Data-Driven Learning of Feedforward Neural Networks with Different Activation Functions*

Grzegorz Dudek[0000−0002−2285−0327]

Electrical Engineering Faculty, Czestochowa University of Technology,
Czestochowa, Poland
grzegorz.dudek@pcz.pl

Abstract. This work contributes to the development of a new data-driven method (D-DM) of feedforward neural networks (FNNs) learning. This method was proposed recently as a way of improving randomized learning of FNNs by adjusting the network parameters to the target function fluctuations. The method employs logistic sigmoid activation functions for hidden nodes. In this study, we introduce other activation functions, such as bipolar sigmoid, sine function, saturating linear functions, reLU, and softplus. We derive formulas for their parameters, i.e. weights and biases. In the simulation study, we evaluate the performance of FNN data-driven learning with different activation functions. The results indicate that the sigmoid activation functions perform much better than others in the approximation of complex, fluctuated target functions.

Keywords: Data-driven learning · Feedforward neural networks · Randomized learning algorithms.

1 Introduction

FNNs are widely used as predictive models to fit data distribution. They learn using gradient descent methods and ensure a universal approximation property. However, gradient-based algorithms suffer from many drawbacks which make the learning process ineffective and time-consuming. This is because gradient learning is sensitive to local minima, flat regions, and saddle points of the loss function. Moreover, its application is time-consuming for complex target functions (TFs), big data, and large FNN architectures. Randomized learning was proposed as an alternative to gradient-based learning. In this approach, the parameters of the hidden nodes are selected randomly from any interval, and stay fixed. Only the output weights are learned. The optimization problem in randomized learning becomes convex and can be solved by a standard linear least-squares method [1]. This leads to very fast training. The the universal approximation property is kept when the random parameters are selected from a symmetric interval according to any continuous sampling distribution [2]. The main problems in randomized

* Supported by Grant 2017/27/B/ST6/01804 from the National Science Centre, Poland.

learning are [3], [4]: how to select the interval and distribution for the random parameters, and whether the weights and biases should be chosen from the same interval and distribution.

It was shown in [5] and [6] that the weights and biases of hidden nodes have different functions and should be selected separately. The weights decide about the activation function (AF) slopes and should reflect the TF complexity, while the biases decide about the AF shift and should ensure the placement of the most nonlinear fragments of AFs into the input hypercube. These fragments are most useful for modeling TF fluctuations. The method proposed in [5] selects the proper interval for the weights based on AF features and TF properties. The biases are calculated based on the weights and data scope. This approach introduces the AFs into the input hypercube and adjusts the interval for weights to TF complexity. In [6], instead of generating the weights, the slope angles of AFs were randomly selected. This changed the distribution of weights, which typically is a uniform one. This new distribution ensured that the slope angles of AFs were uniformly distributed, which improved results by preventing overfitting, especially for highly nonlinear TFs.

To improve further FNN randomized learning, in [7], a D-DM was proposed. This method introduces the AFs into randomly selected regions of the input space and adjusts the AF slopes to the TF slopes in these regions. As a result, the AFs mimic the TF locally, and their linear combination approximates smoothly the entire TF. This work contributes to the development of data-driven FNN learning by introducing different AFs, i.e. bipolar sigmoid, sine function, saturating linear functions, reLU, and softplus. For each AF, the formulas for weights and biases are derived.

The remainder of this paper is structured as follows. In Section 2, the framework of D-DM is presented. The formulas for hidden nodes parameters for different AFs are derived in Section 3. The performance of FNN data-driven learning with different AFs is evaluated in Section 4. Finally, Section 5 concludes the work.

2 Framework of the Data-Driven FNN Learning

Let us consider a shallow FNN architecture with n inputs, a single-hidden layer, and a single output. AFs of hidden nodes, $h(\mathbf{x})$, map nonlinearly input vectors $\mathbf{x} = [x_1, x_2, ..., x_n]^T \in \mathbb{R}^n$ into an m-dimensional feature space. An output node combines linearly m nonlinear transformations of the inputs. The function expressed by this FNN has the form:

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

where β_i is the output weight linking the i-th hidden node with the output node.

Such FNN architecture has a universal approximation property, even when the hidden layer parameters are not trained but generated randomly from the proper distribution [8], [2].

The output weights $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_m]^T$ can be determined by solving the following linear problem: $\mathbf{H}\boldsymbol{\beta} = \mathbf{Y}$, where $\mathbf{H} = [\mathbf{h}(\mathbf{x}_1), \mathbf{h}(\mathbf{x}_2), ..., \mathbf{h}(\mathbf{x}_N)]^T \in \mathbb{R}^{N \times m}$ is the hidden layer output matrix, and $\mathbf{Y} = [y_1, y_2, ..., y_N]^T$ is a vector of target outputs. The optimal solution for $\boldsymbol{\beta}$ is given by:

$$\boldsymbol{\beta} = \mathbf{H}^{+}\mathbf{Y} \tag{2}$$

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

The hidden node parameters, i.e. weights $\mathbf{a} = [a_1, a_2, ..., a_n]^T$ and a bias b, control slopes and position of AF in the input space. For a sigmoid AF given by the formula:

$$h(\mathbf{x}) = \frac{1}{1 + \exp\left(-\left(\mathbf{a}^T\mathbf{x} + b\right)\right)} \tag{3}$$

weight a_j decides about the sigmoid slope in the j-th direction and bias b decides about the sigmoid shift along a hyperplane containing all x-axes. The appropriate selection of the slopes and shifts of all sigmoids determine the fitting accuracy of FNN to the TF. To adjust the sigmoids to the local features of the TF, in [7], a D-DM for FNN learning was proposed. This method selects an input space region by randomly choosing one of the training points for each sigmoid. Then, it places the sigmoid in this region and adjusts the sigmoid slopes to the TF slopes in the neighborhood of the chosen point. By combining linearly all the sigmoids randomly placed in the input space, we obtain a fitted surface which reflects the TF shape in different regions.

The D-DM algorithm, in the first step, selects randomly training point $\mathbf{x}^*$. Then, sigmoid S is placed in the input space in such a way that one of its inflection points, P, is in $\mathbf{x}^*$. The sigmoid value at the inflection point is 0.5:

$$h(\mathbf{x}^*) = \frac{1}{1 + \exp\left(-\left(\mathbf{a}^T\mathbf{x}^* + b\right)\right)} = 0.5 \tag{4}$$

From this equation we obtain the sigmoid bias as:

$$b = -\mathbf{a}^T\mathbf{x}^* \tag{5}$$

The slopes of sigmoid S are adjusted to the TF slopes in $\mathbf{x}^*$. The TF slopes in $\mathbf{x}^*$ are estimated by fitting hyperplane T to the neighborhood of $\mathbf{x}^*$. The neighborhood, $\Psi(\mathbf{x}^*)$, contains point $\mathbf{x}^*$ and k training points nearest to it. Hyperplane T has the form:

$$y = a'_1 x_1 + a'_2 x_2 + ... + a'_n x_n + b' \tag{6}$$

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

We assume that sigmoid S is tangent to hyperplane T in point $\mathbf{x}^*$. This means that the partial derivatives of S and T in $\mathbf{x}^*$ are the same. Comparing the formulas for partial derivatives of both functions, we obtain an equation for the sigmoid weights (see [7] for details):

$$a_j = 4a'_j, \quad j = 1, 2, ..., n \tag{7}$$

To generate all the FNN hidden nodes, the D-DM algorithm repeats the procedure described above m times. So, for each node it randomly selects training point $\mathbf{x}^*$, fits hyperplane T to its neighborhood $\Psi(\mathbf{x}^*)$, calculates weights a_j according to (7), and calculates biases b according to (5). Finally, it calculates hidden layer output matrix $\mathbf{H}$, and output weights from (2). The resulting function, $\varphi(\mathbf{x})$, constructed in line with such data-driven learning, reflects TF fluctuations.

The D-DM has two hyperparameters: the number of hidden nodes m and neighbourhood size k. They control the fitting performance of the model and its bias-variance tradeoff. Their optimal values for a given TF should be tuned during cross-validation.

3 Data-Driven FNN Learning with Different Activation Functions

When we employ other AFs instead of logistic sigmoids, the projection matrix $\mathbf{H}$ changes in a way which can entail changes in the approximation properties of the model. Using other AFs requires the derivation of new formulas for the hidden node parameters in the following ways.

Bipolar sigmoid SIGMOID_B. Usually the bipolar sigmoid is defined as a hyperbolic tangent function. In this study, we define it slightly differently:

$$h_{sigb}(\mathbf{x}) = \frac{2}{1 + \exp\left(-\left(\mathbf{a}^T\mathbf{x} + b\right)\right)} - 1 \tag{8}$$

D-DM places SIGMOID_B in the input space in such a way that one of its inflection points is in the randomly selected training point, $\mathbf{x}^*$. The SIGMOID_B value at the inflection points is 0, so, $h_{sigb}(\mathbf{x}^*) = 0$. From this equation we obtain the formula for the bias, which is the same as for the unipolar sigmoid (SIGMOID_U), (5).

To find weights a_j, we equate the partial derivatives of SIGMOID_B in $\mathbf{x}^*$ to the partial derivatives of hyperplane T, (6):

$$\frac{\partial h_{sigb}(\mathbf{x}^*)}{\partial x_j} = \frac{1}{2} a_j (1 + h_{sigb}(\mathbf{x}^*))(1 - h_{sigb}(\mathbf{x}^*)) = a'_j \tag{9}$$

From this equation, taking into account that $h_{sigb}(\mathbf{x}^*) = 0$, we obtain:

$$a_j = 2a'_j, \quad j = 1, 2, ..., n \tag{10}$$

Sine function SINE. Let us place the SINE AF, $h_{sin}(\mathbf{x}) = \sin(\mathbf{a}^T\mathbf{x} + b)$, in the input space in such a way that it has one of its inflection point in randomly selected training point $\mathbf{x}^*$. The SINE value in the inflection points is 0, so,

$h_{sin}(\mathbf{x}^*) = 0$. From this equation we obtain the formula for bias, which is the same as for both sigmoid AFs, (5).
To determine equations for the weights for SINE, we equate the partial derivatives of SINE in $\mathbf{x}^*$ to the partial derivatives of hyperplane T, (6):

$$\frac{\partial h_{sin}(\mathbf{x}^*)}{\partial x_j} = a_j \cos(\mathbf{a}^T\mathbf{x}^* + b) = a'_j \tag{11}$$

Taking into account that $\sin(\mathbf{a}^T\mathbf{x}^* + b) = 0$ implies $\cos(\mathbf{a}^T\mathbf{x}^* + b) = 1$, from (11) we obtain:

$$a_j = a'_j, \quad j = 1, 2, ..., n \tag{12}$$

Saturating linear unipolar function SATLIN_U. This is a linearized version of SIGMOID_U defined as follows:

$$h_{satu}(\mathbf{x}) = \begin{cases} 0 & \text{if } z \leq 0 \\ z & \text{if } 0 < z < 1 \\ 1 & \text{if } z \geq 1 \end{cases} \tag{13}$$

where $z = \mathbf{a}^T\mathbf{x} + b$.
SATLIN_U is placed in the input space in such a way that it has a value of 0.5 in $\mathbf{x}^*$. This is analogous to SIGMOID_U to which SATLIN_U has a similar shape. Thus, $\mathbf{a}^T\mathbf{x}^* + b = 0.5$. From this equation we obtain:

$$b = 0.5 - \mathbf{a}^T\mathbf{x}^* \tag{14}$$

We assume that the middle segment of $h_{satu}(\mathbf{x})$, $\mathbf{a}^T\mathbf{x}+b$, has the same slopes as hyperplane T, thus:

$$a_j = a'_j, \quad j = 1, 2, ..., n \tag{15}$$

Saturating linear bipolar function SATLIN_B. This AF is a linearized version of bipolar sigmoid SIGMOID_B:

$$h_{satb}(\mathbf{x}) = \begin{cases} -1 & \text{if } z \leq -1 \\ z & \text{if } -1 < z < 1 \\ 1 & \text{if } z \geq 1 \end{cases} \tag{16}$$

where $z = \mathbf{a}^T\mathbf{x} + b$.
SATLIN_B is placed in the input space in such a way that it has a value of 0 in $\mathbf{x}^*$. Thus, $\mathbf{a}^T\mathbf{x}^* + b = 0$. From this equation we obtain the same formula for a bias as for sigmoid AFs, (5).
As with SATLIN_U, we assume that the middle segment of SATLIN_B has the same slopes as hyperplane T. Thus, weights a_j are the same as the T coefficients, (15).

Rectified linear unit RELU. This is an AF commonly used in deep learning. It is expressed by:

$$h_{reLU}(\mathbf{x}) = \begin{cases} 0 & \text{if } z \leq 0 \\ z & \text{if } z > 0 \end{cases} \tag{17}$$

where $z = \mathbf{a}^T\mathbf{x} + b$.

RELU is composed of two half-hyperplanes: the first being $y = 0$ and the second $y = \mathbf{a}^T\mathbf{x}+b$. D-DM places the RELU AF in the input space so that the second half-hyperplane coincides with hyperplane T. Thus, their coefficients are the same:

$$b = b', \quad a_j = a'_j, \quad j = 1, 2, ..., n \tag{18}$$

Softplus SOFTPLUS. This is a smooth approximation of the RELU. It is expressed by:

$$h_{soft}(\mathbf{x}) = \ln\left(1 + \exp\left(\mathbf{a}^T\mathbf{x} + b\right)\right) \tag{19}$$

For $\mathbf{x} = [0, 0, ..., 0]$ and $b = 0$, the value of $h_{soft}(\mathbf{x}) = \ln(2)$. Let us shift this function in such a way that it has the value of $\ln(2)$ in $\mathbf{x}^*$. In such a case $\ln(1 + \exp(\mathbf{a}^T\mathbf{x}^* + b)) = \ln(2)$. From this equation we obtain a formula for b, which is the same as for the sigmoids (5).

Now, let us assume that the slopes of SOFTPLUS in $\mathbf{x}^*$ are the same as the slopes of T. Equating the partial derivative of both functions we obtain:

$$\frac{\partial h_{soft}(\mathbf{x}^*)}{\partial x_j} = \frac{a_j}{1 + \exp(-(\mathbf{a}^T\mathbf{x}^* + b))} = a'_j \tag{20}$$

From $\ln(1+\exp(\mathbf{a}^T\mathbf{x}^*+b)) = \ln(2)$ we obtain $1+\exp(\mathbf{a}^T\mathbf{x}^*+b) = 2$. Substituting this into (20), we obtain the weights of hidden nodes with SOFTPLUS AFs:

$$a_j = 2a'_j, \quad j = 1, 2, ..., n \tag{21}$$

Table 1 details the hidden nodes parameters determined by D-DM for different AFs. Note that in all cases, weights a_j reflect hyperplane T coefficients a'_j. Biases for all AFs, excluding RELU, are expressed using a dot product of the weight vector and $\mathbf{x}^*$ vector.

Fig. 1 shows AFs of different types introduced into the input space by D-DM. The training points belonging to the neighborhood of $\mathbf{x}^*$, $\Psi(\mathbf{x}^*)$, are shown as red dots. Note that the AFs in all cases have the same slopes in $\mathbf{x}^*$ as the slope of line T, which estimates the TF slope in $\mathbf{x}^*$. D-DM introduces m AFs in different regions of the input space.

Table 1. Hidden nodes parameters for different activation functions.

	Activation function	Weights a_j	Bias b
SIGMOID_U:	$h_{sigu}(\mathbf{x}) = \frac{1}{1+\exp\left(-\left(\mathbf{a}^T\mathbf{x}+b\right)\right)}$	$4a'_j$	$-\mathbf{a}^T\mathbf{x}^*$
SIGMOID_B:	$h_{sigb}(\mathbf{x}) = \frac{2}{1+\exp\left(-\left(\mathbf{a}^T\mathbf{x}+b\right)\right)} - 1$	$2a'_j$	$-\mathbf{a}^T\mathbf{x}^*$
SINE:	$h_{sin}(\mathbf{x}) = \sin(\mathbf{a}^T\mathbf{x}+b)$	a'_j	$-\mathbf{a}^T\mathbf{x}^*$
SATLIN_U:	$h_{satu}(\mathbf{x}) = \begin{cases} 0 & \text{if } z \le 0 \\ z & \text{if } 0 < z < 1 \\ 1 & \text{if } z \ge 1 \end{cases}$	a'_j	$0.5 - \mathbf{a}^T\mathbf{x}^*$
SATLIN_B:	$h_{satb}(\mathbf{x}) = \begin{cases} -1 & \text{if } z \le -1 \\ z & \text{if } -1 < z < 1 \\ 1 & \text{if } z \ge 1 \end{cases}$	a'_j	$-\mathbf{a}^T\mathbf{x}^*$
RELU:	$h_{reLU}(\mathbf{x}) = \begin{cases} 0 & \text{if } z \le 0 \\ z & \text{if } z > 0 \end{cases}$	a'_j	b'
SOFTPLUS:	$h_{soft}(\mathbf{x}) = \ln\left(1 + \exp\left(\mathbf{a}^T\mathbf{x}+b\right)\right)$	$2a'_j$	$-\mathbf{a}^T\mathbf{x}^*$

where a'_j and b' are coefficients of hyperplane T, $y = a'_1x_1 + a'_2x_2 + ... + a'_nx_n + b'$, adjusted to the TF in the neighborhood $\Psi(\mathbf{x}^*)$ of randomly selected training point $\mathbf{x}^*$; $z = \mathbf{a}^T\mathbf{x} + b$.

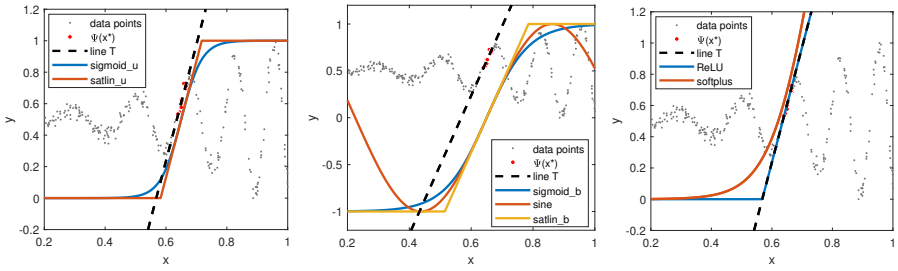

Fig. 1. AFs of different types introduced into the input space in $\mathbf{x}^*$ by D-DM.

4 Simulation Study

In this section, we report the experimental results over several regression problems in order to compare the fitting properties of D-DM with different AFs, . They include an approximation of extremely nonlinear TFs:

TF1 $g(x) = \sin(20 \cdot \exp x) \cdot x^2$, $x \in [0, 1]$
TF2 $g(x) = 0.2e^{-(10x-4)^2} + 0.5e^{-(80x-40)^2} + 0.3e^{-(80x-20)^2}$, $x \in [0, 1]$.
TF3 $g(\mathbf{x}) = \sum_{j=1}^{n} \sin(20 \cdot \exp x_j) \cdot x_j^2$, $x_i \in [0, 1]$
TF4 $g(\mathbf{x}) = -\sum_{i=1}^{n} \sin(x_i) \sin^{20}\left(\frac{ix_i^2}{\pi}\right)$, $x_i \in [0, \pi]$
TF5 $g(\mathbf{x}) = 418.9829n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|})$, $x_i \in [-500, 500]$

Both the training and test sets for TF1 and TF2 included 5000 points. For the training set, argument x was generated randomly from $U(0, 1)$, and for the

test set, it was evenly distributed in $[0, 1]$. The function values were normalized in the range $[0, 1]$. Note that TF1 starts flat, near $x = 0$, then has increasing fluctuations (see Fig. 3). TF2 has two spikes that could be difficult to model with FNN (see Fig. 5).

TF3-TF5 are multivariate functions. We considered these functions with $n = 2, 5$ and 10 arguments. The sizes of the training and test sets depended on the number of arguments. They were 5000 for $n = 2$, 20,000 for $n = 5$, and 50,000 for $n = 10$. All arguments for TF3-TF5 were normalized to $[0, 1]$, and the function values were normalized to $[-1, 1]$. Two-argument functions TF3-TF5 are shown in Fig. 2. Note that TF3 is a multivariate variant of TF1. It combines flat regions with strongly fluctuated regions. TF4 expresses flat regions with perpendicular grooves. TF5 fluctuates strongly, showing the greatest amplitude at the borders.

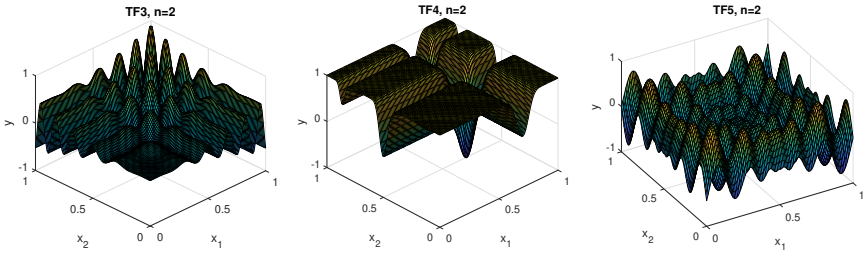

Fig. 2. Target functions TF3-TF5.

Fig. 3 shows the results of TF1 fitting. The fitted lines are composed of AFs of different shapes. The AFs distributed by D-DM in the input interval (shown by the gray field) are shown in the lower panels. FNN included 30 hidden nodes. The neighborhood size was 2 ($k = 1$). As you can see from Fig. 3, the slopes of the AFs reflect the TF slopes. D-DM introduces the steepest fragments of the AFs into the input interval. These fragments are the most useful for modeling the TF fluctuations. The saturated AF fragments in the input interval are avoided. The best fitting results were achieved for both sigmoid AFs. SINE cannot cope with a TF with variable intensity of fluctuations. Neither RELU, which yielded the highest fitting error, nor the saturating linear functions are not able to fit smoothly to TF1. The smooth counterpart of RELU, SOFTPLUS, improves significantly on the RELU fitting results by offering a smooth approximation of TF1. Obviously, the results are dependent on the number of hidden nodes. The left panel of Fig. 4 shows the TF1 fitting error for different numbers of hidden nodes. As can be seen from this figure, the sigmoid AFs outperformed all the others. Slightly worse results were achieved for SOFTPLUS, while the highest error was observed for RELU. Detailed results for each AF, i.e. RMSE for the maximal number of hidden nodes shown in the figures, are presented in Table 2. The lowest errors, i.e. those that are at least 5% lower than the others, are marked in bold in this table.

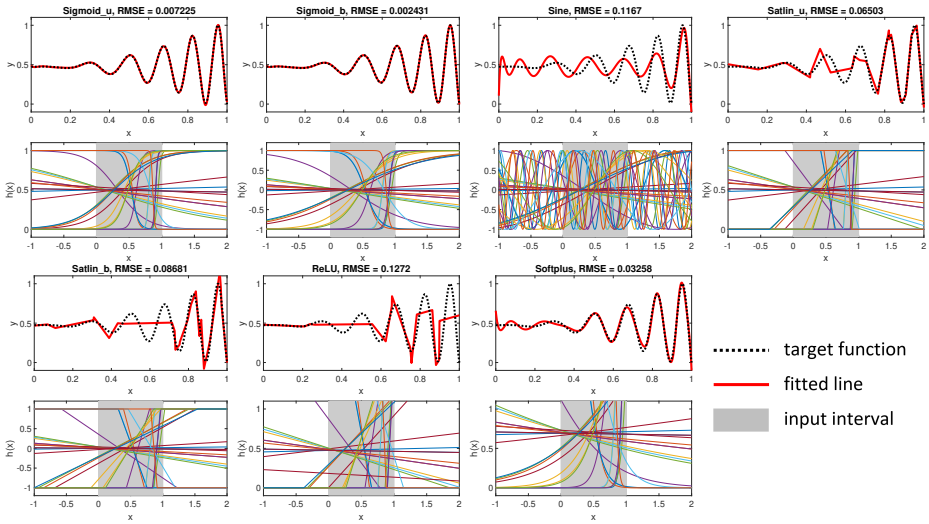

Fig. 3. TF1: Results of D-DM fitting for different AFs.

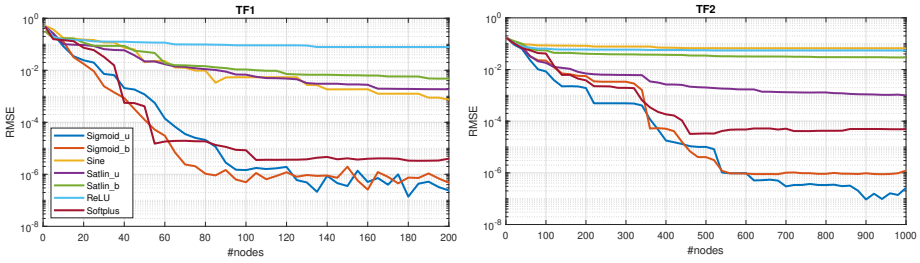

Fig. 4. Convergence of FNN for TF1 and TF2.

Table 2. Fitting errors (RMSE).

	SIGMOID_U	SIGMOID_B	SINE	SATLIN_U	SATLIN_B	RELU	SOFTPLUS
TF1	**2.39E-7**	4.74E-7	7.44E-4	1.86E-3	4.78E-3	7.84E-2	4.00E-6
TF2	**2.63E-7**	1.23E-6	6.65E-2	9.93E-4	2.93E-2	5.46E-2	4.89E-5
TF3 $n = 2$	2.19E-5	**2.26E-6**	1.64E-3	5.81E-3	9.01E-3	1.87E-2	-
TF3 $n = 5$	0.2214	0.2215	0.2213	0.2214	0.2215	0.2212	-
TF3 $n = 10$	0.2329	0.2328	0.2331	0.2329	0.2328	0.2329	-
TF4 $n = 2$	**6.69E-7**	4.87E-6	3.95E-2	2.65E-3	9.05E-3	5.18E-2	-
TF4 $n = 5$	0.2419	0.2412	0.2411	0.2381	0.2433	0.2418	-
TF4 $n = 10$	0.2611	0.2723	0.3095	0.2618	0.2738	0.2571	-
TF5 $n = 2$	**0.0083**	0.0116	0.0426	0.0257	0.0258	0.0319	-
TF5 $n = 5$	0.2385	0.2380	0.2404	0.2390	0.2381	0.2405	-
TF5 $n = 10$	0.2246	0.2243	0.2260	0.2247	0.2243	0.2238	-

Fig. 5 shows fitting results for TF2 (120 hidden nodes and $k = 1$). In this case, SIGMOID_U and SIGMOID_B provided the best fitting, while SATLIN_U and SOFTPLUS provided a slightly worse fitting. Other AFs could not cope with this TF. For them, increasing the number of hidden nodes did not improve results and RMSE remained outside the acceptable level of 0.01 (see right panel of Fig. 4 and Table 2).

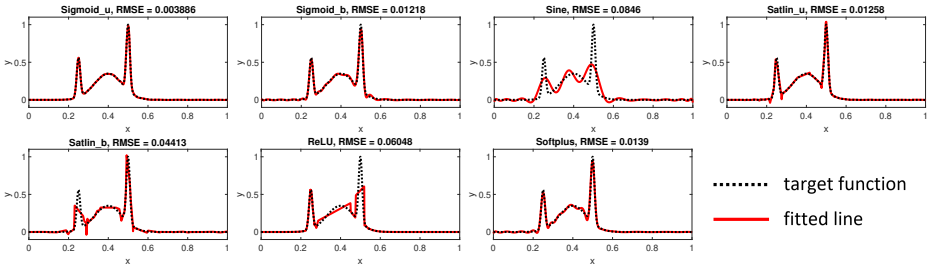

Fig. 5. TF2: Results of D-DM fitting for different AFs.

Fig. 6 shows the convergence curves of FNN trained using D-DM for two-argument TF3-TF5 ($k = n$). In all these cases, the sigmoid AFs yielded the best results, while RELU, SINE and both saturating linear functions yielded the worst results. SOFTPLUS suffered from numerical problems related to the rapid growth of this function and exceeding the limit for double precision numbers. So, in Table 2, no results for SOFTPLUS are given.

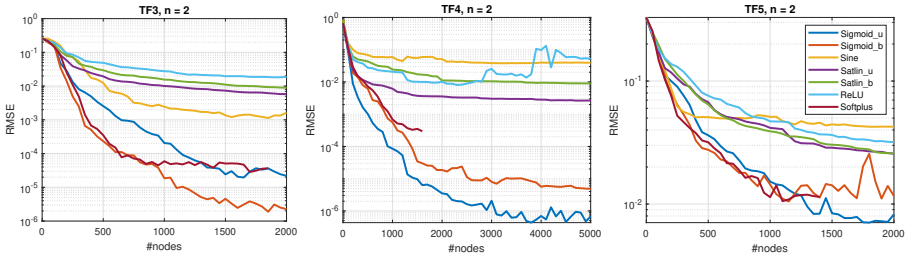

Fig. 6. Convergence of FNN for TF3-TF5, $n = 2$.

In the case of multidimensional modeling ($n = 5$ and 10), results for all AFs were comparable (see Figs. 7 and 8; $k = n$). This could be explained by the change in the TF landscape, which flattens with an increasing number of dimensions. When modeling flat TF, the AF shape turned out not to be as important as in the case of TF with strong fluctuations.

It is obvious from the performed simulations that the approximation properties of FNN trained using D-DM strongly depend on the AF type. The most

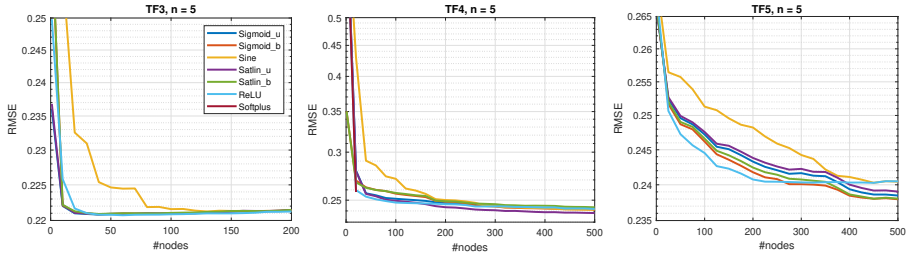

Fig. 7. Convergence of FNN for TF3-TF5, $n = 5$.

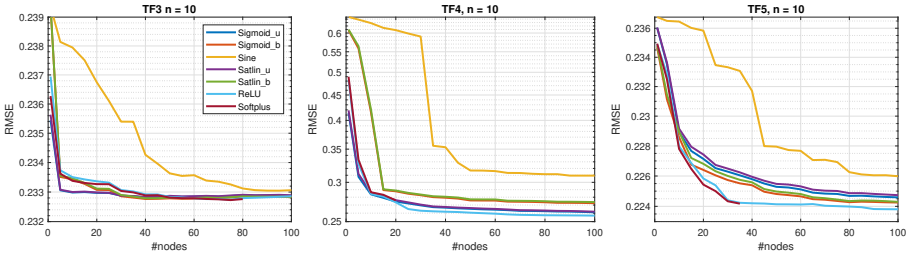

Fig. 8. Convergence of FNN for TF3-TF5, $n = 10$.

useful for smoothing highly nonlinear TFs with fluctuations turned out to be the sigmoid AFs. The piecewise linear functions, i.e. RELU, SATLIN_U, and SATLIN_B, have problems with modeling smoothly complex TFs. Their linear parts do not fit accurately to TF nonlinearities. Likewise SINE AFs cannot build an acceptable fitted function for the fluctuated TFs. The reason for this is probably the periodic nature of SINE. When SINE AF is introduced into the input space to improve the fitted function in region $\Psi(\mathbf{x}^*)$, it can worsen the fitted function in other regions by introducing unwanted fluctuations. SOFTPLUS AF gave slightly worse results than sigmoid AFs for one-argument TFs, but it caused numerical problems for multivariate TFs.

5 Conclusion

The data-driven FNN learning described in this study is an alternative to both standard gradient-based learning and randomized learning. It allows us to bypass the tedious iterative process of tuning weights based on gradients. In the proposed approach, the parameters of hidden nodes are calculated based on the local properties of the TF. The AFs, which compose the fitted function, are introduced into the input space in randomly selected regions and their slopes are adjusted to the TF slopes in these regions. Consequently, the set of AFs reflects the TF fluctuations in different regions, which leads to accurate approximation. Our approach is completely different from typical randomized learning, where the AF parameters are chosen randomly and do not reflect the TF landscape.

D-DM finds the network parameters quickly, without repeatedly presenting the training set.

FNN performance strongly depends on AF shape. In this work, using a data-driven approach, we derived equations for the hidden node parameters for different AFs. As our experimental study has shown, the best FNN performance in smoothing highly nonlinear TFs was achieved by the sigmoid AFs. They were able to fit to the TF fluctuations. RELU AF, which is very popular in deep learning, fared very poorly in fluctuation modeling due to its piecewise linear nature. Its smooth counterpart, SOFTPLUS, produced much better results but suffered from numerical problems related to rapid growth.

References

1. Principe, J., Chen, B.: Universal approximation with convex optimization: Gimmick or reality? IEEE Computational Intelligence Magazine **10**(2), 68–77 (2015)
2. Husmeier, D.: Random vector functional link (RVFL) networks. In: Neural Networks for Conditional Probability Estimation: Forecasting Beyond Point Predictions, chap. 6, pp. 87–97. Springer-Verlag, London (1999)
3. Cao, W., Wang, X., Ming, Z., Gao, J.: A review on neural networks with random weights. Neurocomputing **275**, 278-–287 (2018)
4. Zhang, L., Suganthan, P.: A survey of randomized algorithms for training neural networks. Inf. Sci. **364–365**, 146-–155 (2016)
5. Dudek, G.: Generating random weights and biases in feedforward neural networks with random hidden nodes. Information Sciences, **481**, 33–56 (2019)
6. Dudek, G.: Generating random parameters in feedforward neural networks with random hidden nodes: Drawbacks of the standard method and how to improve it. In: Yang, H., Pasupa, K., Leung, A.CS., Kwok, J.T., Chan, J.H., King, I. (eds) Neural Information Processing. ICONIP 2020. Communications in Computer and Information Science, vol. 1333, pp. 598-606, Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63823-8_68
7. Dudek, G.:Data-driven randomized learning of feedforward neural networks, 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, pp. 1–8 (2020). https://doi.org/10.1109/IJCNN48605.2020.9207353
8. Igelnik, B., Pao, Y.H.: Stochastic choice of basis functions in adaptive function approximation and the functional-link net, IEEE Trans. Neural Netw. **6**(6), 1320—1329 (1995)