gaussian AFs. First we increase the input weight values defining their range as $[0, 10]$. Then we assume that the maxima of AFs are in II. This leads to the same range for biases (12) as in the case of Gaussian AFs. Results of approximation in Fig. 4 are shown. Note that FC is piecewise linear and unsmooth. Combining triangular basis functions results in the "jagged" FC. When instead of random weights and biases constant input weights were used ($a_i = 5$) and biases were uniformly distributed in $[-a_i, 0]$, the results were similar (RMSE = 0.016).

In the case of hard-limit AFs the FC is a linear combination of unit step functions. When the step position (jump from 0 to 1 or vice versa) is outside the II we get constant fragment of AF in the II. Such fragments are useless for modeling TF fragments of nonzero slopes. When using default settings for ranges of input weights and biases, many AFs have their jumps outside the II of $[0, 1]$. The jump positions can be calculated from: $x = -b/a$. For b randomly generated from the uniform distribution on the interval $[0, 1]$ and a generated similarly on $[-1, 1]$ about 75% of AFs have jumps outside our II. To bring the jumps into II

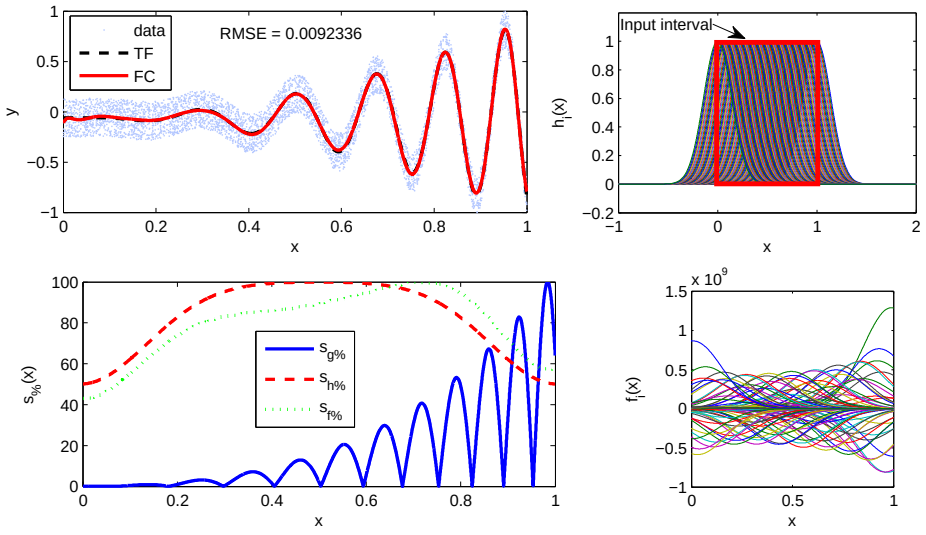

Fig. 3. Results of approximation using ELM with 100 Gaussian neurons evenly distributed in the II

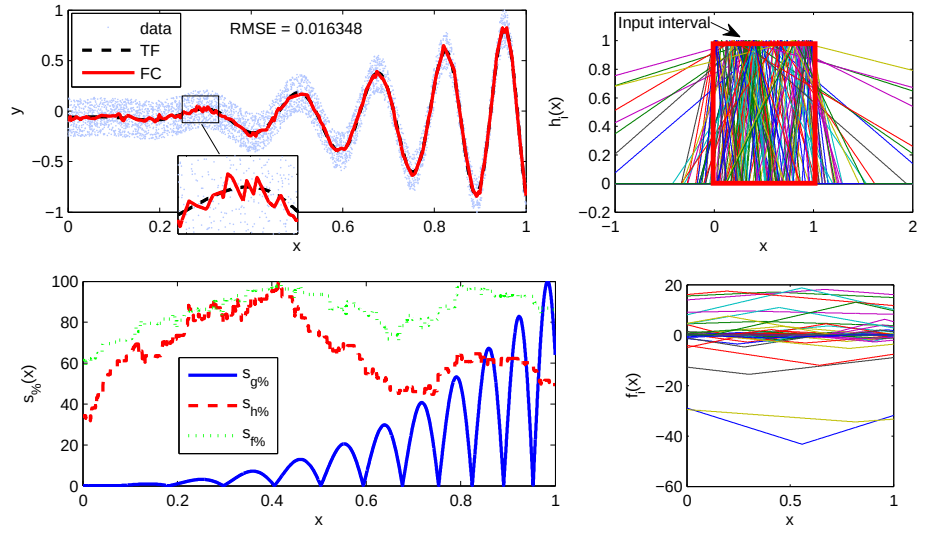

Fig. 4. Results of approximation using ELM with 100 triangular neurons generated from the proposed intervals

the biases should be randomly generated from the intervals: $[-a_i, 0]$, if $a_i \geq 0$ or $[0, -a_i]$, if $a_i < 0$. The value of a_i is not important in this case. (In other types of AFs presented in Table 1, a_i regulates the slope of AF, but not in hard-limit function). Only its sign deciding about the hard-limit function direction, i.e. from 0 to 1 or from 1 to 0, is important. So we can assume $a_i \in \{-1, 1\}$, and draw its value with the same probability. In Fig. 5 the results of approximation using ELM composed of 100 hidden neurons with hard-limit AFs are presented. Note that FS is a step function. When we use constant input weights $a_i = +1$ for even-numbered neurons and $a_i = -1$ for odd-numbered ones, and biases evenly distributed in $[-1, 0]$, for $a_i = +1$ or in $[0, 1]$, for $a_i = -1$, the results were similar (RMSE = 0.046).

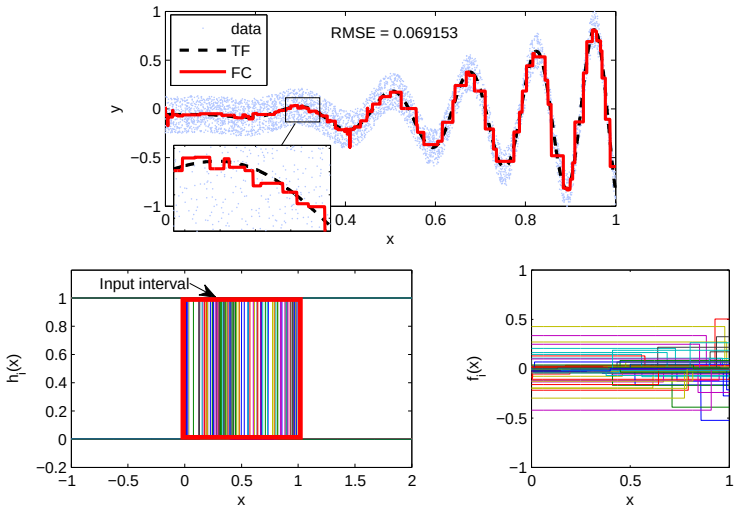

Fig. 5. Results of approximation using ELM with 100 hard-limit neurons generated from the proposed intervals

Now we test ELM with sine functions as AFs. For default ranges for input weights and biases the results of approximations look similar to results for Gaussian AFs presented in Fig. 1. To increase variability of AFs in the II we change the bounds of the interval from which input weights are generated to $[0, 30]$. The biases are generated from the range $[0, 2\pi]$. This range for b_i ensures uniform distribution of a single sine AF in II. In such a case, at each point x from II, each value of the AF from its codomain is achievable. The results of approximation for ELM with 100 sine AFs with parameters randomly generated from the above intervals in Fig. 6 are shown. Note that due to the periodicity of AFs their variability is the same inside and outside the II. When instead of random weights and biases we use constant input weights and biases evenly distributed in $[0, 2\pi]$, the resulting FC does not fit to TF. It has a form of a sine function with changing amplitude, and with decreasing period with a_i.

The ranges from which input weights and biases should be randomly generated for a single variable function approximation in Table 1 are summarized. Sigmoid AF was analyzed in [4]. The border value A of the interval for input weights depends on the AF type and variability of the TF. When TF is flat, lower border values can be used. This prevents overfitting. For TF with high variability, higher values of A are needed to prevent underfitting. Generally, parameter A controls bias-variance tradeoff of the ELM. In all cases except sine AF the intervals for biases are dependent on the input weights. So for each neuron first input weight is randomly chosen, and next bias is randomly generated from the appropriate interval.

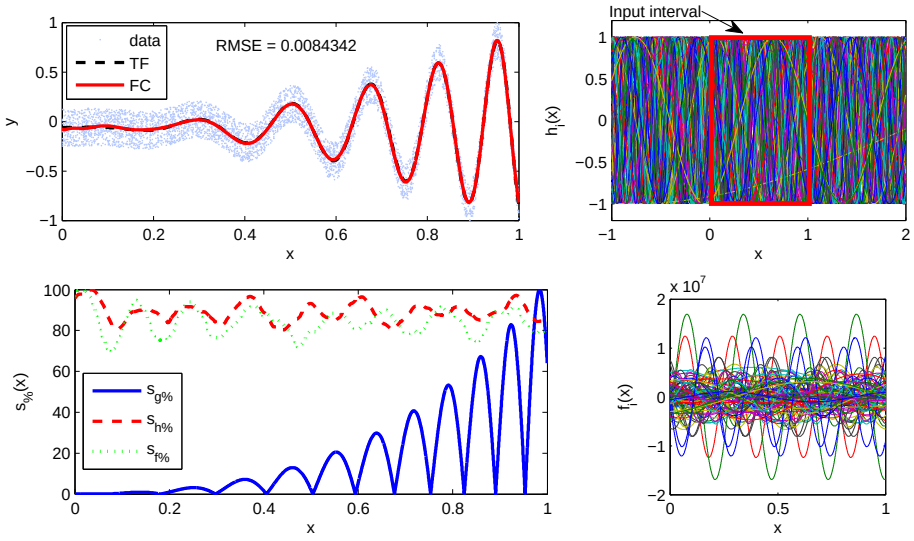

Fig. 6. Results of approximation using ELM with 100 sine neurons generated from the proposed intervals

4 Conclusions

The fitted curve in ELM is a linear combination of the basis functions, i.e. the activation functions of the hidden neurons. The basis function is a simple nonlinear piecewise continuous function which parameters are randomly generated in ELM. The set of basis functions should have the sufficient flexibility to ensure the best fitting to the target function in the input interval. In the classical learning scheme, such as gradient descent-based learning, the input weights and biases are adjusted during learning. This results in the modifications of the basis functions: they change their slopes and slide along the x-axis. So the flexibility of the set of basis functions is adapted to the complexity of the target function. In ELM such a mechanism does not work. Therefore, the ELM designer should ensure the flexibility of the basis function set in the input interval.

Table 1. Activation functions and ranges for their parameters

Activation function	Weights $a_i \in$	Biases $b_i \in$
Sigmoid $[1+\exp(-(a_i x+b_i))]^{-1}$	$[-A, A]$	$\left[-\ln\left(\frac{q}{1-q}\right)-a_i, -\ln\left(\frac{1-q}{q}\right)\right]$, if $a_i \geq 0$ $\left[-\ln\left(\frac{q}{1-q}\right), -\ln\left(\frac{1-q}{q}\right)-a_i\right]$, if $a_i < 0$
Gaussian $\exp(-(a_i x+b_i)^2)$	$[0, A]$	$[-a_i, 0]$
Triangular $\begin{cases} 1-\lvert a_i x+b_i\rvert, & \text{if } \lvert a_i x+b_i\rvert \leq 1 \\ 0, & \text{otherwise} \end{cases}$	$[0, A]$	$[-a_i, 0]$
Hard-limit $\begin{cases} 1, & \text{if } a_i x+b_i \leq 0 \\ 0, & \text{otherwise} \end{cases}$	$\{-1, 1\}$	$[-1, 0]$, if $a_i = 1$ $[0, 1]$, if $a_i = -1$
Sine function $\sin(a_i x+b_i)$	$[0, A]$	$[0, 2\pi]$

where: $A > 0, q \in [0.5, 1]$ (see [4]).

The aim of this work is to find the ranges from which basis function parameters should be randomly generated. These ranges are dependent on the basis function type, moreover, the ranges for biases are dependent on the input weights. In the future work the results achieved here for approximation of the single argument function we will try to generalize for multiple argument functions. It is worth examining also the ability of ELM generalization depending on the input weight ranges determining the slopes of the activation functions.

References

1. Huang G.-B., Zhu Q.-Y. and Siew C.-K.: Extreme Learning Machine: Theory and Applications. Neurocomputing 70, 489-501 (2006)
2. Huang G.-B., Zhou H., Ding X., and Zhang R.: Extreme Learning Machine for Regression and Multiclass Classification. IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics 42(2), 513-529 (2012)
3. Huang G.,. Huang G.-B, Song S., and You K.: Trends in Extreme Learning Machines: A Review. Neural Networks 61(1), 32-48 (2015)
4. Dudek G.: Extreme Learning Machine for Function Approximation - Interval Problem of Input Weights and Biases. Proc. 2nd IEEE International Conference on Cybernetics CYBCONF 2015 (in print)