Data-Driven Randomized Learning of Feedforward Neural Networks[⋆]

Grzegorz Dudek[1][0000−0002−2285−0327]

Electrical Engineering Department, Czstochowa University of Technology,
Czstochowa, Poland
dudek@el.pcz.czest.pl

Abstract. Randomized methods of neural network learning suffer from a problem with the generation of random parameters as they are difficult to set optimally to obtain a good projection space. The standard method draws the parameters from a fixed interval which is independent of the data scope and activation function type. This does not lead to good results in the approximation of the strongly nonlinear functions. In this work, a method which adjusts the random parameters, representing the slopes and positions of the sigmoids, to the target function features is proposed. The method randomly selects the input space regions, places the sigmoids in these regions and then adjusts the sigmoid slopes to the local fluctuations of the target function. This brings very good results in the approximation of the complex target functions when compared to the standard fixed interval method and other methods recently proposed in the literature.

Keywords: Data-driven randomized learning · Feedforward neural networks · Neural networks with random hidden nodes · Randomized learning algorithms.

1 Introduction

The learning of feedforward neural networks (FNNs) is an optimization process where the error function is highly nonconvex. Flat regions of the error function as well as many local minima and saddle points hinder and slow down the learning. This is because the gradient algorithms commonly used for FNN learning are very sensitive to the surface of the objective function and fall into the traps of the local minima. Moreover, the gradient calculations are time consuming, especially for a complex target function (TF), a big training data set or for a network with many hidden neurons and many layers. In randomized learning the gradient descend methods do not have to be used. The learning process is split into two stages. In the first stage, the parameters of the hidden neurons (single-hidden-layer FNN is considered), i.e. the weights and biases, are randomly selected. They are not trained at all and stay fixed. In the second stage, the output weights,

connecting the hidden layer with the output layer, are trained. The optimization problem becomes convex and can be solved using a standard linear least-squares [1], which is the simplest, most studied and scalable learning procedure to date.

The standard method of generating the parameters of the hidden neurons, i.e. the weights and biases, is to select them randomly with a fixed interval. It was theoretically proven that when the random parameters are selected from a symmetric interval according to any continuous sampling distribution, the FNN is a universal approximator [2]. The problem with selection of the appropriate interval for the parameters has not yet been solved, and is considered to be one of the most important research gaps in the field of randomized algorithms for NN training [3], [4]. In many practical applications of FNN with random parameters this interval is set as $[-1, 1]$ without any justification, regardless of the problem type (classification or regression), data, and activation function type. Some works have shown that such an interval is misleading because the network is unable to model nonlinear maps, no matter how many training samples are provided or what size networks are used [5]. So, the optimization of this interval is recommended for a specified task [6], [2]. Such a way of improving the performance of the FNN with random parameters has been used in many works, e.g. [7].

In [8] it was noted that the weights and biases have different meanings, i.e. weights represent the sigmoid slope and biases represent its shift, and therefore they should not be generated from the same interval. The method proposed in [8] 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 a surface approximating a nonlinear TF. The weights and biases are dependent on the input data range and activation function type. This leads to an improvement in the approximation performance of the network. Another approach for generating the random parameters was proposed in [9]. This method, firstly, selects at random the slope angles of the sigmoids from the interval adjusted to the TF fluctuations, then rotates the sigmoids randomly and finally shifts them into the input space according to the data distribution. This gives much better results than the standard approach with fixed intervals.

In this work we do not select the hidden neurons parameters from specified intervals. Instead, we propose to adjust them to the local features of the TF. The proposed method selects the input space region by randomly choosing one of the training points, then places the sigmoid in this region and adjusts the sigmoid slope to the TF slope in the neighborhood of the chosen point. Combining linearly the randomly placed sigmoids in the input space, we obtain a fitted surface which reflects the TF features in different regions.

2   Randomized Learning of FNNs

A single-hidden-layer FNN for a single output case and an input $\mathbf{x} = [x_1, x_2, ..., x_n]^T \subset \mathbb{R}^n$ is defined by linearly combining m nonlinear transformations of the input $h_i(\mathbf{x})$:

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

where β_i is the weight between the i-th hidden neuron and the output neuron, and $h_i(\mathbf{x})$ is represented by an activation function of the i-th hidden neuron, e.g. a sigmoid:

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

In randomized FNNs, the weights $\mathbf{a}_i = [a_{i,1}, a_{i,2}, \ldots, a_{i,n}]^T$ and bias b_i are generated randomly for each neuron according to any continuous sampling distribution. Usually $a_{i,j} \sim U(a_{min}, a_{max})$ and $b_i \sim U(b_{min}, b_{max})$.

Note that the activation function (2) applies some nonlinearity on a random linear combination of the input vector. As a result, the activation function is randomly located in the space, has random slope and rotation.

The hidden layer output matrix for N training samples is:

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & \ldots & h_m(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ h_1(\mathbf{x}_N) & \ldots & h_m(\mathbf{x}_N) \end{bmatrix} \tag{3}$$

where the i-th column of $\mathbf{H}$ is the i-th hidden node output vector with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, and $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_m(\mathbf{x})]$ is a nonlinear mapping from n-dimensional input space to m-dimensional feature space, wherein, typically $m \gg n$.

The parameters of the hidden neurons, $a_{i,j}$ and b_i are fixed, so, the matrix $\mathbf{H}$ is calculated only once and remains unchanged.

The output weights β_i are determined by solving the following linear problem:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{Y} \tag{4}$$

where $\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_m]^T$ is a vector of output weights and $\mathbf{Y} = [y_1, y_2, \ldots, y_N]^T$ is a vector of target outputs.

A least mean squares solution of (4) can be achieved within a single learning step by using the MoorePenrose pseudoinverse $\mathbf{H}^+$ of matrix $\mathbf{H}$:

$$\boldsymbol{\beta} = \mathbf{H}^+ \mathbf{Y} \tag{5}$$

In the above described randomized learning there are five hyperparameters which influence strongly on the approximation abilities of the network. They are: the number of hidden nodes m and the bounds of the scopes for weights

and biases, i.e. $a_{min}, a_{max}, b_{min}$ and b_{max}. In most of the works on randomized algorithms for FNNs, the bounds of parameters are selected as fixed, regardless of the data and activation function types. Typically $a_{min} = b_{min} = -1$ and $a_{max} = b_{max} = 1$. The hidden neuron sigmoids, whose linear combination (2) builds the function fitting data, should deliver the nonlinear fragments, avoiding their saturated fragments, and so achieve the required accuracy of approximation. As demonstrated in [8] when using typical interval for random parameters, $[-1, 1]$, the sigmoids are not distributed properly in the input space and their steepness does not correspond to the TF steepness. In some works, the authors optimize the interval for the random parameters by searching for its bounds $[-u, u]$.

To improve the randomized learning performance, the method proposed in [8] randomly generates the weights and biases of the hidden nodes, depending on the input data range and activation function type, in such a way so as to introduce the nonlinear fragments of the activation functions in the input space region containing the data points. Additionally the slopes of the activation functions are adjusted to the TF complexity. According to the proposed method the weights of the i-th hidden node are calculated as follows:

$$a_{i,j} = \zeta_j \frac{\Sigma_i}{\sum\limits_{l=1}^{n} \zeta_l}, \quad j = 1, 2, ..., n \tag{6}$$

where $\zeta_1, \zeta_2, \ldots, \zeta_n \sim U(-1, 1)$ are i.i.d. numbers and Σ_i is the sum of weights of the i-th node, which is randomly chosen from the interval:

$$|\Sigma_i| \in \left[\ln\left(\frac{1-r}{r}\right), s \cdot \ln\left(\frac{1-r}{r}\right) \right] \tag{7}$$

There are two parameters which decide on the activation function slope: $r \in (0, 0.5)$ and $s > 1$. Specifically, they determine two boundary sigmoids between which the activation functions are randomly generated. These parameters are adjusted to the data in cross-validation.

The biases of the hidden nodes are determined setting the inflection points of the sigmoids at some points $\mathbf{x}^*$ randomly selected from the input space or, alternatively, randomly chosen from the training set. The bias of the i-th node is calculated as follows:

$$b_i = -\mathbf{a}_i^T \mathbf{x}^* \tag{8}$$

From the above equations we can conclude that the new approach to selection of the random parameters is a radical departure from the standard approach. Instead of generating both weights and biases from the fixed interval, in the new approach, we first generate the sum of the all node weights from the interval (7), and then randomly generate individual weights from (6). In the next step, the bias is generated from (8) on the basis of randomly chosen point $\mathbf{x}^*$ and weight vector $\mathbf{a}_i$. The derivations of the above equations and more detailed discussion on this topic, including other activation function types, can be found in [8].

Another method for improving the performance of FNN randomized learning was proposed in [9]. Firstly, it randomly choses the slope angles of the hidden neurons activation functions from an interval adjusted to the complexity of the TF. Then, the activation functions are randomly rotated around the y-axis and finally, they are distributed across the input space according to data distribution. For complex TFs, with strong fluctuations, the proposed method gives incomparably better results than the standard approach with the fixed interval for the random parameters. This is because it adjusts the slopes of the activation functions to the data and introduces their steepest fragments into the input space, avoiding their saturation fragments.

In this approach, the weights of the i-th hidden node are calculated from:

$$a_{i,j} = -4\frac{a'_{i,j}}{a'_{i,0}}, \quad j = 1, 2, ..., n \tag{9}$$

where $a'_{i,j}$ are components of the normal vector $\mathbf{n}$ to the hyperplane, which is tangent to the sigmoid at their inflection points.

The angle between the normal vector $\mathbf{n}$ and the unit vector in the direction of the y-axis, α, is randomly selected from the interval $(\alpha_{min}, \alpha_{max})$. The bounds of this interval are adjusted to the TF in cross-validation. These bounds control the slopes of the sigmoids and thus the flexibility of the model. The rotation of the individual sigmoid is random, determined by choosing randomly the components of the normal vector $a'_1, \ldots, a'_n \sim U(-1, 1)$ and calculating component a'_0 from:

$$a'_0 = (-1)^c \frac{\sqrt{(a'_1)^2 + ... + (a'_n)^2}}{\tan \alpha} \tag{10}$$

where $c \sim U\{0, 1\}$.

To distribute the sigmoids across the input space their biases are calculated from (8), where $\mathbf{x}^*$ are the randomly chosen training points. Details of this method can be found in [9].

The methods proposed in [8] and [9] allow us to control the slope of the sigmoids forming the fitted function, and hence the degree of generalization of the network and bias-variance tradeoff of the model. In Fig. 1, the approximation of the highly nonlinear function is shown for the standard method of generating random parameters from the fixed intervals $[-1, 1]$ and for the method proposed in [9] (let us denote this method with the acronym RARSM, i.e. random sigmoid slope angle, rotation and shift method). The TF is in the form:

$$g(x) = \sin(20 \cdot \exp(x)) \cdot x^2 \tag{11}$$

The FNNs learns from a training set containing 5000 points (x_l, y_l), where x_l are uniformly randomly distributed on $[0, 1]$ and y_l are calculated from (11) and then distorted by adding the uniform noise distributed in $[-0.2, 0.2]$. The test set of the same size expresses the true TF (11). The outputs are normalized in the range $[0, 1]$.

In both cases 35 hidden neurons were used. In RARSM $\alpha_{min} = 30°$ and $\alpha_{max} = 90°$. The sigmoids distributed in the input interval, which is shown as

a gray field, are shown in the middle panel of Fig. 1. After weighing them by
the output weights β_i we obtain the curves shown in the bottom panel. The
sum of these curves gives the fitted functions, which are drawn with solid lines
in the upper panel. Note that the sigmoids generated from the fixed internal
$[-1, 1]$ are too flat and their steepest fragments, around their inflection points,
do not correspond to the steep fragments of the TF. As a result, they cannot
be combined to obtain the TF, even when we increase the number of neurons
to several hundreds or even thousands. In a completely different manner the
sigmoids are generated by the RARSM. As we can see from the middle right
panel of Fig. 1, the sigmoids have their steepest fragments inside the input
interval. Their slopes are also fitted to the TF. This results in a reduction the
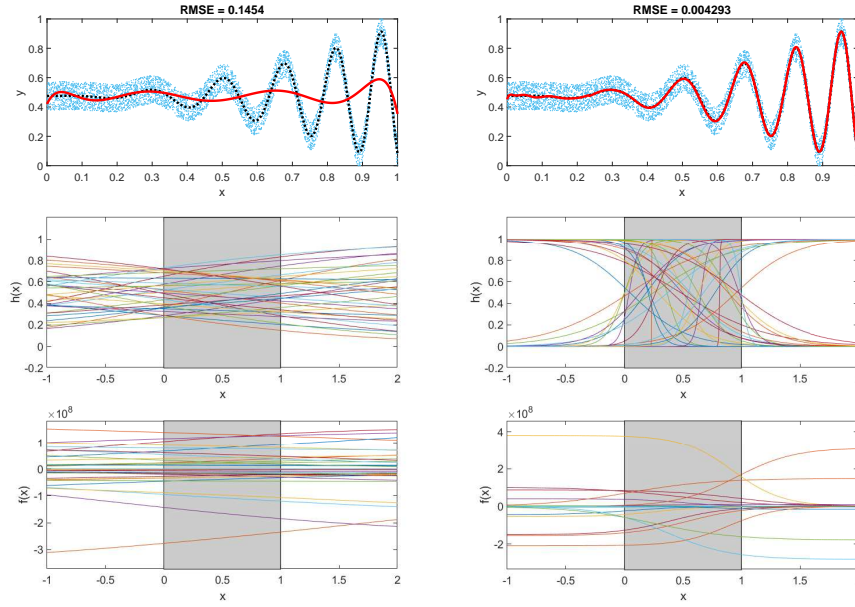error from $RMSE = 0.1454$ for the fixed interval to $RMSE = 0.0043$.

Fig. 1. Fitted curves (upper panel), hidden node sigmoids (middle panel) and weighted
sigmoids (bottom panel) for the standard method (left panel) and the RARSM method
(right panel).

3   Data-Driven Generation of Hidden Nodes Parameters

In the above described methods of FNN randomized learning, we can observe an
evolution. The first step of this evolution is the completely random generation of
the hidden node weights and biases, both from a fixed interval, typically $[-1, 1]$,
according to any continuous sampling distribution, usually a uniform one. In the

second step, we try to find the best interval for the weights so as to match the slopes of sigmoids to the TF complexity. After adjusting slopes, we calculate the biases in order to distribute the sigmoids randomly across the input space. The interval for the weights is dependent on the two hyperparameters, r and s, which are searched in the cross-validation. In the third step of the evolution, we introduce hyperparameters describing the sigmoid shape which are more intuitive than r and s, i.e the limit slope angles of the sigmoids. These hyperparameters can be adjusted to the TF complexity or their default values can be applied, $\alpha_{min} = 0°$ and $\alpha_{max} = 90°$. The forth step of the evolution, proposed in this work, is adjusting the sigmoids individually to the local complexity of the TF. The weights and biases of the sigmoids are no longer random, but they are adjusted to data in randomly chosen regions of the input space. So, each sigmoid models locally the TF in the neighborhood of a randomly selected training point. The fitted function is constructed typically as a linear combination of the sigmoids (1). The weights in this combination are calculated according to (8) using the MoorePenrose pseudoinverse. The idea behind the proposed method is shown on the single-variable TF example in subsection 3.1, and its multivariable extension is presented in subsection 3.2.

3.1   The Idea behind the Method

The idea behind the proposed method can be exemplified by the approximation of the single-variable TF (11). Sigmoids are used as hidden nodes activation functions:

$$h(x) = \frac{1}{1 + \exp(-(ax + b))} \tag{12}$$

where a is a weight controlling a slope of the sigmoid and b is a bias shifting the sigmoid along the x-axis.

It would be convenient to place the sigmoids in the input space and adjust their slopes in such a way that they correspond to the TF fluctuations. To do so, let us select randomly training point x^* and find its k nearest neighbors. These $k + 1$ points form the neighborhood of x^*, $\Psi(x^*)$, and express the local features of the TF around x^*. Now, let us fit to these points straight line T:

$$y = a'x + b' \tag{13}$$

Note that coefficient a' expresses the slope of the line which corresponds to the slope of the TF in $\Psi(x^*)$.

Let set some sigmoid S in the input space in such a way that its inflection point P is in x^*. Remembering that the sigmoid value for the inflection point is 0.5, we get:

$$h(x^*) = \frac{1}{1 + \exp(-(ax^* + b))} = 0.5 \tag{14}$$

Let us assume that the slope of S at P is the same as the slope of the line T. This means that the derivative of S at $P = x^*$ is equal to the derivative of the line T, thus:

$$ah(x^*)(1 - h(x^*)) = a' \tag{15}$$

Substituting $h(x^*) = 0.5$ from (14) into (15) we obtain:

$$a \cdot 0.5 \cdot (1 - 0.5) = a' \tag{16}$$

and finally weight a of sigmoid S is:

$$a = 4a' \tag{17}$$

From (14) we also obtain:

$$b = -ax^* \tag{18}$$

So, the sigmoid which models locally the TF in the neighborhood of x^* has weight a dependent on slope parameter a' of the line fitted to $\Psi(x^*)$. Additionally bias b is dependent on slope parameter a' as well as on point x^*.

Fig. 2 shows an example sigmoids set according to the three randomly selected points x^* and their neighborhoods composed of x^* and their ten closest points. Note that the sigmoids reflect the slopes of the TF around points x^*.
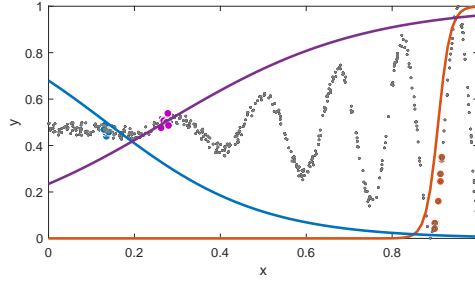

Fig. 2. Examples of setting the sigmoids to the neighborhoods (larger colored points) of the three randomly selected training points.

Fig. 3 shows function (12) approximation when using the proposed method with 25 hidden neurons and 100 nearest neighbors. Compare Fig. 3 with Fig. 1 and note the similar errors for the proposed method and RARSM. Note also the different distribution of the the sigmoids which have different slopes in these both cases. In the proposed method the steeper sigmoids are generated at the right border of the input interval, where the fluctuations of the TF are stronger. At the left border, where the TF is flat, the sigmoids are less steep. While in the

RARSM the steepness of the sigmoids does not depend on the local TF features and is similar in each region of the input space.

It is worth mentioning that the proposed method needs only 25 hidden neurons to obtain the same error level as the RARSM with 35 neurons. The RMSE for the RARSM with 25 neurons was 0.031, which is almost seven times larger than for the proposed method. The smaller number of neurons in the proposed approach is due to the fact that the sigmoids are better fitted to TF fluctuations.
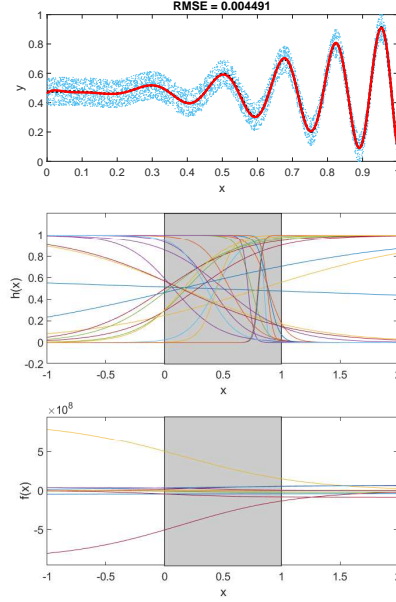

Fig. 3. Fitted curve (upper panel), hidden node sigmoids (middle panel) and weighted sigmoids (bottom panel) for the proposed method.

3.2   Multivariable Function Fitting

In this subsection, the proposed method is extended to the general case of multivariable function fitting. In this case the TF is a function of n input variables included in the vector $\mathbf{x} = [x_1, x_2, ..., x_n]^T \subset \mathbb{R}^n$. Similarly to the single-variable case, we place the sigmoids in the input space and adjust their slopes in such a way that they correspond to the TF fluctuations. The slopes of the TF in some point $\mathbf{x}^*$ are approximated by hyperplane T fitted to the neighborhood $\Psi(\mathbf{x}^*)$ which includes this point and its k nearest neighbors among the training points. This hyperplane is of the form:

$$y = a'_1 x_1 + a'_2 x_2 + ... + a'_n x_n + b' \tag{19}$$

where coefficient a'_j expresses a slope of hyperplane T in the j-th direction.

Let us consider a sigmoid S which has one of its inflection points, P, in the randomly selected training point $\mathbf{x}^*$:

$$h(\mathbf{x}^*) = \frac{1}{1 + \exp\left(-\left(\mathbf{a}^T\mathbf{x}^* + b\right)\right)} = 0.5 \tag{20}$$

where $\mathbf{a} = [a_1, a_2, ..., a_n]^T \subset \mathbb{R}^n$.

The slope of this sigmoid S at point $\mathbf{x}^*$ in the j-th direction is expressed by a partial derivative:

$$\frac{\partial h(\mathbf{x}^*)}{\partial x_j} = a_j h(\mathbf{x}^*)(1 - h(\mathbf{x}^*)) \tag{21}$$

We want the slope of sigmoid S at $\mathbf{x}^*$ in the j-th direction to be the same as the slope of the TF in this point, which is approximated by the hyperplane T slope a'_j. Thus:

$$a_j h(\mathbf{x}^*)(1 - h(\mathbf{x}^*)) = a'_j \tag{22}$$

After substituting $h(\mathbf{x}^*) = 0.5$ from (20) into (22) we obtain:

$$a_j = 4a'_j, \quad j = 1, 2, ..., n \tag{23}$$

Directly from (20) we also obtain:

$$b = -\mathbf{a}^T\mathbf{x}^* \tag{24}$$

Note that the weights of the hidden neuron, a_j, expressing the slopes of the sigmoid in all n directions, are proportional to the hyperplane T coefficients corresponding to these directions. These coefficients approximate the TF slopes at the randomly selected point $\mathbf{x}^*$. The bias of sigmoid S is a linear combination of point $\mathbf{x}^*$ and sigmoid weights $\mathbf{a}$. The sigmoid S reflects the local features of the TF around point $\mathbf{x}^*$. Selecting randomly a set of points $\mathbf{x}^*$ we generate a set of sigmoids reflecting the local features of the TF in different regions. These sigmoids are the basis functions which are linearly combined to get the fitted function approximating the TF. The weights in this combination are calculated using the MoorePenrose pseudoinverse (8).

The proposed method places the sigmoids in the input space setting their inflection points on the randomly selected training points $\mathbf{x}^*$ and adjusting the sigmoid slopes to the slopes of the TF around these points. The TF slope is approximated by hyperplane T fitted to the neighborhood of $\mathbf{x}^*$, i.e. $\mathbf{x}^*$ and its k nearest neighbors. The TF is defined in $(n + 1)$-dimensional space. To define hyperplane T in such a space at least $n + 1$ points are needed. So, the number of nearest neighbors k should not be less than n.

The number of nearest neighbors controls the bias-variance tradeoff of the model. The optimal value of k depends on the random error observed in the data and the TF complexity. When the training points represent a TF with low error, the number of nearest neighbors k should be lower. For higher errors a low value

of k leads to overfitting. On the other hand, a too large k causes underfitting. This hyperparameter should be tuned in the cross-validation to the given data as well as the second hyperparameter, the number of hidden nodes m. In the experimental part of the work, the impact of the noise level in the data on the hyperparameters is investigated.

Algorithm 1 summarizes the proposed method.

Algorithm 1 Data-Driven Generating the Parameters of FNN Hidden Nodes

---

Input:

Number of hidden nodes m
Number of nearest neighbors $k \geq n$
Training set $\Phi = \{(\mathbf{x}_l, y_l) | \mathbf{x}_l \in \mathbb{R}^n, y_l \in \mathbb{R}, l = 1, 2, \ldots, N\}$

Output:

Weights $\mathbf{A} = \begin{bmatrix} a_{1,1} & \ldots & a_{m,1} \\ \vdots & \ddots & \vdots \\ a_{1,n} & \ldots & a_{m,n} \end{bmatrix}$

Biases $\mathbf{b} = [b_1, \ldots, b_m]$

Procedure:

for $i = 1$ **to** m **do**
    Choose randomly $\mathbf{x}^* = \mathbf{x}_l \in \Phi$, where $l \sim U\{1, 2, \ldots, N\}$
    Create the set $\Psi(\mathbf{x}^*)$ containing $\mathbf{x}^*$ and its k nearest neighbors in Φ
    Fit the hyperplane to $\Psi(\mathbf{x}^*)$:

$$y = a_1' x_1 + a_2' x_2 + \ldots + a_n' x_n + b'$$

Calculate the weights for the i-th node:

$$a_{i,j} = 4a_j', \quad j = 1, 2, \ldots, n$$

Calculate the bias for the i-th node:

$$b_i = -\sum_{j=1}^{n} a_{i,j} x_j^*$$

end for

---

4   Simulation study

This section reports some experimental results of the proposed method, including the impact of the noise level in data on the hyperparameters and performance evaluation. In the first experiment we analyze how the noise disturbing data affects the hyperparameters of the proposed methods. Simulations were carried out on a two-variable TF of the form:

$$g(\mathbf{x}) = \sin\left(20 \cdot \exp\left(x_1\right)\right) \cdot x_1^2 + \sin\left(20 \cdot \exp\left(x_2\right)\right) \cdot x_2^2 \qquad (25)$$

On the basis of this function, training set Φ was created containing 5000 points $(\mathbf{x}_l, y_l)$, where the components of $\mathbf{x}_l$ are independently uniformly randomly distributed on $[0, 1]$ and y_l are generated from (25), then normalized to the range $[0, 1]$ and finally distorted by adding the uniform noise distributed in $[c, c]$. The testing set represents the TF without noise normalized into $[0, 1]$. It contains 10,000 points distributed regularly on a grid in the input space.

To introduce the noise of different level to data we changed the noise boundary c from 0 to 1 with steps of 0.1. This translates into a noise level from 0 to 100%, defined as the ratio of the noise range to the TF range. The TF and the data points for two noise levels, $c = 0.2$ and $c = 1$, are shown in Fig. 4. For each noise level we changed the neighborhood $\Psi(x^*)$ size, $k' = 3, 5, 7, 10, 20, ..., 100$, where $k' = k + 1$, keeping the fixed number of hidden nodes $m = 300$. For each setting, 100 independent trials of FNN training were performed.

The left panel of Fig. 5 shows the test root-mean-square error (RMSE) for a different noise levels and neighborhood size. On the right panel, the boxplots are shown for three noise levels: $c = 0.1, c = 0.5$ and $c = 1$. The optimal neighborhood size k' was 20 for the lower noise levels ($c \leqslant 0.5$) and 30 for the higher noise levels ($c > 0.5$). The model tends to overfit for the lower than the optimal values of k', and for higher values it tends to underfit.

In the next step, for each noise level we changed the number of hidden nodes $m = 50, 100, ..., 500$ keeping a fixed size of the neighborhood $k' = 20$. Fig. 6 shows the test RMSE surface and the boxplots for this case. The optimal node numbers were: $m = 250$ in the broad range of noise level from 0.1 to 0.7, $m = 200$ for $c = 0.8$ and $c = 0.9$, and $m = 50$ for $c = 1$. We can observe from Figs. 5 and 6 that when the noise level is small, the underestimation of both hyperparameters, k' and m, is more disadvantageous in terms of the error than overestimation.

In the next experiments, we compare the results of the proposed method with the methods described in Section 2:

- FIM - fixed interval method, standard method with fixed interval for the random parameters $[-1, 1]$,
- OIM - optimized interval method, the method with the optimized interval for the random parameters $[-u, u]$, where u in our simulations was selected from a given set $\{0.1, 0.5, 1, 2,$ $3, 4, 5, 10, 15, 20, 30, 40, 50, 100, 200, 300, 400, 500\}$,
- rsM - the method proposed in [8] with two parameters, r and s, which were selected from the sets: $r \in \{0.0001, 0.001, 0.01, 0.015, 0.02, 0.3, 0.4, 0.5\}$ and $s \in \{2, 4, 6, 8, 10, 20, 30\}$,
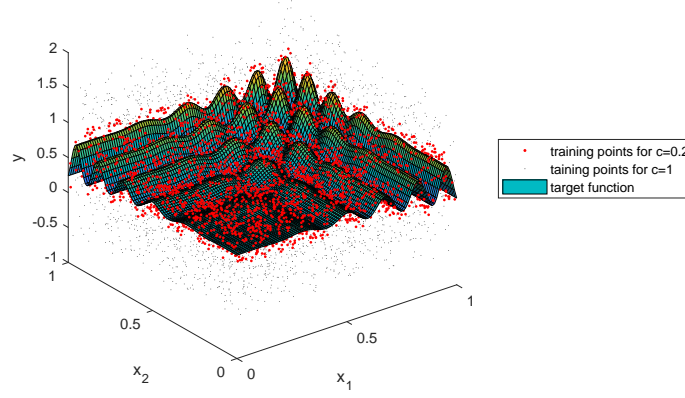
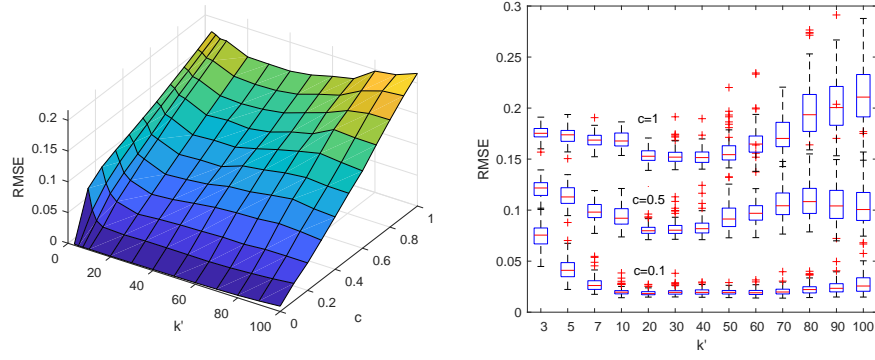**Fig. 4.** Target function and the training points for $c = 0.2$ and $c = 1$.

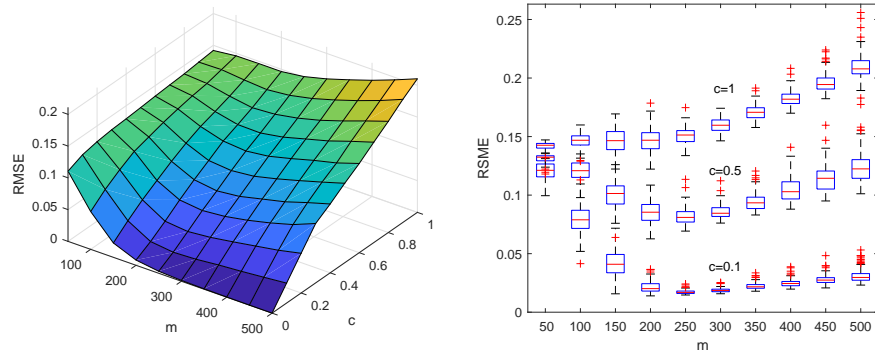**Fig. 5.** RMSE for different neighborhood size k' and noise level c at $m = 300$.

Fig. 6. RMSE for different number of nodes m and noise level c at $k' = 20$.

– RARSM - random slope angle, rotation and shift method, proposed in [9] with two parameters selected from the sets: $\alpha_{min} \in \{0°, 5°, ..., 85°\}$ and $\alpha_{max} \in \{\alpha_{min} + 5°, \alpha_{min} + 10°, ..., 90°\}$,
– D-DM - data-driven method proposed in this work with parameter k' which was selected from the set $\{5, 10, ..., 50\}$.

For each method of random parameter generation, the number of hidden nodes was selected from the set $\{50, 100, ..., 1000\}$. The selection of the optimal hyperparameter values was carried out in the grid search procedure using 10-fold cross-validation. For the optimal values of the hyperparameters 100 independent trials of training were performed and test errors were calculated.

First, we use function (25) with error level $c = 0.2$ as the test function. The left panel of Fig. 7 shows the cross-validation errors for different numbers of nodes at optimal values of other hyperparameters. As you can see in this figure, both FIM and OIM failed completely. Optimization of the interval bounds $[-u, u]$ in OIM brought only a slight improvement in accuracy when compared to FIM. More sophisticated methods, rsM, RARSM and D-DM, are incomparably more accurate. Note that the proposed D-DM needs the smallest number of neurons to get the best performance when compared to other methods.

The right panel of Fig. 7 shows the boxplots of the test RMSE for 100 trials of the learning sessions carried out at the optimal values of hyperparameters. These simulations are summarized in Table 1. Clearly, from this table, D-DM outperforms the other methods in terms of accuracy.
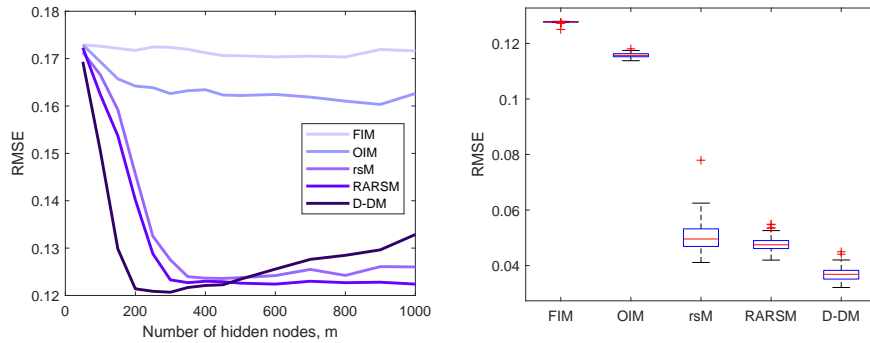

Fig. 7. RMSE for different number of nodes m (left panel) and distribution of the test RMSE for the tested methods (right panel).

In the next experiments we use two multivariable datasets:

– Stock daily stock prices from January 1988 through October 1991, for ten aerospace companies. The task is to aproximate the price of the 10-th company given the prices of the others (950 samples, 9 input variables, source: http://www.keel.es/).

– Kin8nm   a realistic simulation of the forward dynamics of an 8 link all-revolute robot arm. The task is to predict the distance of the end-effector from a target. The inputs are things like joint postions, twist angles, etc. (8192 samples, 8 input variables, source: www.cs.toronto.edu/ delve/data/kin /desc).

The data sets were divided into training sets containing 75% samples selected randomly, and the test sets containing the remaining samples. The test RMSE for both datasets at the optimal values of hyperparameters are visualized by the boxplots in Fig. 8. Table 1 shows the mean RMSE and the optimal hyperparameters of the methods. Note that D-DM demonstrates the best performance compared to other methods. Especially for Kin8nm the significant improvement in accuracy is observed for D-DM. This may be due to the fact that in this case the target function has variable fluctuations. The D-DM, which is designed to deal with such cases, performs best.
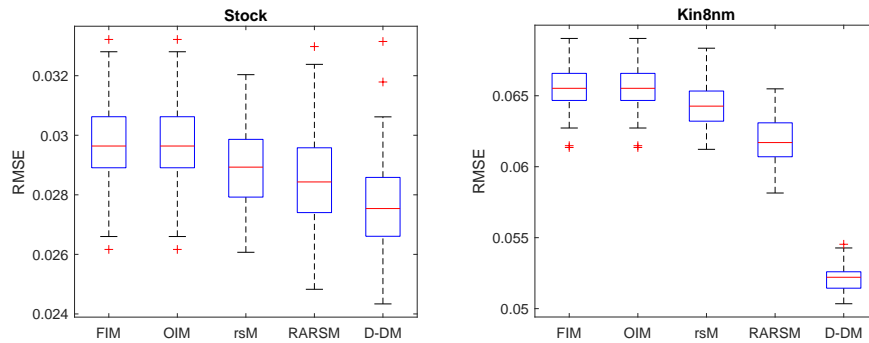

Fig. 8. Distribution of the test RMSE for the Stock and Kin8nm data.

5   Conclusions

The way in which the hidden node parameters are generated is a key issue in the randomized learning of FNN. When these parameters are selected in a standard way from the fixed interval the performance of the network can be weak, especially for complex function fitting.

This work proposes a new approach to generating the parameters of a FNN in randomized learning. The proposed method adjusts the hidden neurons weights and biases, representing the slopes and positions of the sigmoids, to the target function features. The method first randomly selects the input space regions by drawing the points from the training set. Then, the hyperplanes are fitted to the neighborhoods of the selected points and their coefficients are transformed into the sigmoid weights and biases. This results in the placement of the sigmoids

Table 1. Performance comparison of the proposed and comparative methods.

Method	Test RMSE	#nodes	Parameters
Function (25)			
FIM	0.1277 ± 0.00029	800	-
OIM	0.1157 ± 0.00088	1000	$u = 3$
rsM	0.0503 ± 0.00527	450	$r = 0.4, s = 30$
RARSM	0.0477 ± 0.00254	350	$\alpha_{min} = 55°, \alpha_{max} = 70°$
D-DM	0.0370 ± 0.00230	300	$k' = 35$
Stock			
FIM	0.0296 ± 0.00141	200	-
OIM	0.0296 ± 0.00141	200	$u = 1$
rsM	0.0289 ± 0.00138	200	$r = 0.001, s = 2$
RARSM	0.0285 ± 0.00156	200	$\alpha_{min} = 45°, \alpha_{max} = 70°$
D-DM	0.0277 ± 0.00150	250	$k' = 30$
Kin8nm			
FIM	0.0655 ± 0.00153	1300	-
OIM	0.0655 ± 0.00153	1300	$u = 1$
rsM	0.0643 ± 0.00157	1300	$r = 0.4, s = 8$
RARSM	0.0618 ± 0.00157	1300	$\alpha_{min} = 35°, \alpha_{max} = 55°$
D-DM	0.0523 ± 0.00081	900	$k' = 60$

in the selected regions of the input space and the adjustment of their slopes to the local fluctuations of the target function. As simulation research has shown, such a method of generating random parameters brings very good results in the approximation of the complex target functions when compared to the standard fixed interval method and other methods proposed recently in the literature.

Future work will focus on further analysis and improvement of the proposed method as well as rsM and RARSM, and their adaptation to classification problems.

References

1. Principe, J., Chen, B.: Universal approximation with convex optimization: gimmick or reality? IEEE Comput. Intell. Mag. **10**,68–77 (2015)
2. Husmeier, D.: Random vector functional link (RVFL) networks. In: Neural Networks for Conditional Probability Estimation: Forecasting Beyond Point Predictions, chapter 6, 87–97, Springer-Verlag London (1999)
3. Zhang, L., Suganthan, P.: A Survey of randomized algorithms for training neural networks. Information Sciences **364**, 146–155 (2016)
4. Weipeng, C., Wang, X., Ming, Z., Gao, J.: A review on neural networks with random weights. Neurocomputing **275**, 278–287 (2018)
5. Li, M., Wang, D.: Insights into randomized algorithms for neural networks: Practical issues and common pitfalls. Information Sciences **382–383**, 170–178 (2017)
6. Pao, Y.-H., Park, G.H., Sobajic, D.J.: Learning and generalization characteristics of the random vector functional-link net. Nurocomputing **6**(2), 163–180 (1994)

7. Li, M., Huang, C., Wang, D.: Robust stochastic configuration networks with maximum correntropy criterion for uncertain data regression. Information Sciences **473**, 73–86 (2019)
8. Dudek, G.: Generating random weights and biases in feedforward neural networks with random hidden nodes. Information Sciences **481**, 33–56 (2019)
9. Dudek, G.: Improving randomized learning of feedforward neural networks by appropriate generation of random parameters. In: 15th International Work-Conference on Artificial Neural Networks, (2019) (in print)