Generating Random Parameters in Feedforward Neural Networks with Random Hidden Nodes: Drawbacks of the Standard Method and How to Improve It*

Grzegorz Dudek[0000−0002−2285−0327]

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

Abstract. The standard method of generating random weights and biases in feedforward neural networks with random hidden nodes selects them both from the uniform distribution over the same fixed interval. In this work, we show the drawbacks of this approach and propose new methods of generating random parameters. These methods ensure the most nonlinear fragments of sigmoids, which are most useful in modeling target function nonlinearity, are kept in the input hypercube. A new method generating sigmoids with uniformly distributed slope angles demonstrated the best performance on the illustrative examples.

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

1 Introduction

Single-hidden-layer feedforward neural networks with random hidden nodes (FNNRHN) have become popular in recent years due to their fast learning speed, good generalization performance and ease of implementation. Additionally, these networks do not use a gradient descent method for learning, which is time consuming and sensitive to local minima of the error function (which is nonconvex in this case). In randomized learning, weights and biases of the hidden nodes are selected at random from any interval $[-u, u]$, and stay fixed. The optimization problem becomes convex and the output weights can be learned using a simple, scalable standard linear least-squares method [7]. The resulting FNN has a universal approximation capability when the random parameters are selected from a symmetric interval according to any continuous sampling distribution [5]. But how to select this interval and which distribution to use are open questions, and considered to be the most important research gaps in randomized learning [11,2].

Typically, the hidden node weights and biases are both selected from a uniform distribution over the fixed interval, $[-1, 1]$, without scientific justification,

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

regardless of the data, problem to be solved, and activation function type [8]. Some authors optimize the interval looking for u to ensure the best model performance [9,6,10,1]. Recently developed methods [3,4] propose more sophisticated approaches for generating random parameters, where the distribution of the activation functions in space is analyzed and their parameters are adjusted randomly to the data.

In this work we show the drawbacks of a standard method of random parameters generation and propose its modification. We treat the weights and biases separately due to their different functions. The biases are generated on the basis of the weights and points selected from the input space. The resulting sigmoids have their nonlinear fragments, which are most useful for modeling the target function (TF) fluctuations, inside the input hypercube. Moreover, we show how to generate the weights to produce sigmoids with the slope angles distributed uniformly.

2 Generating sigmoids inside the input hypercube

Let us consider an approximation problem of a single-variable TF of the form:

$$g(x) = \sin(20 \cdot \exp x) \cdot x^2 \tag{1}$$

To learn FNNRHN we create a training set Φ containing $N = 5000$ points (x_l, y_l), where $x_l \sim U(0, 1)$ and y_l are calculated from (1) and then distorted by adding noise $\xi \sim U(-0.2, 0.2)$. A test set of the same size is created in the same manner but without noise. The output is normalized in the range $[-1, 1]$.

Fig. 1 shows the results of fitting when using FNNRHN with 100 sigmoid hidden nodes which weights and biases are selected from $U(-1, 1)$ and $U(-10, 10)$. The bottom charts show the hidden node sigmoids whose linear combination forms the function fitting data. This fitted function is shown as a solid line in the upper charts. As you can see from the figure, for $a, b \in [-1, 1]$ the sigmoids are flat and their distribution in the input interval $[0, 1]$ (shown as a grey field) does not correspond to the TF fluctuations. This results in a very weak fit. When $a, b \in [-10, 10]$, the sigmoids are steeper but many of them have their steepest fragments, which are around their inflection points, outside of the input interval. The saturated fragments of these sigmoids, which are in the input interval, are useless for modeling nonlinear TFs. So, many of the 100 sigmoids are wasted. From this simple example it can be concluded that to get a parsimonious flexible FNNRHN model, the sigmoids should be steep enough and their steepest fragments, around the inflection points, should be inside the input interval.

Let us analyze how the inflection points are distributed in space when the weights and biases are selected from a uniform distribution over the interval $[-u, u]$. The sigmoid value at its inflection point χ is 0.5, thus:

$$\frac{1}{1 + \exp(-(a \cdot \chi + b))} = 0.5 \tag{2}$$

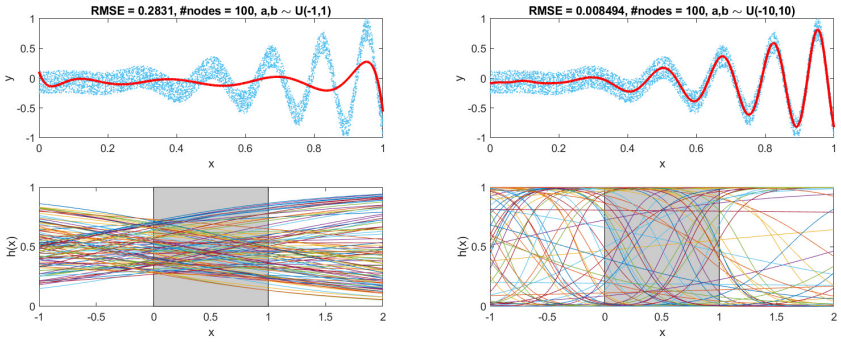

Fig. 1. TF (1) fitting: fitted curves and the sigmoids constructing them for $a, b \sim U(-1, 1)$ (left panel) and for $a, b \sim U(-10, 10)$ (right panel).

From this equation we obtain:

$$\chi = -a/b \tag{3}$$

The distribution of the inflection point is a distribution of the ratio of two independent random variables having both the uniform distribution, $a, b \sim U(-u, u)$. In such a case, the probability density function (PDF) of χ is:

$$f(\chi) = \int_{-\infty}^{\infty} |a| f_A(a) f_B(a\chi) da = \begin{cases} \int_{-u}^{u} |a| f_A(a) f_B(a\chi) da & \text{for } |\chi| < 1 \\ \int_{-\frac{u}{|\chi|}}^{\frac{u}{|\chi|}} |a| f_A(a) f_B(a\chi) da & \text{for } |\chi| \geq 1 \end{cases} \tag{4}$$

$$= \begin{cases} \dfrac{1}{4} & \text{for } |\chi| < 1 \\ \dfrac{1}{4|\chi|^2} & \text{for } |\chi| \geq 1 \end{cases}$$

where f_A and f_B are the PDFs of weights and biases, respectively.

The left panel of Fig. 2 shows the PDF of χ. The same PDF can be obtained when $a \sim U(-u, u)$ and $b \sim U(0, u)$ (case sometimes found in the literature). As you can see from Fig. 2, the probability that the inflection point is inside the input interval (shown as a grey field) is 0.25. This means that most sigmoids have their steepest fragments, which are most useful for modeling TF fluctuations, outside of this interval. For the multivariable case, when we consider n-dimensional sigmoids, the situation improves – see the right panel of Fig. 2. For $n = 2$ almost 46% of sigmoids have their inflection points in the input rectangle. This percentage increases to more than 90% for $n \geq 7$.

To obtain an n-dimensional sigmoid with one of its inflection points χ inside the input hypercube $H = [x_{1,\min}, x_{1,\max}] \times ... \times [x_{n,\min}, x_{n,\max}]$, first, we generate

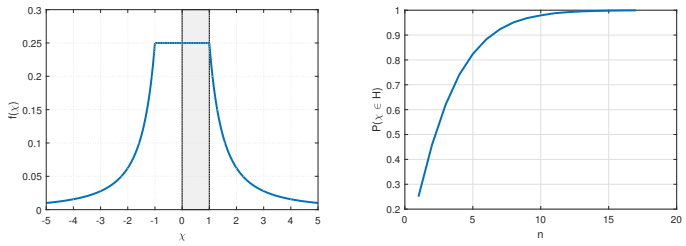

Fig. 2. PDF of χ when $a, b \sim U(-u, u)$ (left panel) and probability that χ belongs to $H = [0, 1]^n$ depending on n (right panel).

weights $\mathbf{a} = [a_1, ..., a_n]^T \subset \mathbb{R}^n$. Then we set the sigmoid in such a way that χ is at some point $\mathbf{x}^*$ from H. Thus:

$$h(\mathbf{x}^*) = \frac{1}{1 + \exp\left(-\left(\mathbf{a}^T\mathbf{x}^* + b\right)\right)} = 0.5 \tag{5}$$

From this equation we obtain:

$$b = -\mathbf{a}^T\mathbf{x}^* \tag{6}$$

Point $\mathbf{x}^* = [x_1^*, ..., x_n^*]$ can be selected as follows:

- this can be some point randomly selected from H: $x_j^* \sim U(x_{j,\min}, x_{j,\max})$, $j = 1, ..., n$. This method is suitable when the input points are evenly distributed in H.
- this can be some randomly selected training point: $\mathbf{x}^* = \mathbf{x}_\xi \in \Phi$, where $\xi \sim U\{1, ..., N\}$. This method distributes the sigmoids according to the data density, avoiding empty regions.
- this can be a prototype of the training point cluster: $\mathbf{x}^* = \mathbf{p}_i$, where $\mathbf{p}_i$ is a prototype of the i-th cluster. This method groups the training points into $m =$#nodes clusters. For each sigmoid a different prototype is taken as $\mathbf{x}^*$.

3 Generating sigmoids with uniformly distributed slope angles

It should be noted that weight a translates nonlinearly into the slope angle of a sigmoid. Let us analyze sigmoid S which has its inflection point χ in $x = 0$. In such a case $b = 0$. A derivative of S in $x = 0$ is equal to the tangent of its slope angle α in χ:

$$\tan\alpha = ah(x)\left(1 - h(x)\right) = \frac{a}{1 + \exp(-(a \cdot 0 + 0))}\left(1 - \frac{1}{1 + \exp(-(a \cdot 0 + 0))}\right) \tag{7}$$

From (7) we obtain the relationship between the weight and the slope angle:

$$\alpha = \arctan \frac{a}{4} \tag{8}$$

This relationship is depicted in Fig. 3 as well as the PDF of α when weights a are generated from different intervals. Note that the relationship between a and α is highly nonlinear. Interval $[-1, 1]$ for a corresponds to the interval $[-14°, 14°]$ for α, so only flat sigmoids are obtainable in such a case. For $a \in [-10, 10]$ we obtain $\alpha \in [-68.2°, 68.2°]$, and for $a \in [-100, 100]$ we obtain $\alpha \in [-87.7°, 87.7°]$. For narrow intervals for a, such as $[-1, 1]$, the distribution of α is similar to a uniform one. When the interval for a is extended, the shape of PDF of α changes – larger angles, near the bounds, are more probable than smaller ones. When $a \in [-100, 100]$, more than 77% of sigmoids are inclined at an angle greater than 80°, so they are very steep. In such a case, there is a real threat of overfitting.

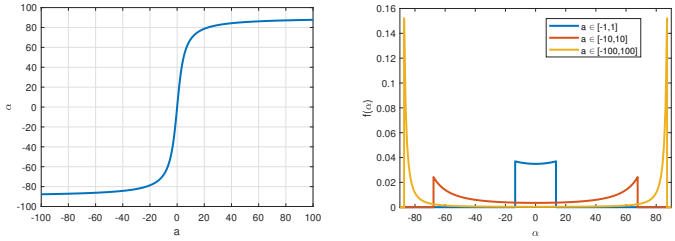

Fig. 3. Relationship between a and α (left panel) and PDF of α for different intervals for a (right panel).

To generate sigmoids with uniformly distributed slope angles, first we generate $|\alpha| \sim U(\alpha_{\min}, \alpha_{\max})$ individually for them, where $\alpha_{\min} \in (0°, 90°)$ and $\alpha_{\max} \in (\alpha_{\min}, 90°)$. The border angles, $\alpha_{\min}$ and $\alpha_{\max}$, can both be adjusted to the problem being solved. For highly nonlinear TFs, with strong fluctuations, only $\alpha_{\min}$ can be adjusted, keeping $\alpha_{\max} = 90°$. Having the angles, we calculate the weights from (8):

$$a = 4 \tan \alpha \tag{9}$$

For the multivariable case, we generate all n weights in this way, independently for each of m sigmoids. This ensures random slopes (between $\alpha_{\min}$ and $\alpha_{\max}$) for the multidimensional sigmoids in each of n directions.

The proposed methods of generating random parameters of the hidden nodes are summarized in Algorithm 1. In this algorithm weights a can be generated randomly from $U(-u, u)$ or optionally, to ensure uniform distribution of the sigmoid slope angles, they can be determined based on the slope angles generated randomly from $U(\alpha_{\min}, \alpha_{\max})$. The bounds: $u, \alpha_{\min}$ and $\alpha_{\max}$ should be selected in cross-validation.

Algorithm 1 Generating Random Parameters of FNNRHN

Input:

Number of hidden nodes m
Number of inputs n
Bounds for weights, $u \in \mathbb{R}^+$, or optionally bounds for slope angles, $\alpha_{\min} \in (0°, 90°)$ and $\alpha_{\max} \in (\alpha_{\min}, 90°)$
Set of m points $\mathbf{x}^* \in H$: $\{\mathbf{x}_1^*, ..., \mathbf{x}_m^*\}$

Output:

$$\text{Weights } \mathbf{A} = \begin{bmatrix} a_{1,1} & \dots & a_{m,1} \\ \vdots & \ddots & \vdots \\ a_{1,n} & \dots & a_{m,n} \end{bmatrix}, \qquad \text{biases } \mathbf{b} = [b_1, \dots, b_m]$$

Procedure:

for $i = 1$ **to** m **do**
  for $j = 1$ **to** n **do**
    Choose randomly $a_{i,j} \sim U(-u, u)$ or optionally
    choose randomly $\alpha_{i,j} \sim U(\alpha_{\min}, \alpha_{\max})$ and calculate

$$a_{i,j} = (-1)^q \cdot 4 \tan \alpha_{i,j}, \text{ where } q \sim U\{0, 1\}$$

  end for
  Calculate $b_i = -\mathbf{a}_i^T \mathbf{x}_i^*$
end for

4 Simulation study

The results of TF (1) fitting when using the proposed method is shown in Fig. 4. In this case the weights were selected from $U(-10, 10)$ and biases were determined according to (8). As you can see from this figure, all sigmoids have their inflection points inside H. The number of hidden nodes to achieve $RMSE = 0.0084$ is 35. To obtain a similar level of error we need over 60 nodes when using the standard method for generating the parameters.

The following experiments concern multivariable function fitting. TF in this case is defined as:

$$g(\mathbf{x}) = \sum_{j=1}^{n} \sin(20 \cdot \exp x_j) \cdot x_j^2 \tag{10}$$

TF (10) is depicted in the upper left panel of Fig. 5. The training set contains N points $(\mathbf{x}_l, y_l)$, where $x_{l,j} \sim U(0, 1)$ and y_l are calculated from (10), then normalized in the range $[-1, 1]$ and distorted by adding noise $\xi \sim U(-0.2, 0.2)$. A test set of the same size is created in the same manner but without noise.

The experiments were carried out for $n = 2$ ($N = 5000$), $n = 5$ ($N = 20000$) and $n = 10$ ($N = 50000$), using:

– SM – the standard method of generating both weights and biases from $U(-u, u)$,

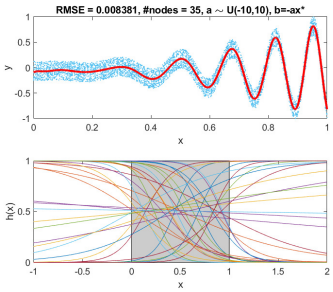

Fig. 4. TF (1) fitting: fitted curve and the sigmoids constructing it for the proposed method.

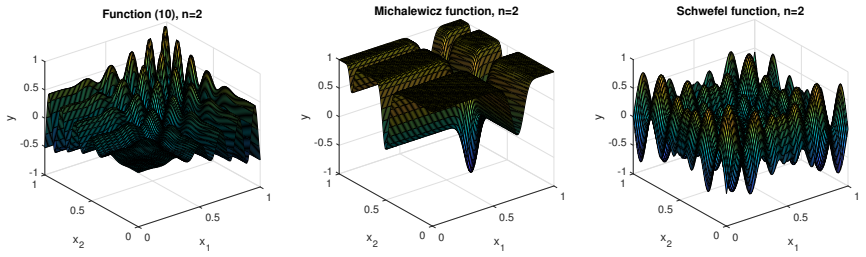

Fig. 5. Target functions.

- PMu – the proposed method of generating weights from $U(-u, u)$ and biases according to (8),
- PMα – the proposed method of generating slope angles from $U(\alpha_{\min}, 90°)$, then calculating weights from (9), and biases from (8).

Fig. 6 shows the mean test errors over 100 trials for different node numbers. For each node number the optimal value of u or $\alpha_{\min}$ was selected from $u \in \{1, 2, ..., 10, 20, 50, 100\}$ and $\alpha_{\min} \in \{0°, 10°, ..., 80°\}$, respectively. As you can see from Fig. 6, PMα in all cases leads to the best results. For $n = 2$ it needs less nodes to get a lower error (0.0352) than PMu and SM. Interestingly, for higher dimensions, using too many nodes leads to an increase in the error for SM and PMu. This can be related to the overfitting caused by the steep nodes generated by the standard method. In the same time, for PMα, where the node slope angles are distributed uniformly, an decrease in the error is observed.

Similar experiments were performed using the following highly nonlinear TFs:

- Michalewicz function:

$$g(\mathbf{x}) = -\sum_{i=1}^{n} \sin(x_i) \sin^{2m}\left(\frac{ix_i^2}{\pi}\right), \text{where } m = 10, x_i \in [0, \pi] \qquad (11)$$

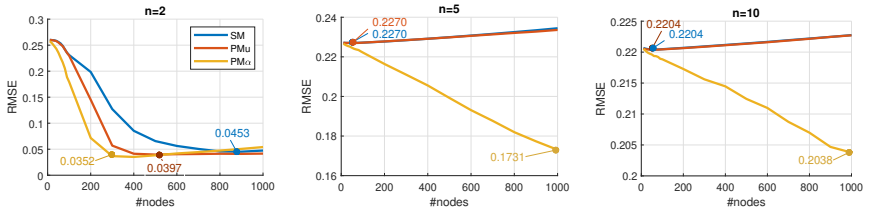

Fig. 6. RMSE depending on the number of nodes for function (10).

– Schwefel function:

$$g(\mathbf{x}) = 418.9829n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|}), \text{where } x_i \in [-500, 500] \tag{12}$$

These TFs after normalization in the range $[0, 1]$ for all x_i and $[-1, 1]$ for y are shown in Fig. 5. The training and test points are generated in the same way as for TF (10) and the training points are distorted by adding noise $\xi \sim U(-0.2, 0.2)$.

Figs. 7 and 8 show the mean test RMSE over 100 trials for different methods of generating random parameters, different number on nodes, and $n = 2$, 5 or 10. Note that PMα in all cases leads to the best results.

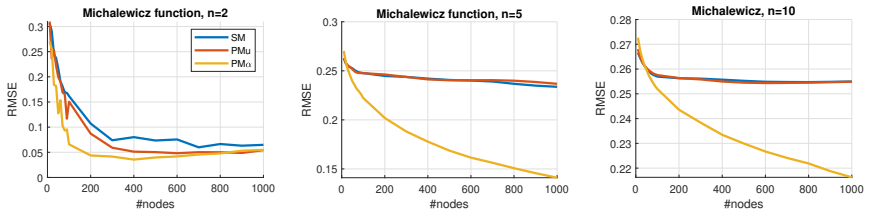

Fig. 7. RMSE depending on the number of nodes for Michalewicz function.

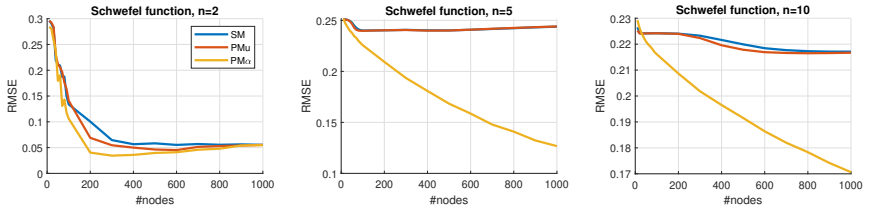

Fig. 8. RMSE depending on the number of nodes for Schwefel function.

5 Conclusion

A drawback of the standard method of generating random hidden nodes in FNNs is that many sigmoids have their most nonlinear fragments outside of the input hypercube, especially for low-dimensional cases. So, they cannot be used for modeling the target function fluctuations. Moreover, it is difficult to adjust the optimal values for weights and biases when the standard method selects these both parameters from the same interval.

In this work, we propose methods of generating random parameters which ensure that all the sigmoids have their steepest fragments inside the input hypercube. In addition, we show how to determine the weights to ensure the sigmoids have uniformly distributed slope angles. This prevents overfitting which can happen when weights are generated in a standard way, especially for highly nonlinear target functions. The proposed methods treat weights and biases separately due to their different functions. The weights are determined first and then biases are determined on the basis of weights and data distribution. The basic conclusions of this work can serve as general guidelines for designing FNNs with random hidden nodes for regression.

References

1. Cao, F., Wang, D., Zhu, H., Wang, Y.: An iterative learning algorithm for feedforward neural networks with random weights. Information Sciences **328**, 546–557 (2016).
2. Cao, W., Wang, X., Ming, Z., Gao, J.: A review on neural networks with random weights. Neurocomputing **275**, 278–287 (2018).
3. Dudek, G.: Generating random weights and biases in feedforward neural networks with random hidden nodes. Information Sciences **481**, 33–56 (2019).
4. Dudek, G.: Improving randomized learning of feedforward neural networks by appropriate generation of random parameters. In: Advances in Computational Intelligence. IWANN 2019. LNCS, vol. 11506, pp. 517–530. Springer (2019).
5. 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).
6. Li, M., Wang, D.: Insights into randomized algorithms for neural networks: Practical issues and common pitfalls. Information Sciences **382–383**, 170–178 (2017).
7. Principe, J., Chen, B.: Universal approximation with convex optimization: gimmick or reality? IEEE Computational Intelligence Magazine **10**(2), 68–77 (2015).
8. Scardapane, S., Comminiello, D., Scarpiniti, M., Uncini, A.: A semi-supervised random vector functional-link network based on the transductive framework. Information Sciences **364-365**, 156–166 (2016).
9. Wang, D., Li, M.: Stochastic configuration networks: Fundamentals and algorithms. IEEE Transactions on Cybernetics **47**(10), 3466–3479 (2017).
10. Zhang, L., Suganthan, P.: A comprehensive evaluation of random vector functional link networks. Information Sciences **367–368**, 1094–1105 (2016).
11. Zhang, L., Suganthan, P.: A survey of randomized algorithms for training neural networks. Information Sciences **364–365**, 146–155 (2016).