Information Sciences 481 (2019) 33–56

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Generating random weights and biases in feedforward neural networks with random hidden nodes

Grzegorz Dudek

Faculty of Electrical Engineering, Czestochowa University of Technology, 17 Armii Krajowej Ave., 42–200 Czestochowa, Poland

ARTICLE INFO

Article history:
Received 12 June 2018
Revised 23 December 2018
Accepted 27 December 2018
Available online 30 December 2018

Keywords:
Activation functions
Function approximation
Feedforward neural networks
Neural networks with random hidden nodes
Randomized learning algorithms

ABSTRACT

Neural networks with random hidden nodes have gained increasing interest from researchers and practical applications. This is due to their unique features such as very fast training and universal approximation property. In these networks the weights and biases of hidden nodes determining the nonlinear feature mapping are set randomly and are not learned. Appropriate selection of the intervals from which weights and biases are selected is extremely important. This topic has not yet been sufficiently explored in the literature. In this work a method of generating random weights and biases is proposed. This method generates the parameters of the hidden nodes in such a way that nonlinear fragments of the activation functions are located in the input space regions with data and can be used to construct the surface approximating a nonlinear target function. The weights and biases are dependent on the input data range and activation function type. The proposed methods allows us to control the generalization degree of the model. These all lead to improvement in approximation performance of the network. Several experiments show very promising results.

1. Introduction

Feedforward neural networks (FNN) are extensively used in regression and classification applications due to their adaptive nature and universal approximation property. FNNs are able to learn from observed data and generalize well in unseen examples. The FNN inner parameters, i.e. weights and biases, are adjustable in the learning process. But due to the layered structure of the network this process is complicated, inefficient and requires the activation functions (AFs) of neurons to be differentiable. The training algorithms which involves the optimization of non-convex objective function, usually employ some form of gradient descent method which are known to be time consuming, sensitive to initial values of parameters and converging to local minima. Moreover some parameters, such as number of hidden nodes or learning algorithm parameters, have to be tuned manually.

In recent years, alternative learning methods have been developed, in which the network parameters are selected randomly, so that the resulting optimization task becomes convex and can be formulated as a linear least-squares problem [11]. Such methods are applied in three broad families of NNs: FNNs, recurrent NNs, and randomized kernel approximations [12]. Many simulation studies reported in the literature show high performance of the randomized models which is compared to fully adaptable ones. Randomization which is cheaper than optimization provides for simplicity in implementation and faster training.

E-mail address: dudek@el.pcz.czest.pl

https://doi.org/10.1016/j.ins.2018.12.063
0020-0255/

In feedforward neural networks with random hidden nodes (FNNRHN), the learning process does not require iterative tuning of weights. The weights and biases of hidden neurons need not to be adjusted. They are randomly selected from some intervals according to any continuous sampling distribution and remain fixed. The only parameters that are learned are the output weights, linking the hidden and output nodes. Thus, FNNRHN can be considered as a linear system in which the output weights are analytically determined through simple generalized inverse operation of the hidden layer output matrices. For this reason, the learning speed can be thousands of times faster than classical gradient descent-based learning. As theoretical studies have shown [5], when the parameters of the hidden nodes are randomly generated from a uniform distribution within a proper range, the resulting neural network is a universal approximator for a continuous function on a bounded finite dimensional set with efficient convergence rate. Husmeier in [4] proved that the universal approximation property also holds for symmetric interval setting of the random parameter scope if the function to be approximated meets Lipschitz condition. However, how to select the range for the random parameters remains an open question. This issue is considered to be one of the most important research gaps in the field of randomized algorithms for training NNs. In applications of FNNRHNs to classification or regression problems the ranges for random parameters of hidden nodes are selected without scientific justification and could not ensure the universal approximation property of the network. Usually these intervals are assigned as fixed, typically $[-1, 1]$, regardless of the data and the AF type. Independency of hidden neurons on data is seen as an asset.

In some papers we can find some suggestions on how to generate random parameters of hidden neurons. In the early work on FNNs with randomization [9], the parameters of hidden nodes were set to be uniform random values in $[-1, 1]$, but authors suggest to optimize this range in a more appropriate range for the specified application. In [4] the author suggests to use symmetric and "large enough" boundaries for the hidden node parameters and advices to optimize them in the training process. More details on generating random parameters of hidden nodes in [1] are given. The weights are chosen from a normal distribution with zero mean and some specified variance that can be adjusted to obtain input-to-node values that do not saturate the sigmoids. The biases are computed to center each sigmoid at one of the training points. This distributes the sigmoids across the input space, as is suggested by the Nguyen–Widrow weight initialization algorithm [8].

The problem with selection of appropriate ranges for random parameters of the hidden nodes is not solved until today. The authors of new solutions in NNs with randomization do not give any hints on the ranges for random parameters [15]. However, in many works concerning FNNRHNs attention is drawn to the significance of the intervals from which random weights and biases are selected. In conclusion of [2] it is rightly pointed out that when network nodes are chosen at random and not subsequently trained, they are usually not placed in accordance with the density of the input data. In such a case training of linear parameters becomes ineffective at reducing errors.

Moreover, the number of nodes needed to approximate a nonlinear map grows exponentially, and the model is very sensitive to the random parameters. To improve effectiveness of the network the authors of [2] advice combining unsupervised placement of network nodes according to the input data density with subsequent supervised or reinforcement learning values of the linear parameters of the approximator. This work motivated the authors of [7] to highlight some risky aspects caused by the randomness in FNNRHN, such as the illogical way of simply selecting a trivial range $[-1, 1]$ for random assignment of the input weights and biases. They analyze some impacts of the scope of random parameters on the model performance, and empirically show that a widely used setting for this scope is misleading. Although, they observe that for some specific scopes the network performs better in both learning and generalization than in other scopes, they do not give tips on how to select appropriate scopes. There is no such tips also in [13], where authors investigate the range for random parameters by introducing a scaling factor to control this range. The work is concluded that scaling down the randomization range to avoid saturating the neurons may risk at degenerating the discrimination power of the random features. Scaling this range up to enhance the discrimination power of the random features may risk saturating the neurons.

Lately, Wang and Li proposed a supervisory mechanism of assigning the input weights and biases of the hidden nodes in their learner model generated incrementally by stochastic configuration algorithms [14]. The random parameters are generated with an inequality constraint adaptively selecting the scope for them, ensuring the universal approximation property of the model. The authors adopt from [4] the symmetric interval setting for the random parameters. The scope $[-\lambda, \lambda]$ is searched in the iterative procedure, from $\lambda = \lambda_{\min}$ to $\lambda = \lambda_{\max}$. The input weights and biases are generated both from the same symmetric interval. A method proposed in this work generate them separately depending on the data (its scope and complexity) and activation function type. In the experimental part of the work we compare the results of the proposed method and the stochastic configuration network [14].

In this paper, we look inside FNNRHN and study how the AFs of neurons compose the fitting curve and how the ranges from which weights and biases are randomly generated affect the approximation ability of the network. First we analyze simple one-dimensional cases and then we focus on multidimensional cases. A method of randomly generating FNNRHN parameters to set nonlinear fragments of AFs in the input space regions containing data points is proposed. This method allows us to control the flatness and steepness of AFs in the input hypercube and hence the degree of generalization of the network.

The remainder of the paper is organized as follows. Section 2 briefly presents FNNRHN learning algorithm. In Section 3 the intervals for random parameters are determined on the basis of theoretical analysis for one-dimensional case. The analysis were performed for four popular AFs: sigmoidal, Gaussian, softplus and sine/cosine. Similar analysis were performed for multidimensional case in Section 4 . Section 5 reports the simulation study and compare results of the proposed method with the newest results from the literature. Section 6 concludes the paper.

2. FNNRHN learning algorithm

The architecture of FNNRHN is the same as for single-hidden-layer feedforward neural network. One output is considered, m hidden neurons and n inputs. The training set is $\Phi = \{(\mathbf{x}_l, y_l) | \mathbf{x}_l \in \mathbb{R}^n, y_l \in \mathbb{R}, l = 1, 2, \ldots, N\}$ and the AF of hidden nodes is $h(x)$. The learning algorithm consists of three steps.

1. Randomly generate hidden node parameters: weights $\mathbf{a}_i = \left[a_{i,1}, a_{i,2}, \ldots, a_{i,n}\right]^T$ and biases b_i, $i = 1, 2, \ldots, m$, according to any continuous sampling distribution. Usually $a_{i,j} \sim U(a_{\min}, a_{\max})$ and $b_i \sim U(b_{\min}, b_{\max})$.
2. Calculate the hidden layer output matrix $\mathbf{H}$:

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

where $h_i(x)$ is an AF of the ith node, which is nonlinear piecewise continuous function, e.g. a sigmoid:

$$h_i(\mathbf{x}) = \frac{1}{1 + \exp\left(-\left(\mathbf{a}_i^T \mathbf{x} + b_i\right)\right)} \tag{2}$$

The ith column of $\mathbf{H}$ is the ith hidden node output vector with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$. Hidden neurons map the data from n-dimensional input space to m-dimensional feature space, and thus, $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_m(\mathbf{x})]$ is a nonlinear feature mapping. The output matrix $\mathbf{H}$ remains unchanged because parameters of the AFs, a_i and b_i, are fixed.

3. Calculate the output weights β_i:

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

where $\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_m]^T$ is a vector of output weights, $\mathbf{Y} = [y_1, y_2, \ldots, y_N]^T$ is a vector of target outputs, and $\mathbf{H}^{+}$ is the Moore–Penrose generalized inverse of matrix $\mathbf{H}$.
The above equation for $\boldsymbol{\beta}$ results from the following criterion for minimizing the approximation error:

$$\min \left\| \mathbf{H}\boldsymbol{\beta} - \mathbf{Y} \right\| \tag{4}$$

The function expressed by FNN is a linear combination of the AFs $h_i(\mathbf{x})$. In the one output case it is of the form:

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

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

The presented network is the most popular solution of FNNRHN. But it should be mentioned, that the prototype of NN with randomization, i.e. Random Vector Functional Link (RVFL) network proposed by Pao and Takefji [10], has direct links from the input layer to the output one.

3. Generating random weights and biases – one-dimensional case

For brevity, we use the following acronyms:

- TF: target function $g(x)$,
- FC: fitted curve $\varphi(x)$,
- II: input interval, i.e. the interval to which inputs are normalized.

To illustrate results the single-variable TF is used of the form:

$$g(x) = \sin\left(20 \cdot \exp(x)\right) \cdot x^2 \tag{6}$$

where $x \in [0, 1]$. A variation of this function increases along the II [0, 1] (see top chart in Fig. 2). The TF is flat at the left border of the II, while towards the right border it expresses increasing oscillations.

The training set contains 5000 points (x_l, y_l), where x_l 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 of the same size is created similarly but without noise. The outputs are normalized into the range $[-1, 1]$.

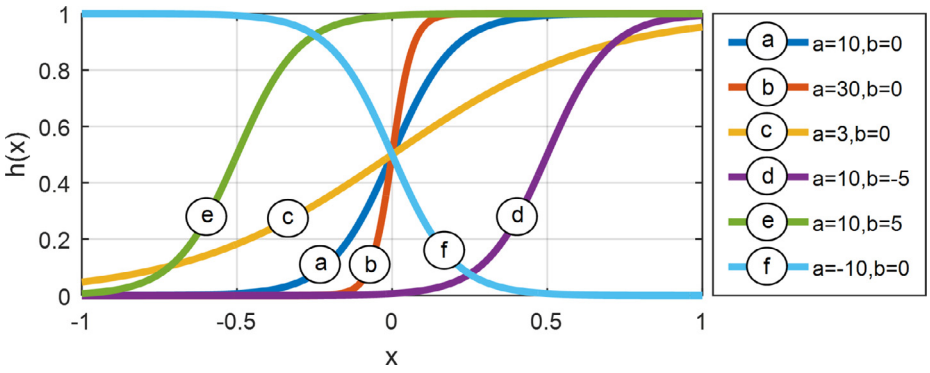

Fig. 1. A sigmoid with different parameters.

3.1. Sigmoid AFs

Let us look inside NN and analyze how the FC is constructed. Let a sigmoid be an AF of hidden neurons:

$$h(x) = \frac{1}{1 + \exp(-(a \cdot x + b))} \tag{7}$$

The weight a decides about a slope of the sigmoid and the bias b shifts the function along the x-axis (see Fig. 1). For positive a the slope of the sigmoid (dh/dx) is positive, and for negative a the slope is negative. The set of hidden neurons represents a set of AFs which are combined linearly to produce FC.

Results of curve (6) fitting when using single-hidden layer FNN with 9 hidden neurons in Fig. 2 are shown. FNN was trained using Levenberg–Marquardt algorithm [3]. The middle chart shows AFs of 9 hidden neurons with optimized parameters a_i and b_i. These parameters are determined in the learning process, as well as the output weights β_i. The bottom chart shows AFs multiplied by the output weights β_i. The sum of these curves gives FC, which is drawn with a solid line in the upper chart. Note that AFs have their nonlinear, steep fragments inside the II (shown as a gray field). These fragments are used to compose a FC. When the TF expresses complex behavior, such as function (6), AFs should be distributed in the II in such a way that their steep fragments correspond to the steep fragments of the TF.

Now, let us use FNNRHN with 100 hidden neurons for fitting curve (6). Let us generate randomly weights and biases over the interval $[-1, 1]$, which is typical for FNNRHN [9]. As we can see from Fig. 3 the AF fragments in the II are too flat and cannot be combined to get our TF. Another example in Fig. 4 is shown. Here weights are generated from $[-10, 10]$ and biases from $[-1, 1]$. As we can see from this figure the steep fragments of AFs are at the left border, where the TF is flat. On the other hand, at the right border, where the TF requires steep fragments, there are the flat AF fragments. This results in poor fitting. The above examples show that the problem is in definition of appropriate intervals for random weights and biases.

To determine the interval for a, let us set a sigmoid S in the II in such a way that its inflection point (which is for $h(x) = 0.5$) is in $x = 0$ and the sigmoid value in $x = 1$ is $r \in (0, 0.5)$ (see top, left chart in Fig. 5). Note that in such case the most nonlinear and steepest fragment of a sigmoid, which is around the inflection point, is inside the II. The parameter r should be lower than 0.5 (sigmoid value for the inflection point). For $r = 0.5$ we have completely flat function. If r decreases toward 0, the sigmoid S is more and more steep in the II. Thus r controls the flatness of S in the II.

When according to our requirements, the inflection point of S in $x = 0$, and the sigmoid value for $x = 1$ is r, then the shift parameter $b = 0$ and we get:

$$\frac{1}{1 + \exp(-(a \cdot 1 + 0))} = r \tag{8}$$

After transformations we obtain from (8) a slope parameter for S:

$$a = -\ln\left(\frac{1 - r}{r}\right) = a_{lim1} \tag{9}$$

(Note that for $r \in (0, 0.5)$ a_{lim1} is negative.)

Let us assume that the AFs building the FC are not flatter than the sigmoid S. Thus, their slope parameters satisfy the condition:

$$a \le -|a_{lim1}| \quad \text{or} \quad a \ge |a_{lim1}| \tag{10}$$

Parameter a_{lim1} defines the flattest AF possible in the set of m AFs. Let:

$$a_{lim2} = s \cdot a_{lim1} \tag{11}$$

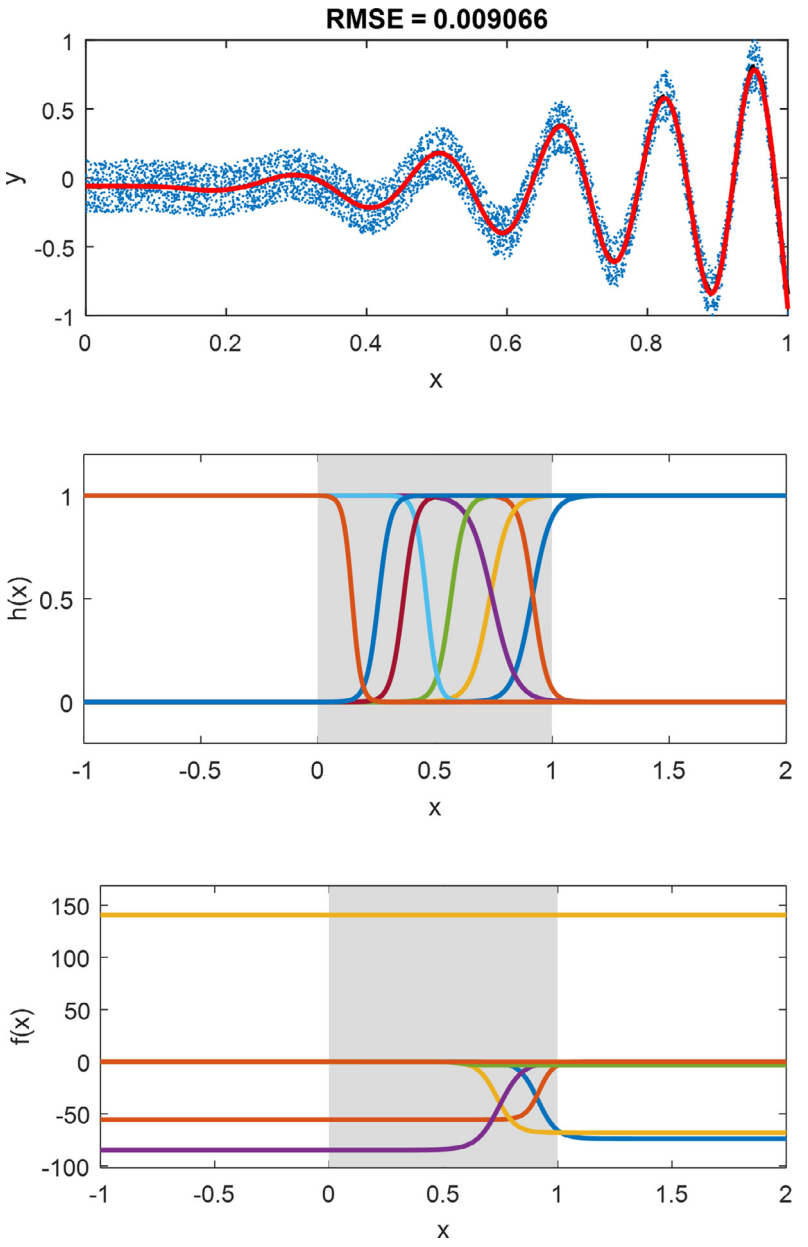

Fig. 2. Results of fitting for FNN with 9 hidden sigmoid nodes learned using Levenberg–Marquardt algorithm.

where $s > 1$ defines the steepest AF possible. So, the slope parameter of the i-th AF can be generated from the ranges:

$$a_i \in [-|a_{lim2}|, -|a_{lim1}|] \cup [|a_{lim1}|, |a_{lim2}|] \tag{12}$$

After substituting (9) and (11) in (12) and simplifying notation we obtain:

$$|a_i| \in \left[\ln\left(\frac{1-r}{r}\right), s \cdot \ln\left(\frac{1-r}{r}\right)\right] \tag{13}$$

Parameter s decides about the maximal steepness of AFs, and should correspond to the steepness of the TF.

When AF satisfies condition (13), it lies between two boundary AFs, with slope parameters a_{lim1} and a_{lim2}, respectively. These boundary AFs allow us to control the steepness of hidden neuron AFs to avoid their saturation fragments in the input interval. This is because saturation fragments are not suitable for nonlinear function fitting.

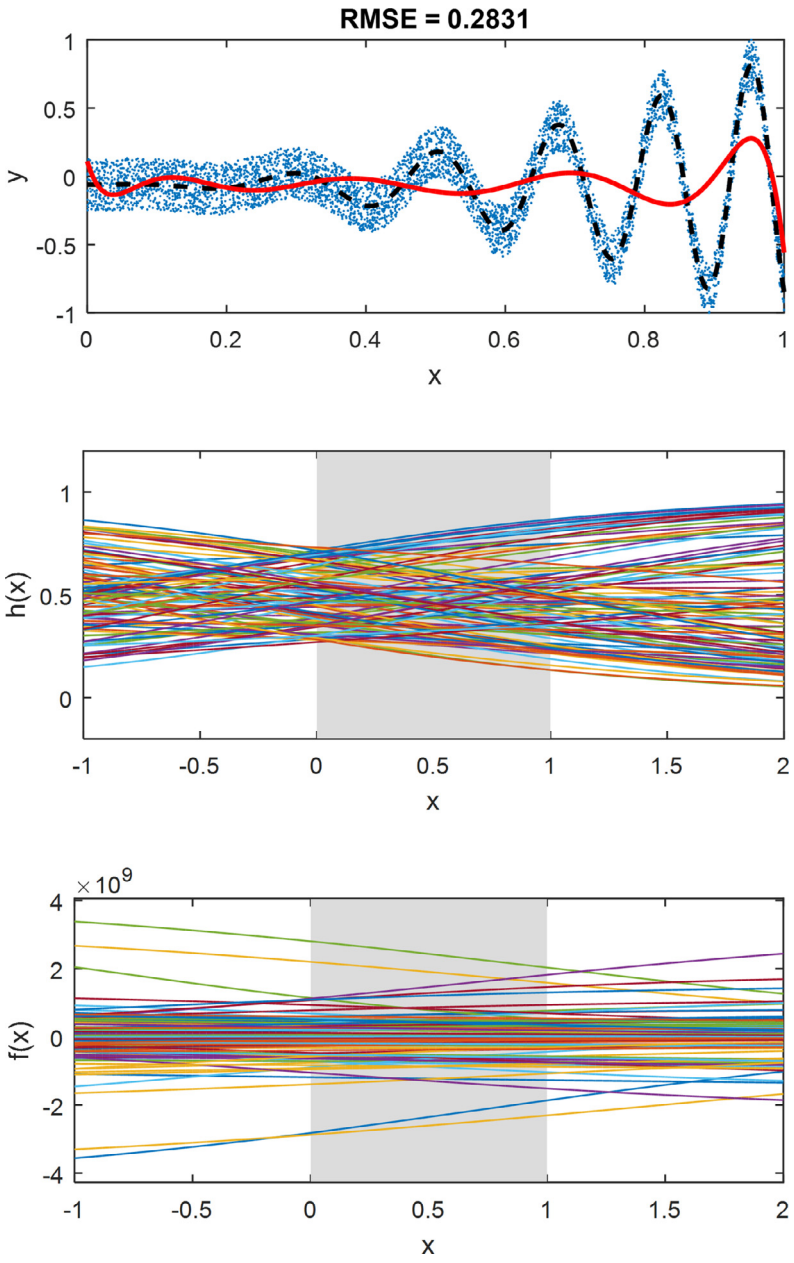

Fig. 3. Results of fitting for FNNRHN with 100 hidden sigmoid nodes, $a, b \in [-1, 1]$.

Now, let us set the shift parameter b in such a way that the sigmoid inflection point is inside the II. So, for some $x \in [0, 1]$ we get:

$$\frac{1}{1 + \exp(-(a \cdot x + b))} = 0.5 \tag{14}$$

After transformations we obtain:

$$b = -a \cdot x \tag{15}$$

For $x = 0$ we get a border of the interval for $b : b_{lim1} = 0$, and for $x = 1$, we get the second border of this interval: $b_{lim2} = -a$. Note, that the interval for the AF shift parameter b is dependent on the value of the slope parameter a of this AF. Thus, the biases should be generated individually for each ith AF from the interval:

$$b_i \in \begin{cases} [0, a_i] & \text{for} \quad a_i \leq 0 \\ [-a_i, 0] & \text{for} \quad a_i > 0 \end{cases} \tag{16}$$

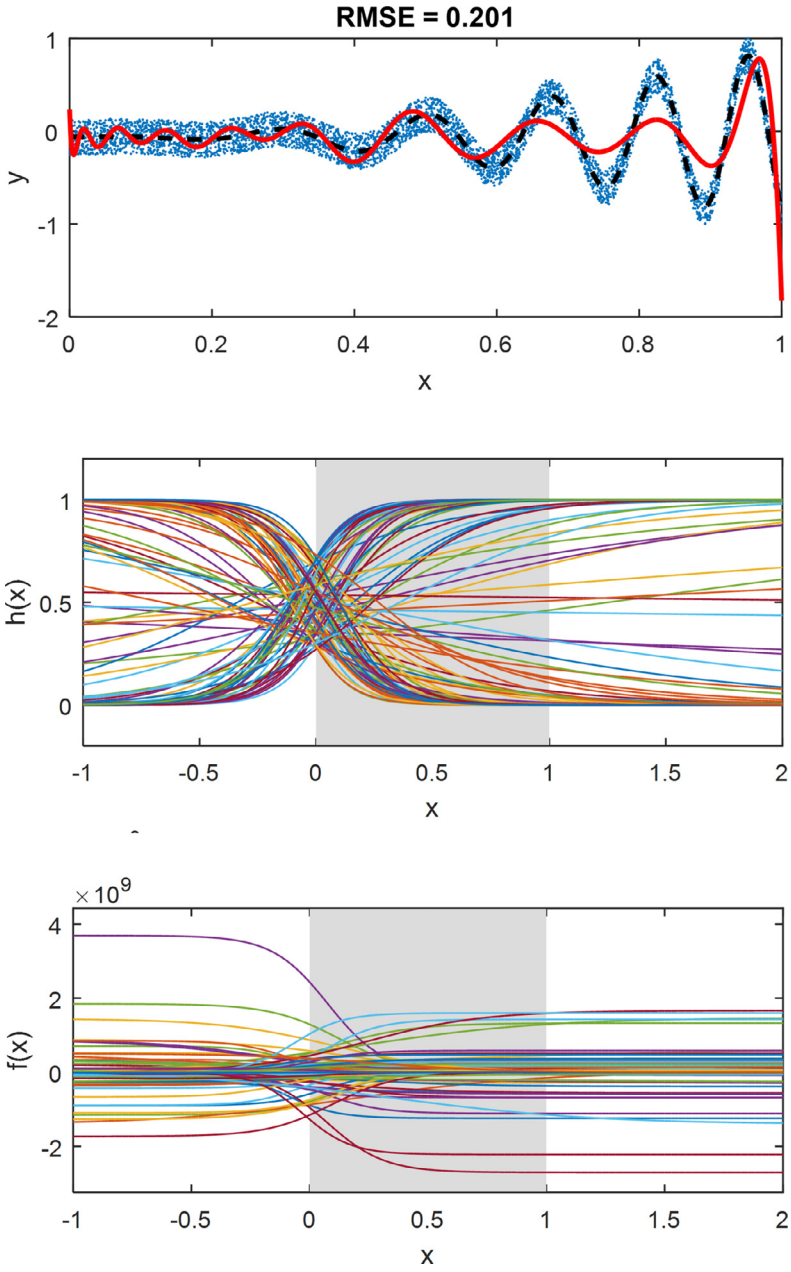

Fig. 4. Results of fitting for FNNRHN with 100 hidden sigmoid nodes, $a \in [-10, 10]$ and $b \in [-1, 1]$.

In Fig. 6 results of fitting are shown, where FNNRHN has 100 hidden nodes and the above described approach is used for generating random weights and biases. For $r = 0.1$ and $s = 3$ from (13) we get: $|a_i| \in [2.20, 6.56]$. Too small value of s leads to underfitting and too high value leads to overfitting. So it is recommended to select this parameter experimentally, e.g. in the cross-validation procedure, as well as parameter r.

3.2. Gaussian AFs

Now, let us consider Gaussian AF of the form:

$$h(x) = \exp\left(-(a \cdot x + b)^2\right) \tag{17}$$

Similarly to a sigmoid, weight a decides about a slope or width of the Gaussian function and the bias b shifts the function along the x-axis. To determine the interval for a, let us set a Gaussian function G in the II in such a way that its maximum

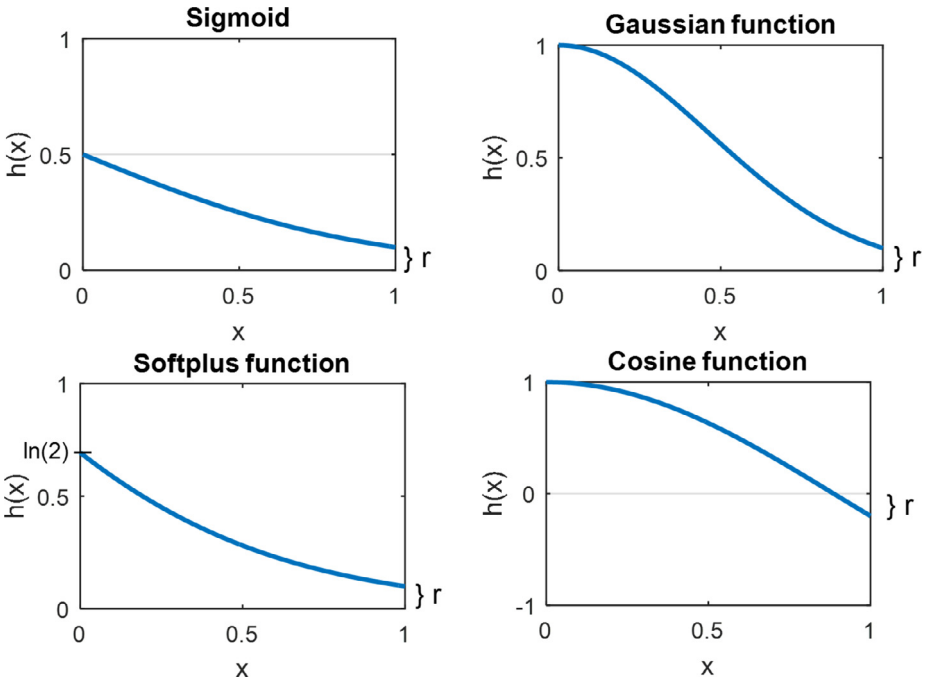

Fig. 5. The flattest fragments of AFs in the II.

$(h(x) = 1)$ is in $x = 0$ and its value in $x = 1$ is $r \in (0, 1)$ (see Fig. 5). In such case $b = 0$ and:

$$\exp\left(-(a \cdot 1 + 0)^2\right) = r \tag{18}$$

From (18) we get a slope parameter for G:

$$a = \sqrt{-\ln(r)} = a_{lim1} \tag{19}$$

Let us assume that the Gaussian AFs building the FC are not flatter than G. Thus, their slope parameters satisfy condition (10). Let (11) defines the steepest Gaussian AF in the set of m AFs. Thus, the slope parameter of the ith AF can be generated from ranges (12). After substituting (19) and (18) in (12) and simplifying notation we obtain:

$$|a_i| \in \left[\sqrt{-\ln(r)}, s \cdot \sqrt{-\ln(r)}\right] \tag{20}$$

Let us set the shift parameter b in such a way that the maximum of the Gaussian AF is inside the II. So, for some $x \in [0, 1]$ we get:

$$\exp\left(-(a \cdot x + b)^2\right) = 1 \tag{21}$$

From (21) we get the same equation for the shift parameter as for a sigmoid (15) and, consequently, the same interval from which the bias should be generated: (16).

Results of fitting when using Gaussian AFs, 100 hidden nodes and the above described approach for generating random weights and biases in Fig. 7 are shown. It was assumed: $r = 0.6$ and $s = 10$. For such value of parameters from (20) we get: $|a_i| \in [0.71, 7.15]$.

3.3. Softplus AFs

Similar considerations are performed for the softplus function:

$$h(x) = \ln\left(1 + \exp\left(a \cdot x + b\right)\right) \tag{22}$$

Parameters a and b play the same role as for sigmoid and Gaussian AFs. To determine the interval for a, let us assume $b = 0$ (no shift). In such case for any a a softplus function value in $x = 0$ is $h(0) = \ln(2)$. Now, let us assume that the value of the softplus function P in $x = 1$ is $r \in (0, \ln(2))$ (see Fig. 5). Thus:

$$\ln\left(1 + \exp\left(a \cdot 1 + 0\right)\right) = r \tag{23}$$

From (23) we get a slope parameter for P:

$$a = \ln(\exp(r) - 1) = a_{lim1} \tag{24}$$

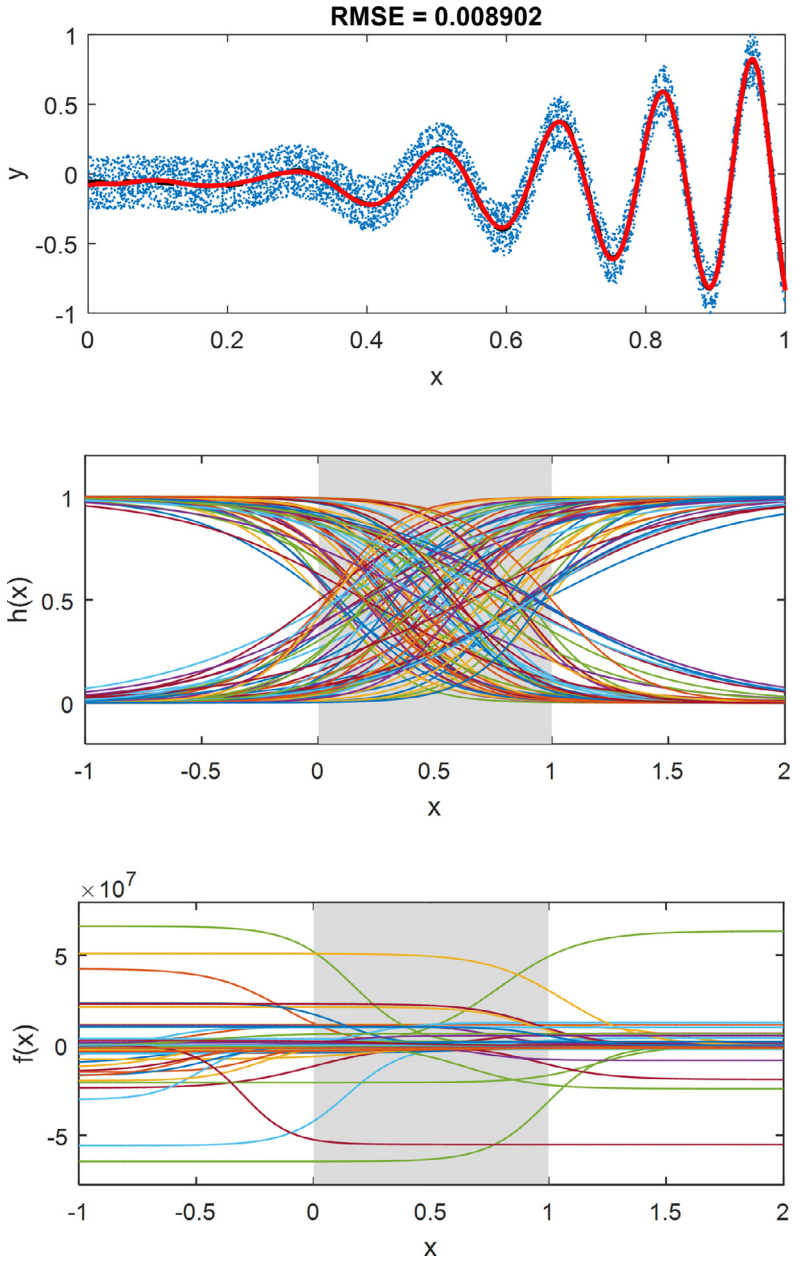

Fig. 6. Results of fitting for FNNRHN with 100 hidden sigmoid nodes, proposed algorithm with $r = 0.1$ and $s = 3$.

Let us assume that AFs are not flatter than P. It means that their slope parameters satisfy condition (10). As in the case of sigmoid and Gaussian AFs, we assume the slope parameter for the steepest AF in the set of hidden nodes as (11). This leads to the following interval for the slope parameter of the ith softplus AF:

$$|a_i| \in [-\ln(\exp(r) - 1), -s \cdot \ln(\exp(r) - 1)] \tag{25}$$

Now, let us set a softplus function so that their most curved fragment, which is around $h(x) = \ln(2)$, is in the II. So, for some $x \in [0, 1]$ we get:

$$\ln(1 + \exp(a \cdot x + b)) = \ln(2) \tag{26}$$

Again we obtain the same equation for b (15) and the same interval for ith AF bias (16).

In Fig. 8 results of fitting using 100 hidden nodes with softplus AFs are shown. It was assumed: $r = 0.3$ and $s = 10$. For such value of parameters from (25) we get: $|a_i| \in [1.05, 10.50]$.

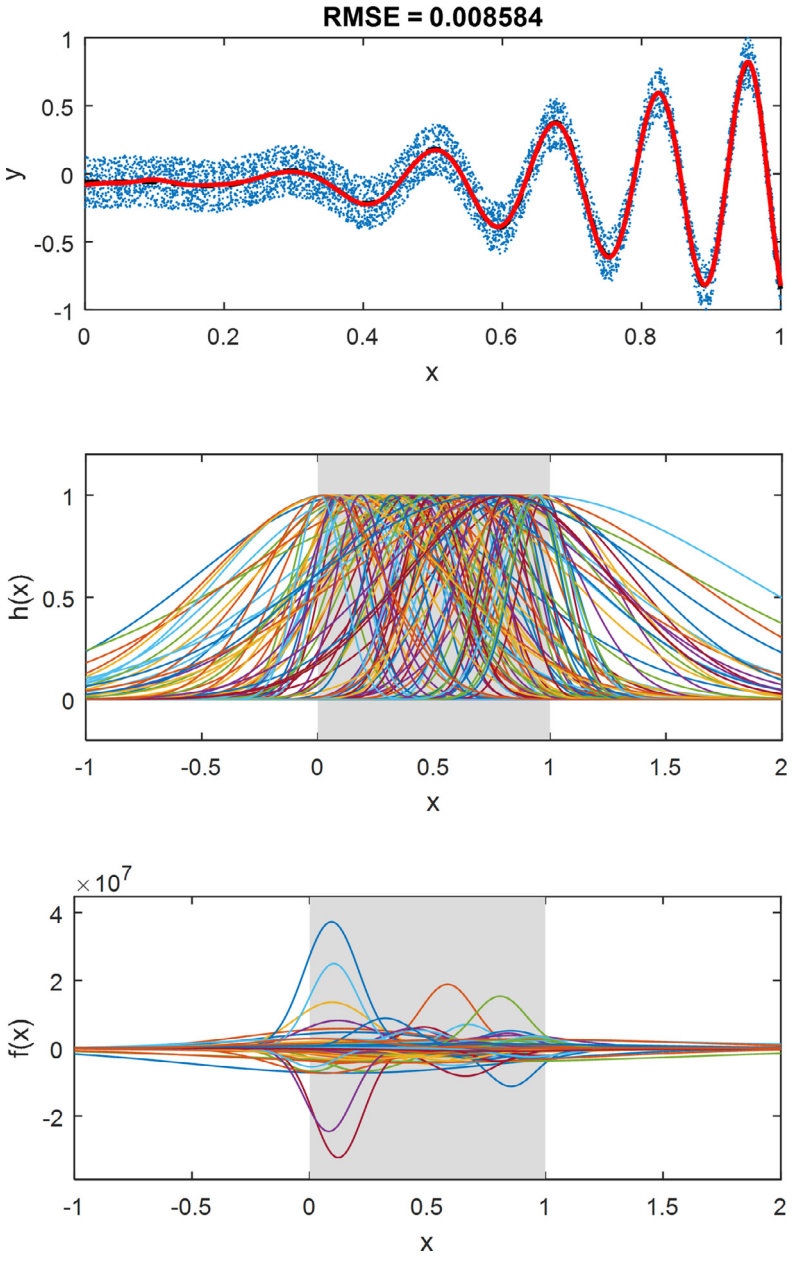

Fig. 7. Results of fitting for FNNRHN with 100 hidden Gaussian nodes, proposed algorithm with $r = 0.6$ and $s = 10$.

3.4. Sine and cosine AFs

Let us consider cosine as a AF:

$$h(x) = \cos(a \cdot x + b) \tag{27}$$

As before parameters a and b decide about slope and shift, respectively. For $b = 0$ and any a the cosine function value in $x = 0$ is $h(0) = 1$. Let us assume that in $x = 1$ the value of the cosine function C is $r \in [-1, 1)$ (see Fig. 5). Thus:

$$\cos(a \cdot 1 + 0) = r \tag{28}$$

and the slope parameter for C is:

$$a = \arccos(r) = a_{lim1} \tag{29}$$

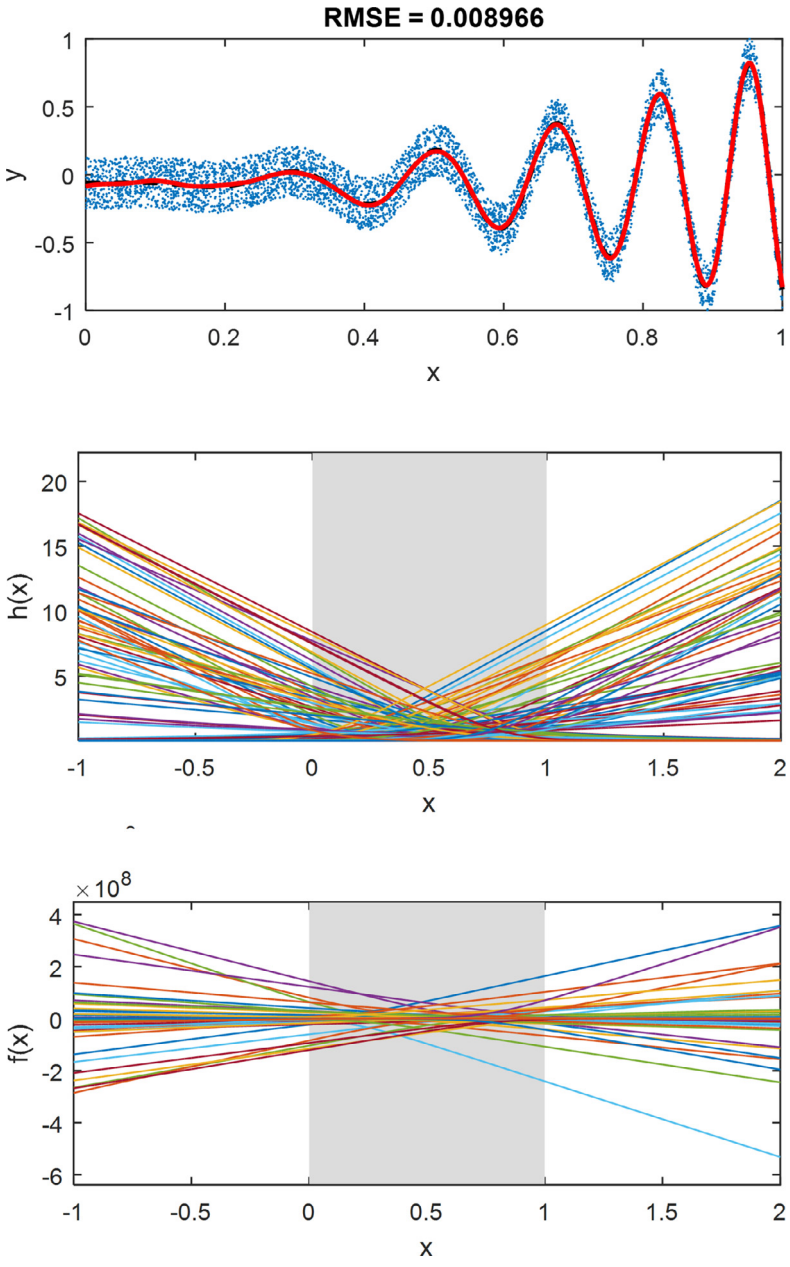

Fig. 8. Results of fitting for FNNRHN with 100 hidden softpuls nodes, proposed algorithm with $r = 0.3$ and $s = 10$.

As before let us assume that AFs are not flatter than C, and not steeper than the cosine function with a slope parameter defined as (11). This leads to the following interval for a_i:

$$|a_i| \in [\arccos(r), s \cdot \arccos(r)] \tag{30}$$

Let us set the cosine function in the II so that for some $x \in [0, 1]$:

$$\cos(a \cdot x + b) = 1 \tag{31}$$

From (31) we obtain equation for b (15) and the interval for ith AF bias (16).

For sine AF exactly the same equations for weight interval (30) and biases interval (16) can be used. This is because sine function is shifted version of the cosine function, Fig. 9.

In Fig. 10 results of fitting using 100 hidden nodes with cosine AFs are shown. It was assumed: $r = 0.2$ and $s = 20$. For such value of parameters from (30) we get: $|a_i| \in [1.37, 27.38]$.

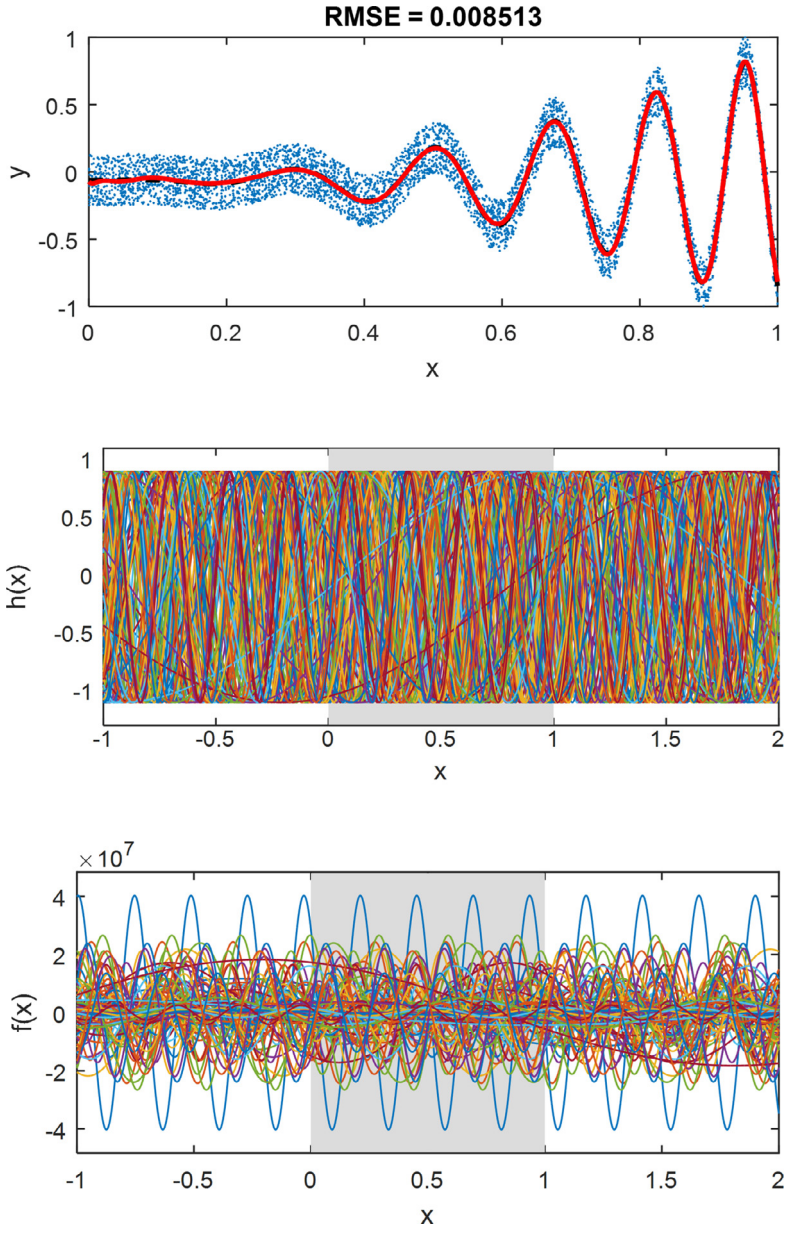

Fig. 9. Results of fitting for FNNRHN with 100 hidden cosine nodes, proposed algorithm with $r = 0.2$ and $s = 20$.

4. Generating random weights and biases - multidimensional case

Let input vectors $\mathbf{x}$ be normalized so that they belong to the n-dimensional unit hypercube $H = [0, 1]^n \subset \mathbb{R}^n$. In the multidimensional case, similarly to the one-dimensional case, weights a decide about slopes of the AF (in different directions in n-dimensional space) and the bias b shifts AF along the x-axes. Our goal is to find a method of generating random weights and biases, to ensure that inside the hypercube H there are nonlinear, steep fragments of AFs.

In this Section for brevity, we use the following acronyms:

- TF: target function $g(x)$,
- FS: fitted surface $\varphi(x)$.

To illustrate results two-variable TF is used of the form (see Fig. 10):

$$g(\mathbf{x}) = \sin\left(20 \cdot \exp\left(x_1\right)\right) \cdot x_1^2 + \sin\left(20 \cdot \exp\left(x_2\right)\right) \cdot x_2^2 \tag{32}$$

where $x_1, x_2 \in [0, 1]$.

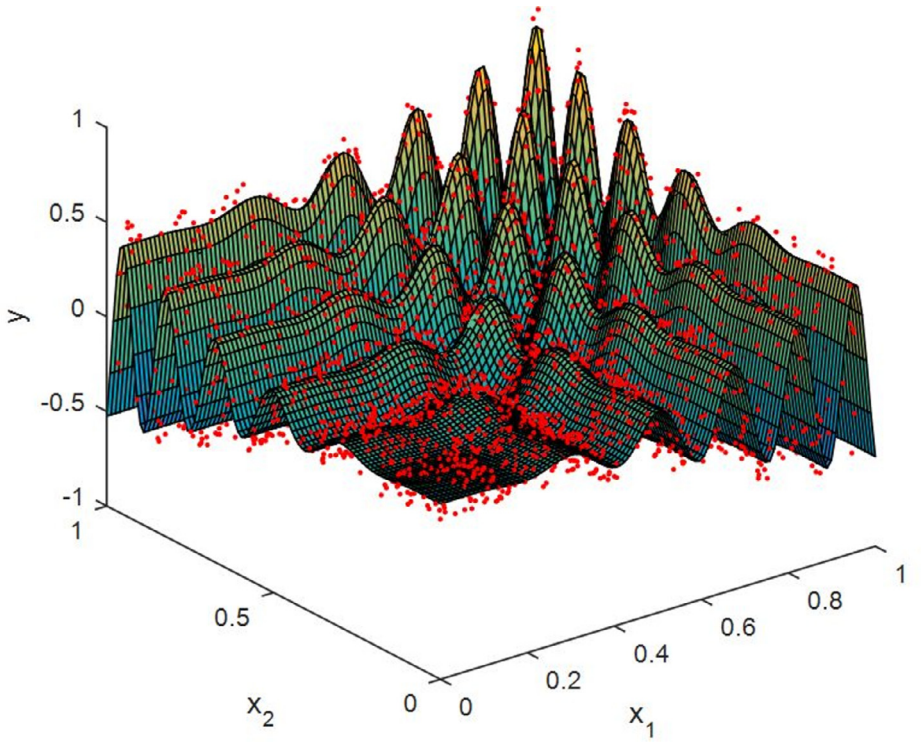

Fig. 10. Target function and training points.

A variation of function (32) is the lowest at the corner [0, 0] and increases towards the corner [1, 1]. The training set contains 5000 points $(\mathbf{x}_l, y_l)$, where components of $\mathbf{x}_l$: $x_{l,1}$ and $x_{l,2}$, are independently uniformly randomly distributed on [0, 1] and y_l are distorted by adding the uniform noise distributed in $[-0.2, 0.2]$. The testing set of the same size is created similarly but without noise. The outputs are normalized into the range $[-1, 1]$.

4.1. Sigmoid AFs

Let us set a sigmoid S:

$$h(\mathbf{x}) = \frac{1}{1 + \exp\left(-\left(\mathbf{a}^T\mathbf{x} + b\right)\right)} \tag{33}$$

inside the hypercube H in such a way that an inflection point (which is for $h(x) = 0.5$) is located in the corner $\mathbf{c}_0 = [0, 0, \ldots, 0]$ and the sigmoid value in the opposite corner $\mathbf{c}_1 = [1, 1, \ldots, 1]$ is $r \in (0,\ 0.5)$ (see top, left chart in Fig. 11 for two-dimensional example). In such a case the shift parameter $b = 0$ and the function value in the corner $\mathbf{c}_1$ is:

$$\frac{1}{1 + \exp\left(-\left(\sum\limits_{k=1}^{n} a_k \cdot 1 + 0\right)\right)} = r \tag{34}$$

From (34) after transformations we get:

$$\sum_{k=1}^{n} a_k = \ln\left(\frac{1-r}{r}\right) = \Sigma_{lim1} \tag{35}$$

Let us assume that the AFs building the FS are not flatter in the direction $\overrightarrow{\mathbf{c}_0\mathbf{c}_1}$ than the function S, and are not steeper in this direction than the sigmoid S' for which the sum of the slope parameters is:

$$\sum_{k=1}^{n} a_k = \Sigma_{lim2} = s \cdot \Sigma_{lim1} \tag{36}$$

where $s > 1$.

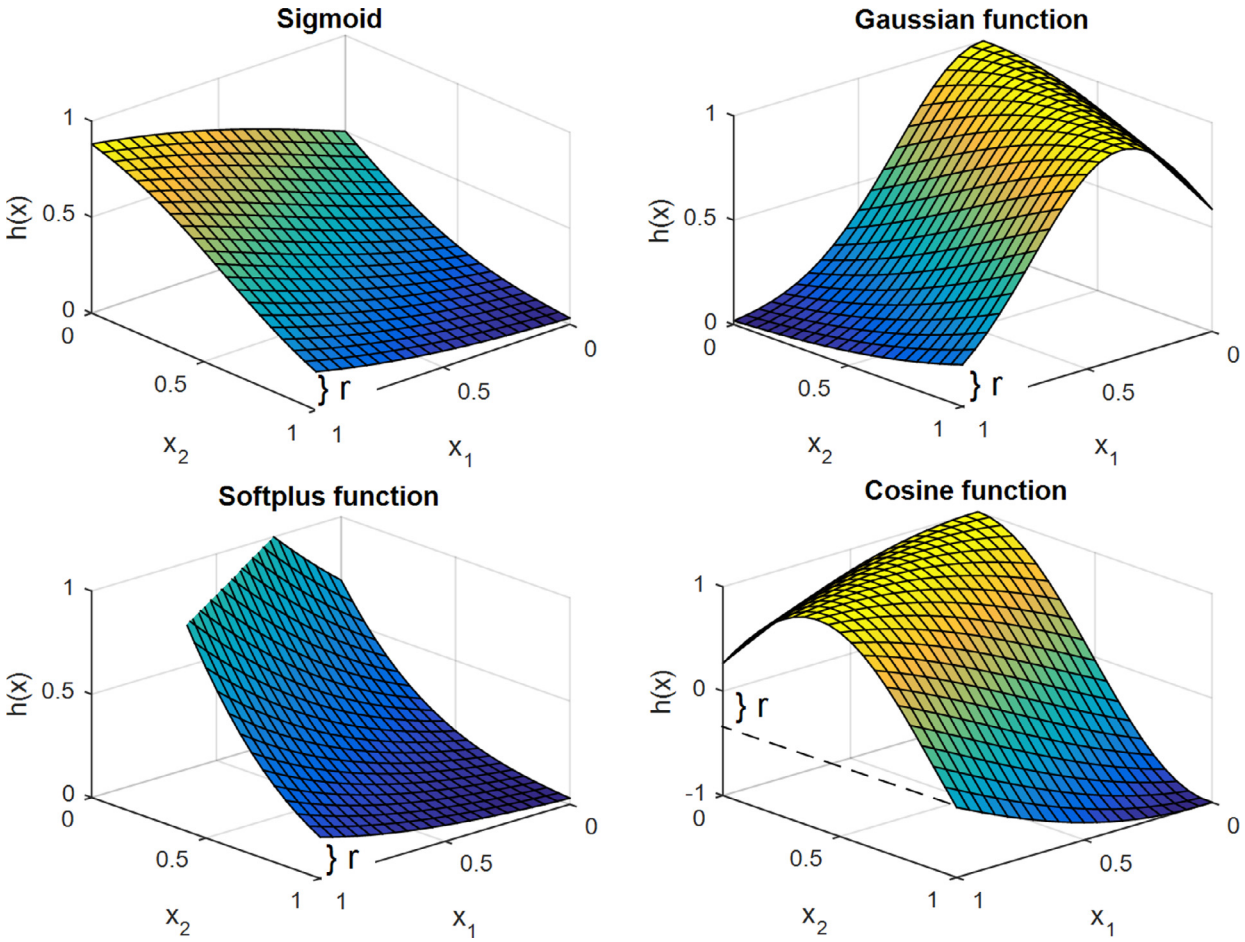

Fig. 11. Examples of AFs considered as the flattest in H in the direction $\overrightarrow{\mathbf{c}_0\mathbf{c}_1}$.

To keep the steepness in the direction $\overrightarrow{\mathbf{c}_0\mathbf{c}_1}$ of the ith AF between the assumed boundaries, the sum of its slope parameters should be from the interval:

$$\Sigma_i \in [-|\Sigma_{lim2}|, -|\Sigma_{lim1}|] \cup [|\Sigma_{lim1}|, |\Sigma_{lim2}|] \tag{37}$$

After substituting (35) and (36) in (37) the interval for Σ_i takes the form:

$$|\Sigma_i| \in \left[\ln\left(\frac{1-r}{r}\right), s \cdot \ln\left(\frac{1-r}{r}\right)\right] \tag{38}$$

The set of weights $a_1, a_2, \ldots, a_n$ for a given AF is generated as follows. First, the sum Σ_i is randomly selected from the interval (38). Then, the set of n i.i.d. numbers is generated randomly: $\zeta_1, \zeta_2, \ldots, \zeta_n \sim U(-1, 1)$. These numbers are recalculated such that their sum is Σ_i. After recalculation we get our weights for the ith AF:

$$a_{i,k} = \zeta_k \frac{\Sigma_i}{\sum_{j=1}^{n} \zeta_j} \tag{39}$$

Now, having weights $a_{i,k}$ the bias for ith AF is determined in such a way that the inflection point of AF located in $\mathbf{c}_0$ for $b = 0$, is shifted to some point $\mathbf{x}$ randomly generated inside the hypercube H. So, for some $\mathbf{x} : x_1, x_2, \ldots, x_n \sim U(0, 1)$ we get:

$$\frac{1}{1 + \exp\left(-\left(\mathbf{a}_i^T \mathbf{x} + b_i\right)\right)} = 0.5 \tag{40}$$

From (40) we obtain the general rule for generating randomly the bias for the ith AF in the case of $H = [0, 1]^n$:

$$b_i = -\sum_{k=1}^{n} a_{i,k} x_k \tag{41}$$

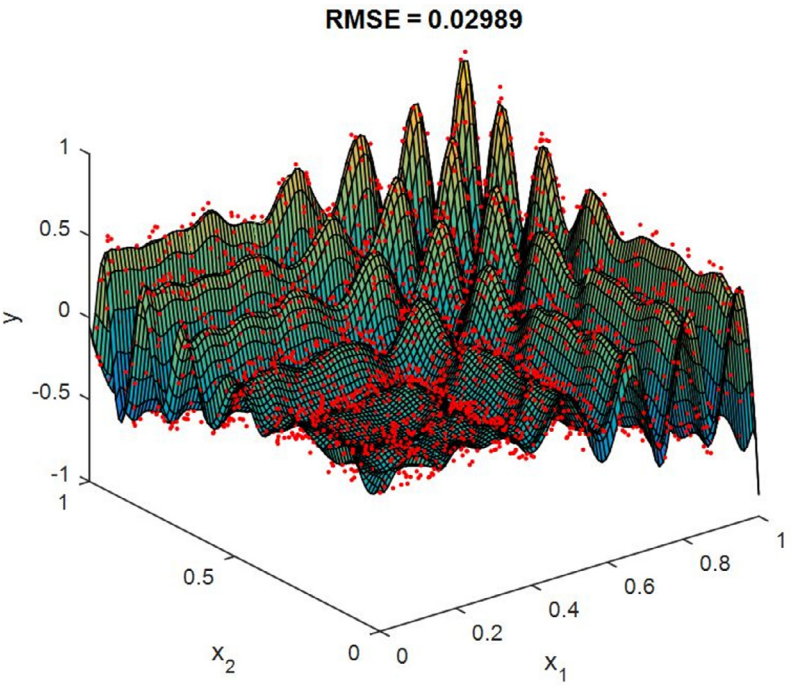

Fig. 12. Results of fitting for FNNRHN with 500 hidden sigmoid nodes, proposed algorithm with $r = 0.1$ and $s = 5$.

where $x_k \sim U(0, 1)$.

In Fig. 12 the FS is shown when using FNNRHN with 500 sigmoid nodes. For $r = 0.1$ and $s = 5$ from (38) we get: $|\Sigma_i| \in [2.20, 10.99]$.

4.2. Gaussian AFs

Let us set a Gaussian function G:

$$h(\mathbf{x}) = \exp\left(-\left(\mathbf{a}^T\mathbf{x} + b\right)^2\right) \tag{42}$$

in the hypercube H in such a way that a maximum point (which is for $h(x) = 1$) is located in the corner $\mathbf{c}_0 = [0, 0, \ldots, 0]$ and the function G value in the opposite corner $\mathbf{c}_1 = [1, 1, \ldots, 1]$ is $r \in (0, 1)$ (see Fig. 11 for two-dimensional example). In such a case the shift parameter b = 0 and the function value in the corner $\mathbf{c}_1$ is:

$$\exp\left(-\left(\sum_{k=1}^{n} a_k \cdot 1 + 0\right)^2\right) = r \tag{43}$$

From (43) we get a condition for the slope parameters of G:

$$\sum_{k=1}^{n} a_k = \sqrt{-\ln(r)} = \Sigma_{lim1} \tag{44}$$

As for a sigmoid, let us assume that the Gaussian AFs of the hidden nodes are not flatter in the direction $\overrightarrow{\mathbf{c}_0\mathbf{c}_1}$ than the function G, and are not steeper in this direction than the Gaussian function G' for which the sum of the slope parameters is (36). Thus, the sum of the slope parameters of the ith Gaussian AF should be from interval (37). After substituting (44) and (36) in (37), this can be written as:

$$|\Sigma_i| \in \left[\sqrt{-\ln(r)}, s \cdot \sqrt{-\ln(r)}\right] \tag{45}$$

The weights $a_{i,1}, a_{i,2}, \ldots, a_{i,n}$ are generated in the same way as for sigmoid AFs from (39).

The bias of the ith AF is determined in such a way that the maximum point located in $\mathbf{c}_0$ for $b = 0$ is shifted to some randomly generated point $\mathbf{x}$ inside the hypercube H. For this $\mathbf{x}$ we get:

$$\exp\left(-\left(\mathbf{a}_i^T\mathbf{x} + b_i\right)^2\right) = 1 \tag{46}$$

From (46) we obtain formulas for b_i: (41).

The FS when using FNNRHN with 500 Gaussian nodes, $r = 0.6$ and $s = 10$ is very similar to that one shown in Fig. 12. The RMSE is 0.02879. For these values of parameters from (45) we get: $|\Sigma_i| \in [0.71, 7.15]$.

4.3. Softplus AFs

When in the softplus function P:

$$h(\mathbf{x}) = \ln\left(1 + \exp\left(\mathbf{a}_i^T\mathbf{x} + b\right)\right) \tag{47}$$

we set $b = 0$, its value in $\mathbf{c}_0 = [0, 0, \ldots, 0]$ is $\ln(2)$. Let us assume that in $c_1 = [1, 1, \ldots, 1]$ the value of P is $r \in (0, \ln(2))$ (see Fig. 11). In such case:

$$\ln\left(1 + \exp\left(\sum_{k=1}^{n} a_k \cdot 1 + 0\right)\right) = r \tag{48}$$

From (47) we get a condition for the slope parameters of P:

$$\sum_{k=1}^{n} a_k = \ln\left(\exp(r) - 1\right) = \Sigma_{lim1} \tag{49}$$

As for the sigmoid and Gaussian AFs let us assume that the softplus AFs are not flatter in the direction $\overrightarrow{\mathbf{c}_0\mathbf{c}_1}$ than the function P, and are not steeper in this direction than the function P' for which the sum of the slope parameters is (36). It means that the sum of the slope parameters for the ith AF should be from interval (37). This can be written as:

$$|\Sigma_i| \in [-\ln\left(\exp(r) - 1\right), -s \cdot \ln\left(\exp(r) - 1\right)] \tag{50}$$

As for sigmoid and Gaussian AFs, the set of weights $a_{i,1}, a_{i,2}, \ldots, a_{i,n}$ for a given softplus AF is generated from (39). To determine the bias b_i of the softplus AF, we shift the softplus function with slopes parameters a_i and $b = 0$, in such a way that the point located in $\mathbf{c}_0$ is shifted to some randomly generated point $\mathbf{x}$ inside the hypercube H. So, for some $\mathbf{x} : x_1, x_2, \ldots, x_n \sim U(0, 1)$ we get:

$$\ln\left(1 + \exp\left(\mathbf{a}_i^T\mathbf{x} + b_i\right)\right) = \ln(2) \tag{51}$$

From (51) we obtain formulas for b_i: (41).

The FS when using FNNRHN with 500 softplus nodes, $r = 0.1$ and $s = 10$ is similar to the FS for sigmoid nodes which is shown in Fig. 12. The RMSE is 0.03099. From (45) we get: $|\Sigma_i| \in [2.25, 22.52]$.

4.4. Sine and Cosine AFs

The value of cosine function C:

$$h(\mathbf{x}) = \cos\left(\mathbf{a}_i^T\mathbf{x} + b\right) \tag{52}$$

in $\mathbf{c}_0 = [0, 0, \ldots, 0]$ for $b = 0$ is 1. Let us assume that in $\mathbf{c}_1 = [1, 1, \ldots, 1]$ the value of C is $r \in [-1, 1)$ (see Fig. 11). Thus:

$$\cos\left(\sum_{k=1}^{n} a_k \cdot 1 + 0\right) = r \tag{53}$$

A condition for the slope parameters of C derived from (53) is:

$$\sum_{k=1}^{n} a_k = \arccos(r) = \Sigma_{lim1} \tag{54}$$

Let us assume that the cosine AFs are not flatter in the direction $\overrightarrow{\mathbf{c}_0\mathbf{c}_1}$ than the function C, and are not steeper in this direction than the function C' for which the sum of the slope parameters is (36). Thus, the sum of the slope parameters for the ith cosine AF should be from interval (37), which can be written as:

$$|\Sigma_i| = [\arccos(r), s \cdot \arccos(r)] \tag{55}$$

Having the sum $|\Sigma_i|$, the set of weights $a_{i,1}, a_{i,2}, \ldots, a_{i,n}$ for the ith cosine AF is generated from (39).

To determine the bias b_i, we shift the cosine function with slopes parameters $\mathbf{a}_i$ and $b = 0$, in such a way that the point located in $\mathbf{c}_0$ is shifted to some randomly generated point $\mathbf{x} \sim U(0, 1)^n$ inside the hypercube H. Thus:

$$\cos\left(\mathbf{a}_i^T\mathbf{x} + b_i\right) = 1 \tag{56}$$

From (56) we obtain formulas for b_i: (41). The same intervals for weights (55) and biases (41) can be assumed for sine AFs.

The FS when using FNNRHN with 500 cosine nodes, $r = -0.9$ and $s = 20$ is similar to that one for sigmoid nodes (Fig. 12). The RMSE is 0.03439. From (55) we get: $|\Sigma_i| \in [2.21, 44.29]$.

Table 1
Generation of the FNNRHN hidden node parameters.

Activation function	Condition for the sum of input weights Σ_i	Interval for r	Weights of ith hidden node	Bias of ith hidden node
$\frac{1}{1+\exp\left(-\left(\mathbf{a}^T\mathbf{x}+b\right)\right)}$	$\lvert\Sigma_i\rvert \in \left[\ln\left(\frac{1-r}{r}\right), s\cdot\ln\left(\frac{1-r}{r}\right)\right]$	$r\in(0,\,0.5)$		
$\exp\left(-\left(\mathbf{a}^T\mathbf{x}+b\right)^2\right)$	$\lvert\Sigma_i\rvert \in \left[\sqrt{-\ln(r)}, s\cdot\sqrt{-\ln(r)}\right]$	$r\in(0,\,1)$	$a_{i,k}=\zeta_k\frac{\Sigma_i}{\sum_{j=1}^{n}\zeta_j}$	$b_i=-\sum_{k=1}^{n}a_{i,k}x_k$
$\ln\left(1+\exp\left(\mathbf{a}^T\mathbf{x}+b\right)\right)$	$\lvert\Sigma_i\rvert \in [-\ln(\exp(r)-1), -s\cdot\ln(\exp(r)-1)]$	$r\in(0,\,\ln(2))$		
$\cos\left(\mathbf{a}^T\mathbf{x}+b\right),\ \sin\left(\mathbf{a}^T\mathbf{x}+b\right)$	$\lvert\Sigma_i\rvert = [\arccos(r), s\cdot\arccos(r)]$	$r\in(-1,\,1]$		

where: $\mathbf{x}\in[0,\,1]^n$; $s>1$; $k=1,2,\ldots,n$; $\zeta_1,\zeta_2,\ldots,\zeta_n$ are i.i.d $U(-1,1)$ random variables; $x_k\sim U(0,\,1)$ or $x_k=x_{\zeta,k}$, where $\mathbf{x}_\zeta\in\Phi,\zeta\sim U\{1,2,\ldots,N\}$ or $x_k=p_{i,k}$, where $\mathbf{p}_i$ is a prototype of the ith cluster of $\mathbf{x}\in\Phi$.

4.5. Discussion

In the above analysis we quietly assumed that input points are evenly distributed in the unit hypercube H. In many (or even mostly) cases they are not. In such a case it is reasonably to shift the AFs in the bias determination step from $\mathbf{c}_0$ not to some randomly selected point $\mathbf{x}$ but to one of the training point. This ensures that all AFs have their nonlinear fragments in the regions containing data. The only modification of the above method is that for generating biases in (41) we use: $[x_1,x_2,\ldots,x_n]=\mathbf{x}_\zeta\in\Phi$, where ζ is a random integer uniformly distributed between 1 and N. Alternative way of choosing $\mathbf{x}_\zeta$ is to select them in regions of the input space where the TF is the most variable (has steep fragments). Another idea to calculate biases b_i is to group training points into m clusters. The prototypes p of these clusters (e.g. centroids) can be taken as the points to which the AFs are shifted from $\mathbf{c}_0$.

The above analysis for multidimensional case in Table 1 are summarized. The proposed process of generating random weights and biases for FNNRHN is shown in Algorithm 1. It requires the inputs to be normalized: $\mathbf{x}\in H=[0,1]^n$.

Algorithm 1 Generation of the hidden node weights and biases for FNNRHN.

Input
Activation function $h(\mathbf{x})$
Number of hidden nodes m
Number of inputs n
Training set Φ (optionally)
Set of prototypes $\{\mathbf{p}_i\}_{i=1,2,\ldots,m}$ (optionally)
Output
Weights $\mathbf{A}=\begin{bmatrix} a_{1,1} & \ldots & a_{m,1} \\ \vdots & \vdots & \vdots \\ a_{1,n} & \ldots & a_{m,n} \end{bmatrix}$
Biases $\mathbf{b}=[b_1,b_2,\ldots,b_m]$
Procedure
Set $r_{\min}=\min_{\mathbf{x}\in\mathbb{R}^n}h(\mathbf{x})$
Set $r_{\max}=h(\mathbf{x})$ for $\mathbf{x}=\mathbf{c}_0=[0,0,\ldots,0]$, $b=0$
Choose $r\in\mathbb{R}$ from $(r_{\min},r_{\max})$
Choose $s\in\mathbb{R}>1$
Transform $h(\mathbf{x})$ assuming $\mathbf{x}=\mathbf{c}_0=[1,1,\ldots,1]$ and $b=0$ to get formula for $\sum_{k=1}^{n}a_k$
Assume $\Sigma_{lim1}=\sum_{k=1}^{n}a_k$ and $\Sigma_{lim2}=s\cdot\Sigma_{lim1}$
for each node $i=1,2,\ldots,m$ **do**
  Choose randomly Σ_i from $[-|\Sigma_{lim2}|,-|\Sigma_{lim1}|]\cup[|\Sigma_{lim1}|,|\Sigma_{lim2}|]$
  Choose randomly i.i.d. $\zeta_1,\zeta_2,\ldots,\zeta_n\sim U(-1,1)$
  for $k=1,2,\ldots,n$ **do**
    Calculate $a_{i,k}=\zeta_k\frac{\Sigma_i}{\sum_{j=1}^{n}\zeta_j}$
  end for
  Choose randomly i.i.d $x_1,x_2,\ldots,x_n\sim U(0,1)$
  or set $[x_1,x_2,\ldots,x_n]=\mathbf{x}_\zeta\in\Phi$, where $\zeta\sim U\{1,2,\ldots,N\}$
  or set $[x_1,x_2,\ldots,x_n]=\mathbf{p}_i$, where $\mathbf{p}_i$ is a prototype of the i-th cluster of $\mathbf{x}\in\Phi$
  Calculate $b_i=-\sum_{k=1}^{n}a_{i,k}x_k$
end for
Return **A, b**

Besides the number of hidden neurons m, there are two parameters in the proposed method of generating random nodes: r defining the flattest AF in the direction $\overrightarrow{\mathbf{c}_0\mathbf{c}_1}$ and s defining the steepest AF in this direction. All AFs of hidden neurons are generated randomly between these boundary AFs. There is no theoretical guideline for choosing the values of r

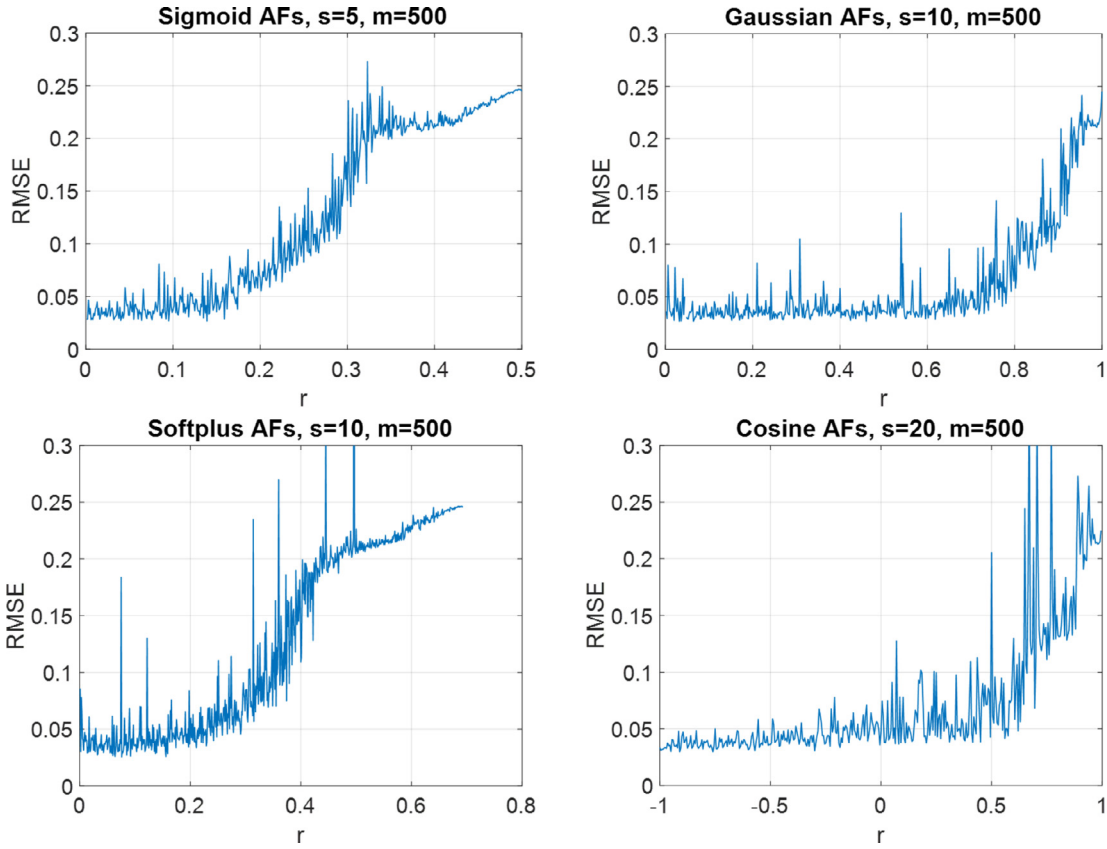

Fig. 13. Impact of r on the results.

and s as well as m. Because the parameters are strictly dependent on the target function, it is recommended to select them experimentally, e.g. in the cross-validation procedure.

Figs. 13–15 show the impact of the parameters on the results for the fitting task considered in this section (TF (32)). As we can see from Fig. 13, small r brings better results. But it should be remembered that these results are for given values of s and m. For other values of these parameters we can achieve different charts. The same can be said about the other charts presented in Figs. 14 and 15. As we might expect, an increase in the number of hidden nodes should lead to a reduction in fitting error. However, the charts in Fig. 15 contradict this expectation. Too large number of neurons leads to deterioration of results and increase in error variance. This is the most apparent for cosine AFs. In summary, it should be noted that the parameters are dependent on each other as well as on the target function. Their selection will be the subject of the future work.

5. Simulation study

In this section the proposed method of FNNRHN random parameters generation are illustrated on several examples. The first example concerns real-world application: short-term load forecasting. The task is to forecast the hourly load of the Polish power system for the next day (24 hourly loads). The input data are 24 hourly loads for the previous day. The data containing hourly loads of the Polish power systems in the period 2012–2015 are from www.entsoe.eu. The test period covers successive days of 2015. For each forecasting task, i.e. the forecast for ith day of the test period, separate model is learned on the training data from history including the same days of the week as the forecasted day. So, when the forecasted day is Monday, the training set contains 24-component load profiles for Sundays as inputs and 24-component load profiles for Mon-days as outputs. Atypical days such as public holidays are excluded from the training and test sets (about a dozen days in a year).

The FNNRHN with sigmoid AFs is used in MIMO variant: 24 inputs and 24 outputs. Because data are strongly correlated, they are not evenly distributed in the input hypercube H. So, in the bias determination step the AFs are shifted to the randomly selected training points. To select the model parameters (r and s; $m = 50$ is assumed to be constant), the network is learned on ten forecasting tasks from history that are the most similar to the current forecasting task. The history from which these ten tasks are selected is limited to the period covering year 2014 and the period of 2015 preceding the current

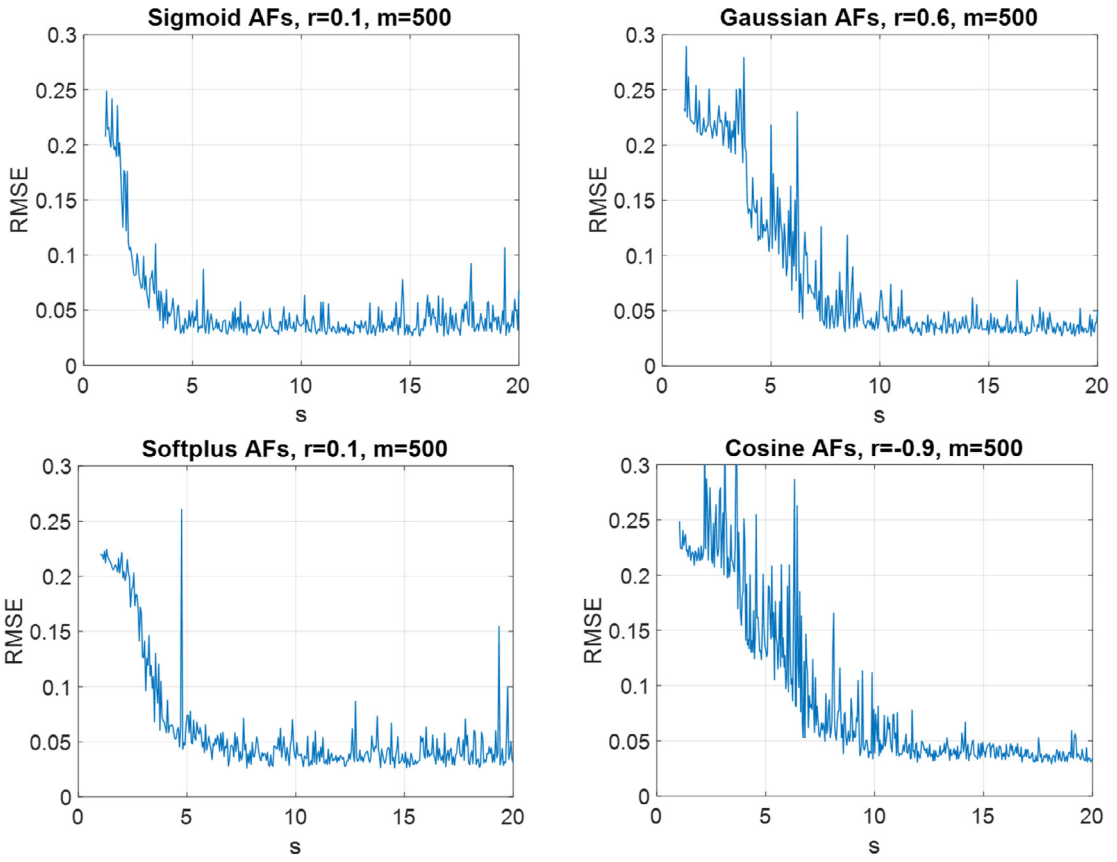

Fig. 14. Impact of s on the results.

forecasting task. By similar forecasting tasks we mean the tasks for the same day of the week as the current task and having input load profile similar to the current one (Euclidean distance is used for similarity measure).

The model parameters are optimized in grid search over the parameter grid: from 0.05 to 0.45 with step 0.05 for r, and from 1 to 10 with step 1 for s. The error measure applied in this study is mean absolute percentage error, MAPE, which is traditionally used as an error measure in short-term load forecasting. The forecasting error for the testing set was equal to 1.21.

For comparison FNNRHN with weights and biases randomly selected from the range $[-1, 1]$ was used. In this case the MAPE was equal to 4.41 for 50 hidden neurons. The relatively high error results from the mismatch of the random parameters range to the input data range, which is [9761, 23728]. In this range neurons are saturated. When increasing the number of hidden neurons the probability of appearing not saturated neurons in the input interval increases as well. This is evident in the results: for 100 neurons the MAPE reduces to 3.70, for 200 neurons to 3.43, and for 500 neurons to 2.97. But we should remember that most of the neurons are saturated and therefore unusable in the modeling of the nonlinear target function.

Fig. 16 shows a sample result of forecasting for a week period in January 2015. For this period the proposed FNNRHN gave MAPE = 1.22, while FNNRHN with $[-1, 1]$ scope for random parameters gave MAPE = 2.90 for 500 nodes and MAPE = 9.18 for 50 nodes.

Next examples, taken from [14], concern regression problems and include a function approximation and three real-world modeling tasks:

- Approximation of the single-variable TF:

$$g(\mathbf{x}) = 0.2e^{-(10x-4)^2} + 0.5e^{-(80x-40)^2} + 0.3e^{-(80x-20)^2} \tag{57}$$

  The training set contains 1000 points (x_l, y_l), where x_l are uniformly randomly distributed on [0, 1]. The test set of size 300 is generated from a regularly spaced grid on [0, 1].
- Stock – daily stock prices from January 1988 through October 1991, for ten aerospace companies. The task is to approximate the price of the 10th company given the prices of the rest. There are 950 samples composed of nine input variables and one output variable. The whole data set was divided into training set containing 75% samples selected randomly, and the test set containing the remaining samples.

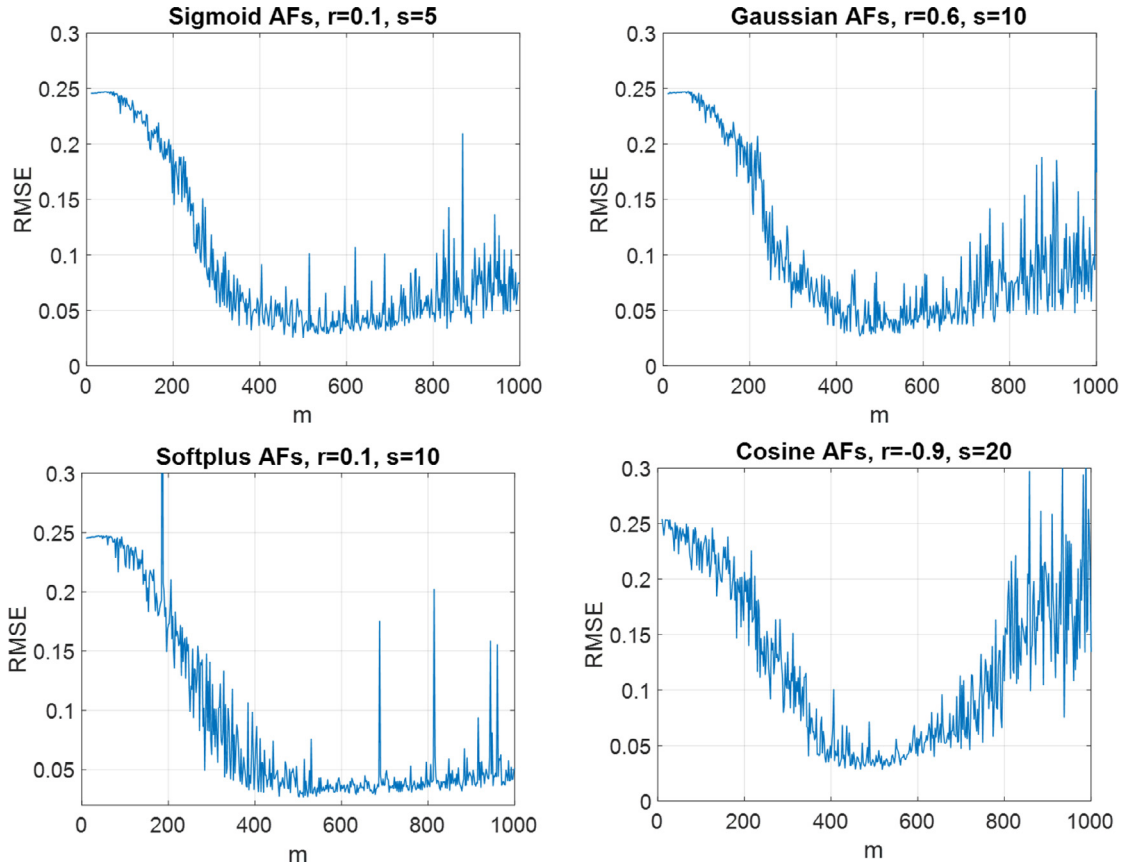

Fig. 15. Impact of m on the results.

- Concrete – the dataset contains the concrete compressive strength, age, and ingredients: cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate. The task is to approximate the highly nonlinear relationship between concrete compressive strength and the ingredients and age. There are 1020 samples composed of eight input variables and one output variable. The whole data set was divided into training and test parts in the same manner as Stock data set.
- Compactiv – the Computer Activity dataset is a collection of computer systems activity measures. The data was collected from a Sun Sparcstation 20/712 with 128 Mbytes of memory running in a multi-user university department. The task is to predict the portion of time that CPUs run in user mode. There are 8192 samples composed of 21 input variables (activity measures) and one output variable. The whole data set was divided into training and test parts in the same manner as Stock data set.

The datasets Stock, Concrete and Compactiv were downloaded from KEEL (Knowledge Extraction based on Evolutionary Learning) dataset repository (http://www.keel.es/). The input and output variables are normalized into [0, 1]. All results reported in this work take averages over 100 independent trials. Root Mean Squares Error (RMSE) was used as a measure of modeling accuracy.

Results are compared with the state-of-the-art method proposed recently in [14] as well as with Modified Quickprop and Incremental Random Vector Functional Link (IRVFL) network. The comparative models adopted from [14] are:

- MQ – Modified Quickprop algorithm proposed in [6] that iteratively finds the appropriate parameters for the new hidden node added in the incremental procedure. The parameters of MQ were set by authors [14] as follows: learning rate = 0.05, maximum iterative number = 200.
- IRVFL – Incremental Random Vector Functional Link network where the model is built incrementally with random assignment of the input weights and biases, and constructive evaluation of its output weights using the least squares method [7]. The random parameters were taken by authors of [14] from the uniform distribution over $[-1, 1]$.
- SCN – Stochastic Configuration Network proposed in [14]. This is a randomized model constructed incrementally using stochastic configuration algorithm where random parameters are generated with an inequality constraint from the adaptively selected scope $[-\lambda, \lambda]$, ensuring the universal approximation property of the built randomized learner model. Among three algorithmic implementations of SCN, the most accurate one was chosen, signed SC-III in [14], where the

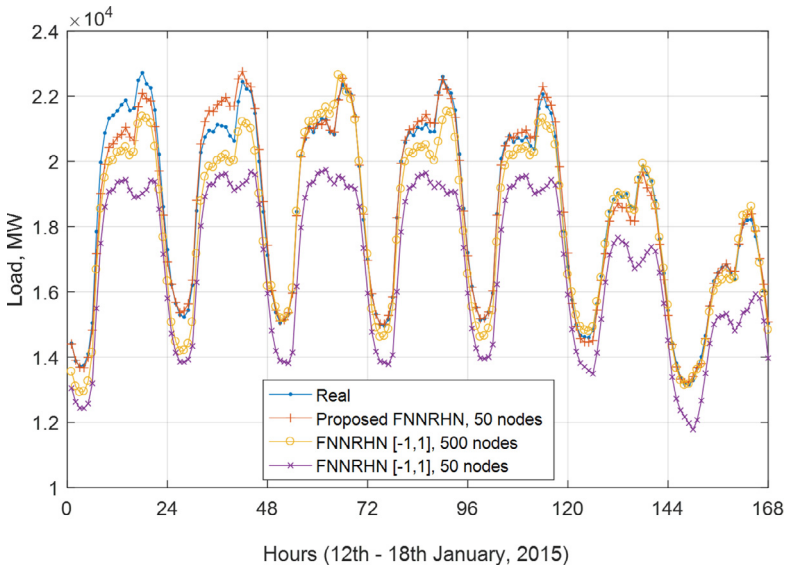

Fig. 16. Sample weekly fragment of the load time series and its forecasts.

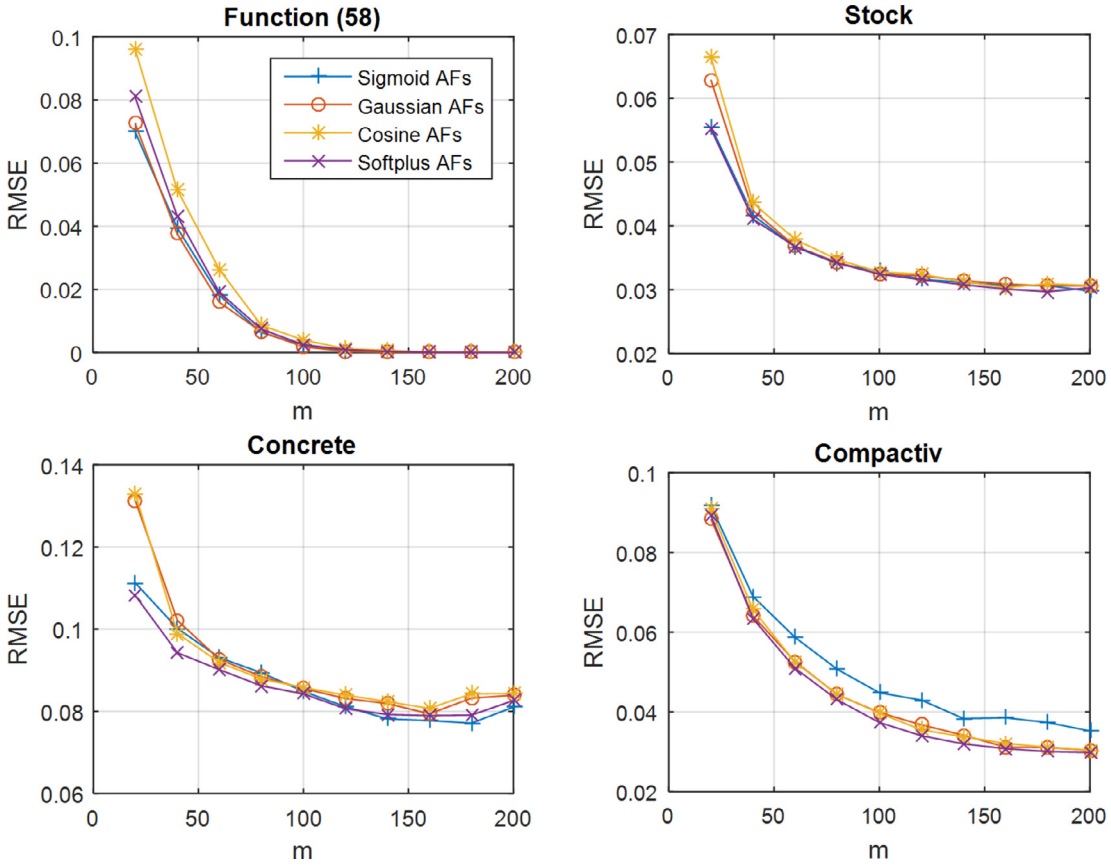

Fig. 17. Fitting error for different number of hidden neurons.

output weights are recalculated all together through solving a global least squares problem each time while a new hidden node is added. Sigmoidal activation function were used for the hidden nodes. The SCN parameters were selected by authors of [14] to ensure the best performance.

Table 2 shows the results: errors and their standard deviations for FNNRHN with different AFs which parameters are generated using the proposed method (FNNRHN-sig, FNNRHN-Gauss, FNNRHN-cos and FNNRHN-soft) as well as for the

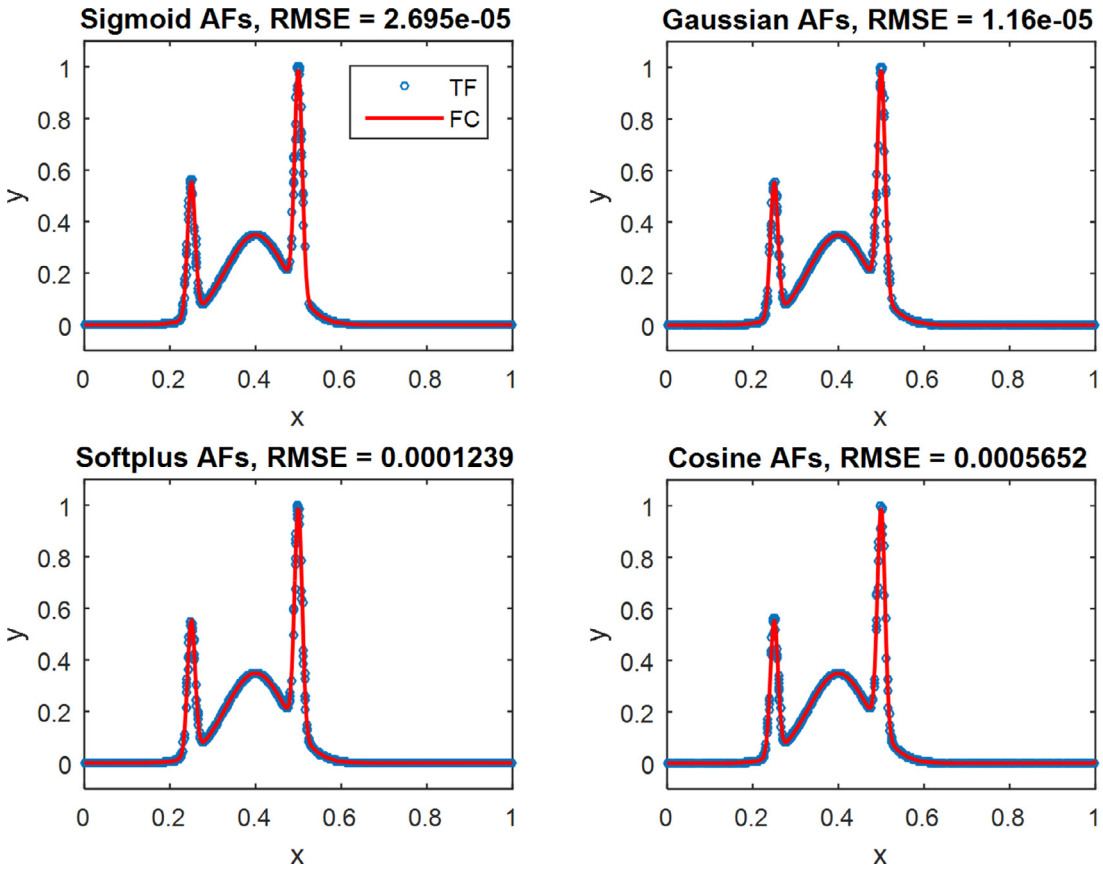

Fig. 18. Results of TF (57) fitting for FNNRHN with different AFs.

comparative models (copied from Table I of [14] for 50 neurons). The optimal parameter values of the proposed method, r and s, are also shown in Table 2. They are selected in the grid search using 10-fold cross-validation. In these procedure only the training parts of the datasets were used. The number of hidden neurons for FNNRHN was set to 50 in all cases to compare results with the comparative models having the same number of neurons.

As we can see from Table 2 the proposed method allows FNNRHN to achieve results better than IRVFL where the scope for random parameters is $[-1, 1]$. MQ gives similar results as FNNRHN for Stock and Compactiv datasets, little better for Concrete dataset, and much worse for function (57). MQ is fully adaptive for new nodes added in the hidden layer using gradient-ascent algorithm. Thus, the hidden neurons are optimally deployed in the input space. This is the reason for quite good results. But it should be remembered that the gradient-ascent algorithm uses also second-order information in optimization of the objective function. It is problematic when one is exploring in the region of a plateau in the error surface, where the first and second derivatives of the function to be optimized with respect to all the parameters are nearly zero. Although the MQ algorithm is equipped with an escape mechanism from these regions, it does not always work. Thus, the optimal solution cannot be guaranteed when the optimization is non-convex, i.e. it is nonlinear in the hidden layer parameters.

In the case of the randomized algorithms for training NNs the optimization problem is linear in the parameters, thus the optimization is convex and has an analytic solution, such as the least squares. But in these algorithms the key issue is to properly generate the random parameters of the hidden neurons to find the orthogonal projection of y into the input space [1]. In IRVFL the random parameters are taken from the fixed range $[-1, 1]$, and there is no guarantee that it is appropriate for the regression problem. So, the results for IRVFL are even worse than for MQ. SCN searches for the random parameter ranges for each new node added to the hidden layer. Thus, this ranges are optimized for each neuron. This translates into much better results than for fixed ranges, which are set without any scientific justification.

SCN-III algorithm outperformed all others including the proposed one. But it should be noticed that in SC-III the random configuration is generated not once like in FNNRHN but many times. In the experiments reported in Table 2 the maximum times of random configuration $T_{\max}$ was set as 200. This step of the SCN algorithm can be viewed as an optimization procedure selecting the most appropriate configuration which maximizes the model performance. SCN-III equipped with this optimization mechanism brings better results than FNNRHN. It seems that for FNNRHN, where the sets of random

Table 2
Results comparison among proposed and comparative models[a].

Algorithm	Parameters r/s	Training RMSE	Test RMSE
Function (58)			
MQ	–	0.1030 ± 0.0001	0.1011 ± 0.0003
IRVFL	–	0.1626 ± 0.0005	0.1617 ± 0.0008
SCN-III	–	0.0097 ± 0.0036	0.0100 ± 0.0033
FNNRHN-sig	0.050/50	0.0356 ± 0.0127	0.0377 ± 0.0137
FNNRHN-Gauss	0.900/160	0.0364 ± 0.0106	0.0361 ± 0.0097
FNNRHN-soft	0.150/90	0.0346 ± 0.0117	0.0367 ± 0.0117
FNNRHN-cos	0.050/90	0.0419 ± 0.0186	0.0428 ± 0.0207
Stock			
MQ	–	0.0410 ± 0.0014	0.0407 ± 0.0017
IRVFL	–	0.1853 ± 0.0248	0.1787 ± 0.0237
SCN-III	–	0.0327 ± 0.0007	0.0347 ± 0.0012
FNNRHN-sig	0.300/6.5	0.0380 ± 0.0018	0.0406 ± 0.0030
FNNRHN-Gauss	0.850/3.5	0.0385 ± 0.0019	0.0402 ± 0.0027
FNNRHN-soft	0.175/2.5	0.0376 ± 0.0015	0.0394 ± 0.0024
FNNRHN-cos	0.950/4.5	0.0381 ± 0.0015	0.0403 ± 0.0023
Concrete			
MQ	–	0.0910 ± 0.0014	0.0869 ± 0.0021
IRVFL	–	0.1929 ± 0.0135	0.1983 ± 0.0166
SCN-III	–	0.0835 ± 0.0012	0.0850 ± 0.0025
FNNRHN-sig	0.300/4.5	0.0937 ± 0.0038	0.1098 ± 0.0044
FNNRHN-Gauss	0.800/1.5	0.0921 ± 0.0036	0.1065 ± 0.0047
FNNRHN-soft	0.350/2	0.0877 ± 0.0018	0.1022 ± 0.0024
FNNRHN-cos	0.950/1.5	0.0880 ± 0.0016	0.1026 ± 0.0024
Compactiv			
MQ	–	0.0600 ± 0.0071	0.0624 ± 0.0075
IRVFL	–	0.1924 ± 0.0283	0.1882 ± 0.0281
SCN-III	–	0.0394 ± 0.0016	0.0418 ± 0.0021
FNNRHN-sig	0.100/3.5	0.0669 ± 0.0064	0.0694 ± 0.0064
FNNRHN-Gauss	0.850/5	0.0609 ± 0.0057	0.0655 ± 0.0070
FNNRHN-soft	0.350/8.5	0.0600 ± 0.0051	0.0636 ± 0.0054
FNNRHN-cos	0.650/2	0.0605 ± 0.0050	0.0666 ± 0.0065

[a] Results for MQ, IRVFL and SCN are taken from [14].

Table 3
Results for fnnrhn with optimized m, r and s.

Algorithm	Parameters $m/r/s$	Training RMSE	Test RMSE
Function (58)			
FNNRHN-sig	200/0.225/140	$2.5 \cdot 10^{-5} \pm 3.7 \cdot 10^{-5}$	$3.1 \cdot 10^{-5} \pm 4.6 \cdot 10^{-5}$
FNNRHN-Gauss	200/0.75/190	$1.1 \cdot 10^{-5} \pm 2.4 \cdot 10^{-5}$	$1.4 \cdot 10^{-5} \pm 3.0 \cdot 10^{-5}$
FNNRHN-soft	200/0.175/100	$8.5 \cdot 10^{-5} \pm 1.3 \cdot 10^{-4}$	$1.0 \cdot 10^{-4} \pm 1.7 \cdot 10^{-4}$
FNNRHN-cos	200/-0.95/190	0.0017 ± 0.0142	0.0031 ± 0.0191
Stock			
FNNRHN-sig	200/0.3/7	0.0208 ± 0.0005	0.0286 ± 0.0014
FNNRHN-Gauss	180/0.6/2.5	0.0220 ± 0.0006	0.0283 ± 0.0016
FNNRHN-soft	180/0.05/4.5	0.0220 ± 0.0006	0.0281 ± 0.0015
FNNRHN-cos	160/0.15/2.5	0.0236 ± 0.0006	0.0287 ± 0.0014
Concrete			
FNNRHN-sig	180/0.425/2.5	0.0527 ± 0.0011	0.0891 ± 0.0066
FNNRHN-Gauss	160/0.85/1.5	0.0600 ± 0.0020	0.0925 ± 0.0060
FNNRHN-soft	160/0.275/5	0.0598 ± 0.0018	0.0918 ± 0.0051
FNNRHN-cos	160/0.55/4	0.0617 ± 0.0022	0.0938 ± 0.0062
Compactiv			
FNNRHN-sig	200/0.2/8.5	0.0320 ± 0.0012	0.0395 ± 0.0025
FNNRHN-Gauss	200/0.85/1.5	0.0274 ± 0.0004	0.0329 ± 0.0018
FNNRHN-soft	200/0.125/1.5	0.0272 ± 0.0004	0.0325 ± 0.0018
FNNRHN-cos	200/0.95/3	0.0275 ± 0.0003	0.0335 ± 0.0019

parameters are not searched, 50 hidden nodes used in experiments are not enough to get the performance comparable with SCN-III. In the next experiments, to improve FNNRHN results the number of hidden nodes m, as well as r and s were selected in the grid search using 10-fold cross-validation. The RMSEs estimated in cross-validation for different number of neurons in Fig. 17 are shown. More neurons provide to better results. Table 3 shows the results when all three parameters of FNNRHN are tuned. As we can see from this table the training and testing errors are much lower than in the case presented in Table 2, where the number of hidden nodes set as 50 is insufficient.

In the light of the considerations carried out in this work, assigning the same fixed ranges for weights and biases is questionable. There is no problem when the TF is not strongly nonlinear or "flat", without spikes and sudden jumps. Such function can be approximated using flat fragments of the AFs, so the ranges for random parameters are not as important. But the problem arises when we approximate a strongly nonlinear function such as TF (57). The proposed FNNRHN gives for this case very good results which are shown in Fig. 18 (compare with Fig. 3 in [14], where FC for IRVFL and SCN are shown).

6. Conclusion

Randomized algorithms for training NNs suffer from design choices, translated in free parameters, which are difficult to set optimally and require many trials and cross-validation to find a good projection space [1]. In this work we demonstrate that the intervals of the random weights and biases in FNNRHN are extremely important due to approximation properties of the network. Activation functions of the hidden neurons are the basis functions which linear combination forms the surface fitting data. For non-linear target function the set of AFs should deliver nonlinear fragments to model the target function in its nonlinear regions with required accuracy.

The main contribution of this work is to propose a practical method of randomly generating weights and biases in FNNRHN to set nonlinear fragments of AFs in the input space region containing data points. The analyzes carried out lead to the conclusion that parameters of hidden nodes are dependent on the input data range and activation function type. Ranges for weights and biases should be considered separately, because these parameters have different meaning. Moreover, the range for the bias of the ith hidden node is strictly dependent on the weights of this node. The proposed method allows us to control the flatness and steepness of the AF set and hence the degree of generalization of the network and bias-variance tradeoff of the model.

Acknowledgements

this work was supported by Grant 2017/27/B/ST6/01804 from the National Science Centre, Poland.

References

[1] S. Ferrari, R.F. Stengel, Smooth function approximation using neural networks, IEEE Trans. Neural Netw. 16 (1) (2005) 24–38.
[2] A.N. Gorban, I.Y. Tyukin, D.V. Prokhorov, K.I. Sofeikov, Approximation with random bases: pro- et contra, Inf. Sci. (2016) 129–145. 364–365.
[3] M.T. Hagan, M. Menhaj, Training feed-forward networks with the marquardt algorithm, IEEE Trans. Neural Netw. 5 (5) (1994) 989–993.
[4] D. Husmeier, Random Vector Functional Link (RVFL) Networks, Neural Networks for Conditional Probability Estimation: Forecasting Beyond Point Predictions, chapter 6, Springer, 1999.
[5] B. Igelnik, Y.H. Pao, Stochastic choice of basis functions in adaptive function approximation and the functional-link net, IEEE Trans. Neural Netw. 6 (6) (1995) 1320–1329.
[6] T.Y. Kwok, D.Y. Yeung, Objective functions for training new hidden units in constructive neural networks, IEEE Trans. Neural Netw. 8 (8) (1997) 1131–1148.
[7] M. Li, D. Wang, Insights into randomized algorithms for neural networks: practical issues and common pitfalls, Inf. Sci. (2017) 170–178. 382–383.
[8] D. Nguyen, B. Widrow, Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights, in: Proc. Int. Joint Conf. Neural Networks, 3, 1990, pp. 21–26.
[9] Y.H. Pao, G.H. Park, D.J. Sobajic, Learning and generalization characteristics of the random vector functional-link net, Neurocomputing 6 (6) (1994) 163–180.
[10] Y.H. Pao, Y. Takefji, Functional-link net computing: theory, system architecture, and functionalities, IEEE Comput. 25 (25) (1992) 76–79.
[11] J. Principe, B. Chen, Universal approximation with convex optimization: gimmick or reality? IEEE Comput. Intell. Mag. 10 (2015) 68–77.
[12] S. Scardapane, D. Wang, Randomness in neural networks: an overview, WIREs Data Min. Knowl. Discov. 7 (7) (2017), doi:10.1002/widm.1200.
[13] W.F. Schmidt, M. Kraaijveld, R.P. Duin, Feedforward Neural Networks with Random Weights, in: Proc. 11th IAPR IEEE Inter. Conf. on Pattern Recognition, 1992, pp. 1–4.
[14] D. Wang, M. Li, Stochastic configuration networks: fundamentals and algorithms, IEEE Trans. Cybern. 47 (47) (2017) 3466–3479.
[15] B. Widrow, A. Greenblatt, Y. Kim, D. Park, The no-prop algorithm: a new learning algorithm for multilayer neural networks, Neural Netw. 37 (2013) 182–188.