Sensitivity Analysis of the Neural Networks Randomized Learning

Grzegorz Dudek(✉)

Electrical Engineering Faculty, Częstochowa University of Technology, Częstochowa, Poland
dudek@el.pcz.czest.pl

Abstract. Randomized algorithms for learning feedforward neural networks are increasingly used in practice. They offer very speed training because the only parameters that are learned are the output weights. Parameters of hidden neurons are generated randomly once and need not to be adjusted. The key issue in randomized learning algorithms is to generate parameters in a right way to ensure good approximation and generalization properties of the network. Recently the method of generating hidden nodes parameters was proposed [1], which ensures better adjustment of the random parameters to the target function and better distribution of neurons in the input space, when comparing to the previous approaches. In this work the new method is tested in terms of sensitivity to the number of neurons, noise in data and data deficit. Experiments shows better results for the new method in comparison to the existing approach of generating random parameters of the network.

Keywords: Randomized learning algorithms · Neural networks with random hidden nodes · Feedforward neural networks

1 Introduction

In conventional learning of neural networks (NNs) all parameters, weights and biases, are required freely adjustable. They are tuned properly during a learning process which usually employs some form of gradient descent method which is known to be time consuming, sensitive to initial values of parameters and converging to local minima. For complex classification or regression problems the training is complicated and inefficient. In recent years randomized learning algorithms for NNs are developed by many researchers. The original idea of building NNs with random weights can be found in [2] and [3]. In these approaches the weights and biases of hidden nodes are assigned with random values and need not to be adjusted during the learning process. Thus, the resulting optimization task solved by NN becomes convex and can be formulated as a linear least-squares

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

L. Rutkowski et al. (Eds.): ICAISC 2019, LNAI 11508, pp. 51–61, 2019.
https://doi.org/10.1007/978-3-030-20912-4_5

problem [4]. This results in a thousandfold increase in the learning speed over the classical gradient descent-based learning. Many simulation studies reported in the literature show high performance of the randomized neural networks (RNNs) which is compared to fully adaptable ones.

Parameters of the RNN hidden neurons are randomly selected from some intervals according to any continuous sampling distribution and do not change. However, how to select these intervals remains an open question. This issue is considered to be one of the most important research gaps in the field of randomized NN learning [5]. In applications of RNNs reported in literature the ranges for random parameters are selected without scientific justification and could not ensure the universal approximation property. Usually these intervals are assigned as $[-1, 1]$, regardless of the data distribution, complexity of the target function and type of an activation function. Some authors note the influence of these intervals on the model performance and suggest to optimize them in a more appropriate range for a specified application [6,7]. For example in [8] 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. Then, the biases are computed to center each sigmoid at one of the training points. In [9] authors combine unsupervised placement of network nodes according to the input data density with subsequent supervised or reinforcement learning values of the linear parameters of the network. In [10] a supervisory mechanism of assigning random weights and biases is proposed for the model generated incrementally by stochastic configuration algorithms. The random parameters are generated adaptively selecting the scope for them, ensuring the universal approximation property of the network.

In this work a new method of generating random NN parameters proposed recently in [1] is investigated. The method generates weights and biases separately depending on the data scope and complexity, and activation function type. It ensures an adjustment of the random parameters to the target function and better distribution of neurons in the input space when comparing to the previous approaches with fixed intervals for random parameters. We test the new method in terms of sensitivity to the number of neurons, noise in data and training data deficit.

The rest of this paper is structured as follows. Section 2 introduces randomized learning algorithms in two versions: a classical one with fixed intervals for random parameters and in the new one proposed in [1]. Section 3 reports experimental results concerning sensitivity analysis for both versions of the randomized learning. Conclusions are given in Sect. 4.

2 Randomized Learning Algorithms

In this work feedforward neural networks (FNNs) with a single hidden layer are considered. The network has n inputs, one output and m hidden nodes with activation functions $h(x)$. The training set is $\Phi = \{(\mathbf{x}_l, y_l)|\mathbf{x}_l \in \mathbb{R}^n, y_l \in \mathbb{R}, l = 1, 2, ..., N\}$.

In the first step of learning the parameters of each hidden node are generated by random: weights $\mathbf{a}_i = [a_{i,1}, a_{i,2}, \ldots, a_{i,n}]^T$ 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 $b_i \sim U(b_{min}, b_{max})$.

In the second step the output matrix for the hidden layer is calculated:

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

where $h_i(x)$ is an activation function of the i-th node, which is nonlinear piecewise continuous function. In this work a sigmoid activation function is used:

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

The i-th column of $\mathbf{H}$ is the i-th hidden node output vector with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$. Note that hidden nodes map the data from n-dimensional input space to m-dimensional feature space, and $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_m(\mathbf{x})]$ is a nonlinear feature mapping. Because the parameters $\mathbf{a}_i$ and b_i are fixed, the output matrix $\mathbf{H}$ is calculated only once and remains unchanged.

The output weights connecting hidden nodes with output node can be obtained by solving the following linear equation system:

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

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 (3) can be expressed by $\boldsymbol{\beta} = \mathbf{H}^+\mathbf{Y}$, where $\mathbf{H}^+$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$.

The network expresses a linear combination of the activation functions $h_i(\mathbf{x})$ of the form:

$$\varphi(\mathbf{x}) = \sum_{i=1}^{m} \beta_i h_i(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} \tag{4}$$

It is worth mentioning that the prototype of NN with randomization was Random Vector Functional Link (RVFL) network proposed by Pao and Takefji [3]. This solution has also direct links from the input layer to the output one. In experimental part of this work we use RVFL as a comparative model.

In most of the works on randomized learning algorithms the intervals for random parameters of hidden nodes are assigned as fixed regardless of the data distribution and activation function type. Typically $a_{min} = b_{min} = -1$ and $a_{max} = b_{max} = 1$. In [1] it was demonstrated that the intervals of the random weights and biases are extremely important due to approximation properties of the network. When they are set as $[-1, 1]$ the neurons operate on the saturation fragments of activation functions and accurate fitting to the strongly nonlinear

function can be impossible. The method proposed in [1] distributes neurons across the input space and adjusts the activation function slopes to the target function steepness. According to this approach the weights of the i-th hidden node are calculated as follows:

$$a_{i,k} = \zeta_k \frac{\Sigma_i}{\sum_{j=1}^{n} \zeta_j} \tag{5}$$

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: $\Sigma_i = a_{i,1} + a_{i,2} + ... + a_{i,n}$, which is randomly selected from the interval:

$$|\Sigma_i| \in \left[\ln\left(\frac{1-r}{r}\right), s \cdot \ln\left(\frac{1-r}{r}\right) \right] \tag{6}$$

Two parameters in (6), $r \in (0, 0.5)$ and $s > 1$, control the steepness of activation functions. Specifically, they determine two boundary sigmoids between which the activation functions are randomly generated.

Having weights $a_{i,k}$, the bias for the i-th activation function is determined in such a way that the inflection point of the sigmoid is set at some point $\mathbf{x}_*$ randomly generated inside the input space. When the input vectors $\mathbf{x}$ are normalized so that they belong to the n-dimensional unit hypercube $H = [0, 1]^n \subset \mathbb{R}^n$, the point $\mathbf{x}_*$ is selected from H, thus $x_{*,1}, x_{*,2}, \ldots, x_{*,n} \sim U(0, 1)$. The bias for the i-th activation function is calculated from:

$$b_i = -\mathbf{a}_i^T \mathbf{x}_* \tag{7}$$

From (7) we can see that the bias of the i-th hidden node is strictly dependent on the weights of this node. When generating random parameters of the hidden nodes, the weights and biases should be considered separately, because these parameters have different meaning. Thus generating them both from the same interval, usually $[-1, 1]$, is incorrect. More detailed discussion on this topic and derivations of the above equations can be found in [1].

In the next section we compare the new method of generating random parameters with the method based on the fixed intervals of $[-1, 1]$ including RVFL where additional direct connections between input and output layers are introduced.

3 Simulation Study

This section reports some simulation results over the regression problem including a two-variable function approximation task. A target function is defined as follows:

$$g(\mathbf{x}) = \sin\left(20 \cdot \exp\left(x_1\right)\right) \cdot x_1^2 + \sin\left(20 \cdot \exp\left(x_2\right)\right) \cdot x_2^2 \tag{8}$$

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

This function is shown in Fig. 1. Note that a variation of function (8) is the lowest around the corner $[0, 0]$ and gradually increases towards the corner $[1, 1]$.

The training set Φ contains 5000 points $(\mathbf{x}_l, y_l)$. The 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 size 100000 points is distributed uniformly in the input space and is not disturbed by noise. It expresses the true target function, which is spanned between -1.64 and 1.78.

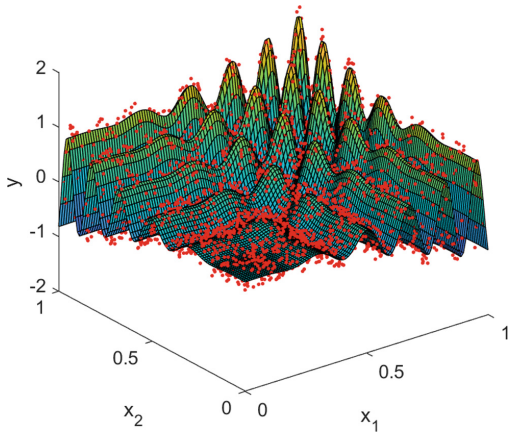

Fig. 1. The target function and training points.

We test three randomized approaches for FNN learning described in the previous section:

- RNN1: FNN with random parameters generated according to (5) and (7),
- RNN2: FNN with random parameters generated from the uniform distribution over $[-1, 1]$,
- RNN3: RVFL network with random parameters generated as in RNN2 from $[-1, 1]$.

In all cases the sigmoidal activation function is used in the hidden nodes. As a measure of accuracy in the comparative studies we use root mean squares error (RMSE). For each experiment 100 independent trials are performed. The r and s parameters for RNN1 were adopted from [1] as 0.1 and 5, respectively. With such values good results were obtained for function (8) approximation.

In the first experiment the impact of the number of hidden nodes on the approximation accuracy of the NNs is investigated. The number of hidden nodes is changed from 100 to 2000 with step of 100. Figure 2 shows the RMSE distributions using box-and-whisker plots for the investigated randomized learning methods. As we can see from this figure, the training error for RNN1 converges to the value of around 0.10. The test error for RNN1 has a minimum ($RMSE = 0.0645$) for 600 nodes. Adding hidden nodes over 600 increases both

RMSE and its variance. This exhibits an overtraining: too many steep nodes fit into noisy data points. RMSE for RNN2 and RNN3, where random parameters are chosen from $[-1, 1]$, is incomparably greater than for RNN1. It is due to using saturated parts of activation functions to compose strongly nonlinear target function. Adding new neurons does not improve the results. The pattern of the training error distribution for different numbers of nodes is very similar to the pattern of the test error distribution. In RNN2 and RNN3 cases, networks are not prone to overfitting with an increase in the number of neurons. They are strongly underfitted. This is exemplified in Fig. 3 (upper charts), where the fitted surfaces for 2000 nodes are shown. For comparison, bottom charts show the fitting surfaces constructed by RNN1 with 500 (good fitting) and 2000 (overfitting) nodes. In Table 1 the errors are shown for the optimal number of hidden nodes. Note that RMSE for test data are above six times lower for RNN1 than for RNN2 and RNN3.

Table 1. Errors for optimal number of neurons.

Approach	#neurons	$RMSE_{trn}$	$RMSE_{tst}$
RNN1	600	0.1119 ± 0.0012	0.0645 ± 0.0393
RNN2	2000	0.4307 ± 0.0001	0.4196 ± 0.0001
RNN3	2000	0.4304 ± 0.0001	0.4193 ± 0.0001

In the second experiment we test how the results are sensitive to the noise disturbing data. The training data are generated from (8) and are distorted by adding the uniform noise distributed in $[-c, c]$. The noise boundary c changes from 0 to 1 with step of 0.1. It means that the noise level defined as the ratio of the noise range to the target function range (which is 3.42) is from 0 to about 58%. For each noise level 100 independent trials are performed for each randomized NN. The number of hidden nodes was set to 500. Results in Fig. 4 are shown. The training and test errors for RNN1 gradually increase with the noise level. The increase is faster for the training error. This is because the test points expressing the true target function are not disturbed by noise, i.e. they are the same for each noise level in training points. The relationship between the percentage increase in the training error and the percentage noise level can be estimated by the linear regression: $\Delta RMSE_{\%} = 32.38 c_{\%} - 35.70$. For test data this equation is of the form: $\Delta RMSE_{\%} = 5.43 c_{\%} - 81.77$. In the case of RNN2 and RNN3, where the flat parts of the activation functions are mostly used by neurons, the fitted surfaces are similar to each other for different noise level in the training data. The training error increases with the noise level because the training points move away from the fitted surface. In the same time, the test error stays at the same level because neither test points nor fitted surfaces change with the noise. But due to modeling using saturated parts of neurons the test error in RNN2 and RNN3 is much bigger than in the case of RNN1.

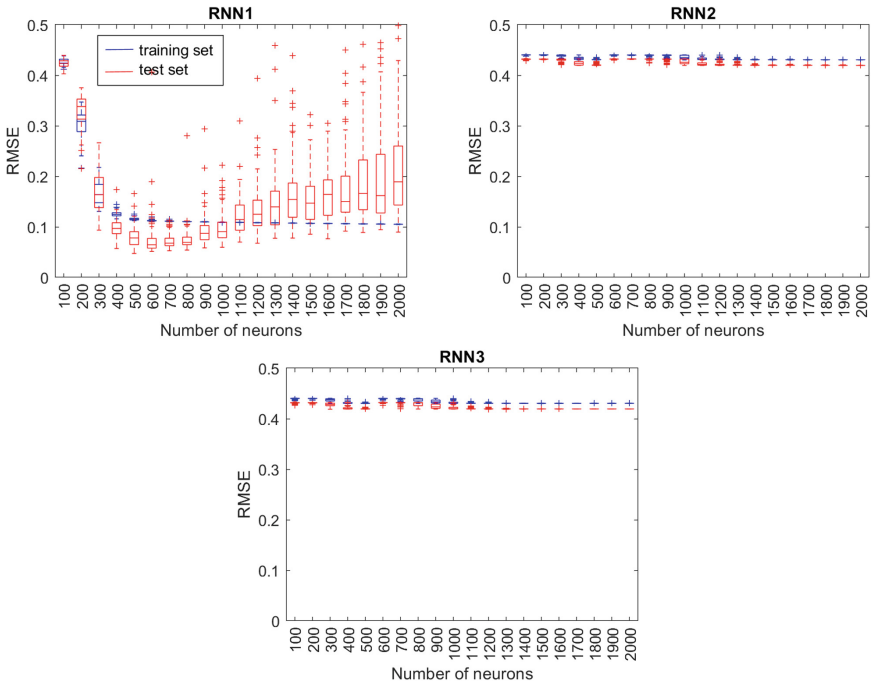

Fig. 2. Impact of the hidden neurons number on the error.

Table 2. Errors for the noise level $c = 1$

Approach	$RMSE_{trn}$	$RMSE_{tst}$
RNN1	0.5570 ± 0.0011	0.2222 ± 0.0747
RNN2	0.6985 ± 0.0011	0.4290 ± 0.0021
RNN3	0.6983 ± 0.0003	0.4282 ± 0.0003

In Table 2 the errors are shown for the noise level $c = 1$ corresponding to the maximum considered disruption of data at level of 58%. In this case the test RMSE for RNN1 increased to 0.2222 from 0.0572 for data without noise. For RNN2 and RNN3 the test RMSE at the maximum noise level was about twice higher than for RNN1.

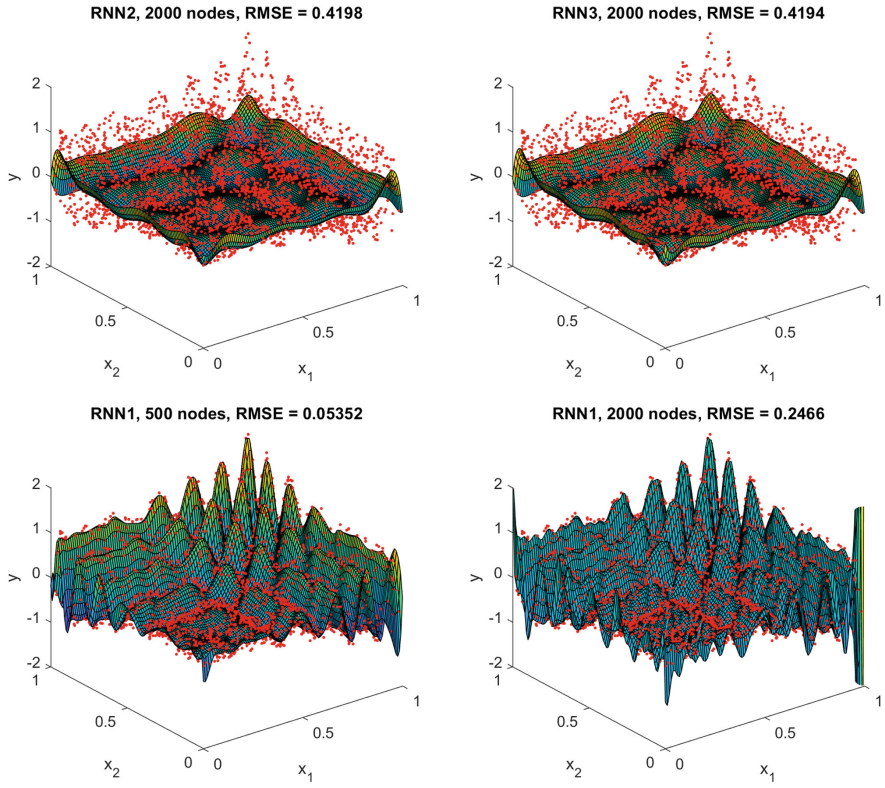

Fig. 3. The surfaces fitted to the training points.

In the third experiment we investigate the influence of the number of training points on accuracy of the randomized NNs. Setting the number of hidden nodes as 500 we change the number of training points from 500 to 5000 with the step of 500. Figure 5 shows the results. For a smaller number of training points the lower training errors are observed. Our 500 steep neurons in RNN1 are able to fit better into a small number of points. But this small set of training points does not reflect sufficiently the target function complexity. Deficit in training points and flexible learning model lead to overfitting (see Fig. 6). This is a cause of bigger test errors for smaller number of training points. Bigger training sets lead to improvement in accuracy on the test set. For RNN2 and RNN3 the fitted surface is not able to fit accurately to the training points (see Fig. 6), and the training error only slightly improves with the number of training points. Due to a poor fitting of the model to the training points, the test error is less sensitive to the training points number when compared to RNN1 case. But in the RNN2 and RNN3 cases the error level is unacceptable high.

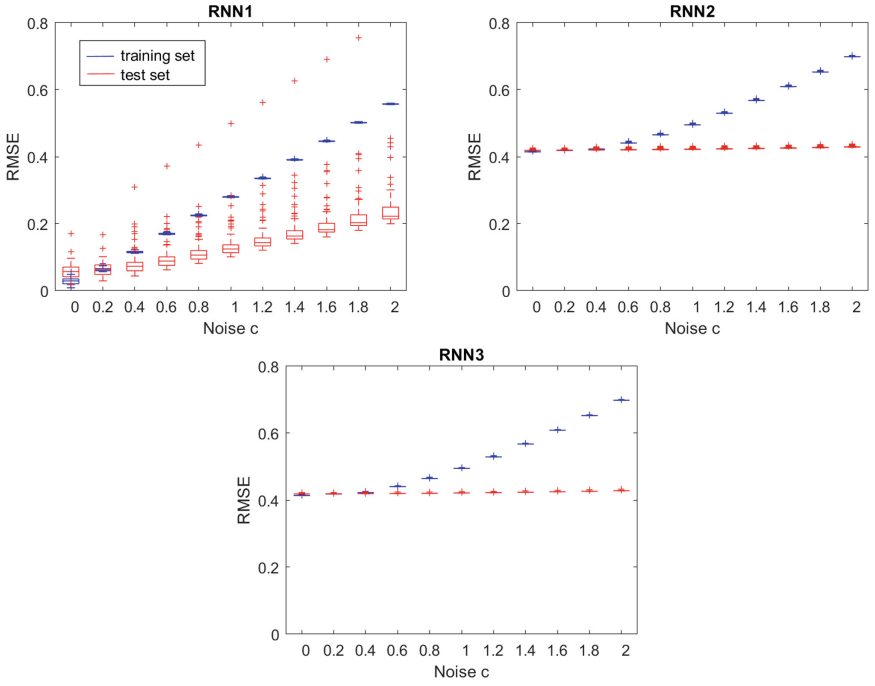

Fig. 4. Impact of the noise level on the error.

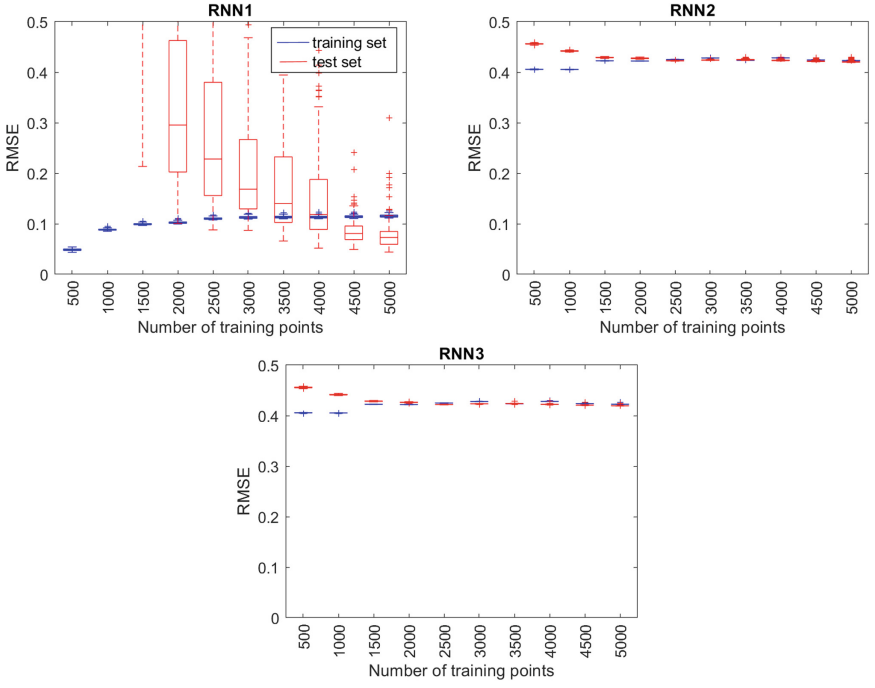

Fig. 5. Impact of the training points number on the error.

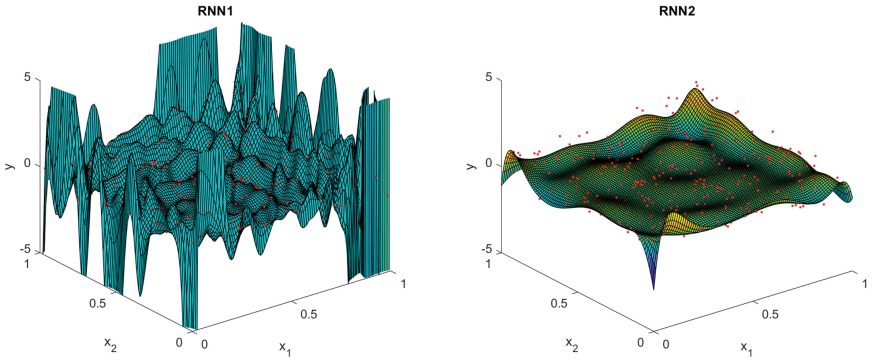

Fig. 6. Fitted surfaces for 500 training points.

4 Conclusion

The way of generating random parameters of the randomized neural networks is extremely important. Typically the random weights and biases are chosen from the fixed interval of $[-1, 1]$. In such case, the activation functions of hidden nodes, which are used for construction the surface fitting data, are usually incorrectly distributed in the input space having their saturated parts in it. Thus they cannot approximate a highly nonlinear target function with required accuracy. This was confirmed in the experimental part of the work. Adding more flat neurons to the network does not improve significantly the results. A randomized network with flat neurons seems to be resistant to noise in the training data and to training data deficit. But this cannot be taken seriously because it results from a weak approximation capacity for complex functions. In contrast to the typical approach of generating random parameters from the fixed interval, the method where these parameters are generated in such a way that the slopes of the activation functions are matched to the steepness of the target function and the neurons are distributed across the input space according to the data arrangement, brings more accurate results. The performance of the network depends on the parameters controlling the slope of the activation functions (r and s) and the number of hidden nodes. Too many steep hidden nodes leads to overfitting which deteriorate generalization properties of the network. So the slope of neurons as well as the neuron number should be adjusted to the target function taking into account the noise level. When data includes high level of noise, the training points move away from the target function and therefore its features are invisible for the network. Also training data deficit makes the target function blurry. In this case the error between rare training points increases, especially when the activation functions are too steep. The solution to these problems is the local fitting of neurons to the target function reflecting its local features. This will be the subject of the future research.

References

1. Dudek, G.: Generating random weights and biases in feedforward neural networks with random hidden nodes. Inf. Sci. **481**, 33–56 (2019)
2. Schmidt, W.F., Kraaijveld, M.A., Duin, R.P.W.: Feedforward neural networks with random weights. In: Proceedings of the 11th IAPR International Conference Pattern Recognition Methodology and Systems, vol. II, pp. 1–4 (1992)
3. Pao, Y.H., Takefji, Y.: Functional-link net computing: theory, system architecture, and functionalities. IEEE Comput. **25**(5), 76–79 (1992)
4. Principe, J., Chen, B.: Universal approximation with convex optimization: gimmick or reality? IEEE Comput. Intell. Mag. **10**, 68–77 (2015)
5. Zhang, L., Suganthan, P.N.: A survey of randomized algorithms for training neural networks. Inf. Sci. **364–365**, 146–155 (2016)
6. Husmeier, D.: Random Vector Functional Link (RVFL) networks. In: Neural Networks for Conditional Probability Estimation: Forecasting Beyond Point Predictions. Perspectives in Neural Computing, Chap. 6, pp. 87–97. Springer, London (1999). https://doi.org/10.1007/978-1-4471-0847-4_6
7. Li, M., Wang, D.: Insights into randomized algorithms for neural networks: practical issues and common pitfalls. Inf. Sci. **382–383**, 170–178 (2017)
8. Ferrari, S., Stengel, R.F.: Smooth function approximation using neural networks. IEEE Trans. Neural Networks **16**(1), 24–38 (2005)
9. Gorban, A.N., Tyukin, I.Y., Prokhorov, D.V., Sofeikov, K.I.: Approximation with random bases: Pro- et Contra. Inf. Sci. **364**, 129–145 (2016)
10. Wang, D., Li, M.: Stochastic configuration networks: fundamentals and algorithms. IEEE Trans. Cybern. **47**(10), 3466–3479 (2017)