ALGORITHMS FOR INITIALIZATION OF NEURAL NETWORK WEIGHTS

A. Pavelka and A. Procházka

Institute of Chemical Technology, Department of Computing and Control Engineering

Abstract

The paper is devoted to the comparison of different approaches to initialization of neural network weights. Most algorithms based on various levels of modification of random weight initialization are used for the multilayer artificial neural networks. Proposed methods were verified for simulated signals at first and then used for modelling of real data of gas consumption in the Czech Republic.

1 Introduction

Initialization of coefficients of neural networks before their optimization represents a very important but also complicated research topic. The main problem is to select the global optimum from all possible local minima on the error surface and to start the optimization from the set of values as close to optimum as possible to minimize the number of the training cycles. Various methods including genetic algorithms can be applied in this stage to find initial values of coefficients. The paper is restricted to the presentation of properties and possibilities of the use of generators of random numbers in the Matlab environment. The main goal is to find suitable methods for setting random initial weights for neural networks.

2 Random Numbers in MATLAB

There are two main types of the generation of random values in the Matlab environment [1]. The first one uses *uniformly* distributed random numbers and arrays produced by the RAND function (Fig. 1). And the second one is based upon *normally* distributed random numbers and arrays generated by the RANDN function (Fig. 2).

Organization of random numbers in Matlab is described in the following example presenting results obtained for the random state set to zero and the use of the RAND function:

```
>> rand('state',0); rand
ans =
    0.9501
```

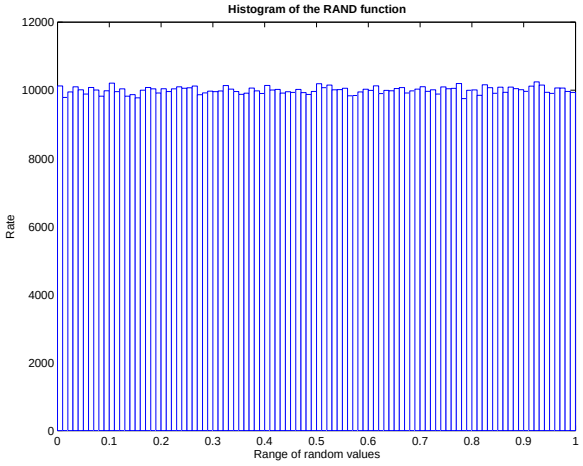

Figure 1: Histogram of the RAND function

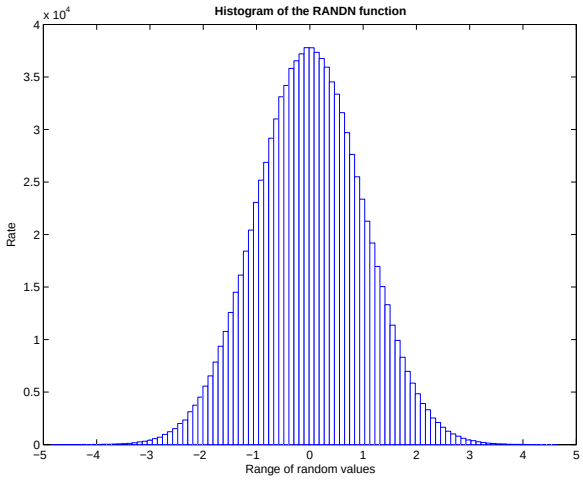

Figure 2: Histogram of the RANDN function

generating different random number after each following use of this function. The column vector of random values can be defined by command

```
>> rand('state',0); rand(5,1)
ans =
    0.9501
    0.2311
    0.6068
    0.4860
    0.8913
```

while the row vector of random values can be generated in the similar way using command

```
>> rand('state',0); rand(1,5)
ans =
    0.9501   0.2311   0.6068   0.4860   0.8913
```

and the random matrix can be obtained through function

```
>> rand('state',0); rand(3,3)
ans =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214
```

The Neural Network Toolbox [2] provides specific modification of random functions by commands RANDS, RANDNC and RANDNR. The RANDS is a symmetric random weight/bias initialization function. The RANDNC is a weight initialization function that generates a random weight matrix columns of which are normalized to the value of one. The RANDNR is a weight initialization function that generates a random weight matrix rows of which are normalized to the value of one.

The RANDS provides uniformly distributed random values symmetric around zero value. Output of this function is the same as RAND function, but multiplied by two and translated by one $(2 * RAND - 1)$. Effect of this operation moves the mean μ of generated values to zero instead of 0.5 of the original function and range of RAND from $\langle 0, 1\rangle$ to $\langle -1, 1\rangle$ as presented in Figs. 1, 3 and 4.

The RANDNC function is based on the RANDS function, that is normalized by columns. In the similar way the RANDR function is based on the RANDS function normalized by rows.

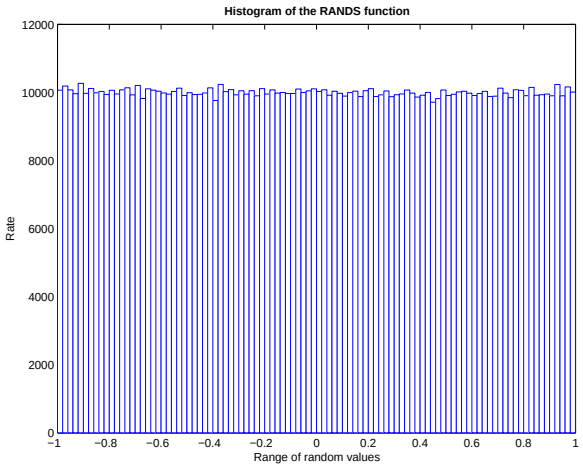

Figure 3: Histogram of the RANDS function

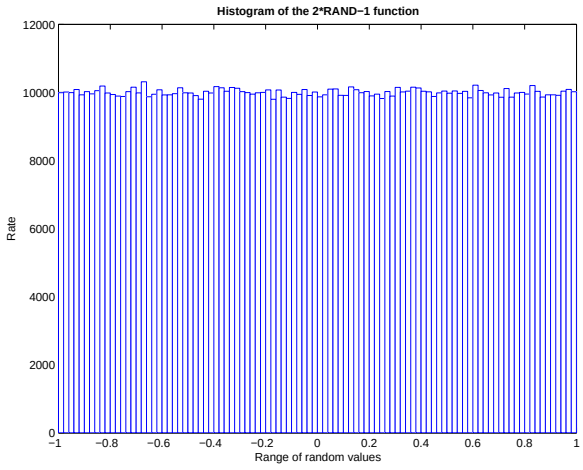

Figure 4: Histogram of the 2*RAND-1 function

Having a matrix $\mathbf{X}$ with elements x_{ij},

$$\mathbf{X} = \begin{bmatrix} x_{11} & \cdots & x_{1j} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots & & \vdots \\ x_{i1} & \cdots & x_{ij} & \cdots & x_{in} \\ \vdots & & \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mj} & \cdots & x_{mn} \end{bmatrix} \tag{1}$$

then normalized matrix by columns $\mathbf{X}^{\mathrm{C}}$ with elements x_{ij}^{C} and matrix normalized in rows $\mathbf{X}^{\mathrm{R}}$ with x_{ij}^{R} are defined by relations

$$x_{ij}^{\mathrm{C}} = x_{ij} \cdot \sqrt{\frac{1}{\sum_{j=1}^{n} x_{ij}^2}} \qquad\qquad x_{ij}^{\mathrm{R}} = x_{ij} \cdot \sqrt{\frac{1}{\sum_{i=1}^{m} x_{ij}^2}} \tag{2}$$

Graphical presentation of RANDNR and RANDNC function is shown in Fig. 5 and Fig. 6. Upper figures show histograms calculated over rows of random matrix and the lower over columns.

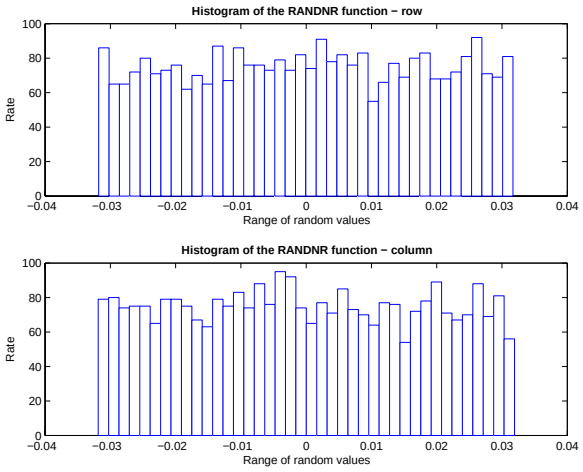

Figure 5: Histogram of the RANDNR function

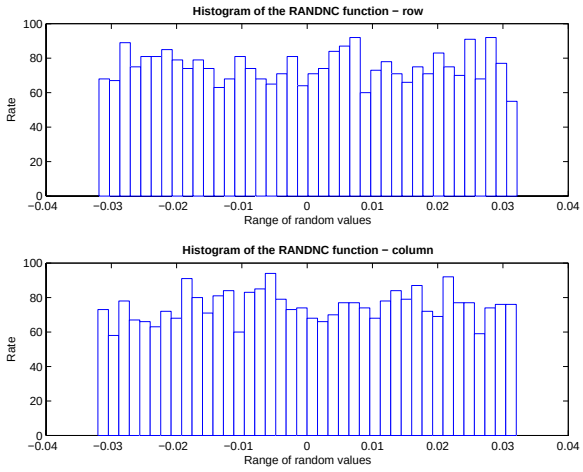

Figure 6: Histogram of the RANDNC function

3 Special Modification

The main algorithm used in the Matlab environment for initialization of two layer neural network is the Nguyen-Widrow layer initialization function called INITNW. This function is based on the paper [3]. The Nguyen-Widrow method generates initial weights and bias values for a layer, so that the active regions of the layers neurons will be distributed approximately evenly over the input space [2].

The main idea of the Nguyen-Widrow algorithm is as follows. We pick small random values as initial weights of the neural network. The weights are then modified in such a manner that the region of interest is divided into small intervals. It is then reasonable to consider speeding up the training process by setting the initial weights of the first layer so that each node is assigned its own interval at the start of training. As the network is trained, each hidden node still has the freedom to adjust its interval size and location during training. However, most of these adjustments will probably be small since the majority of the weight movements were eliminated by Nguyen-Widrow method of setting their initial values.

Let's have two-layer network with one input that is trained to approximate a function of one variable. This function is to be approximated by the neural network over the region $\langle -1,\ 1\rangle$, with its length equal to 2. There are H units in the first layer, therefore each of its units will

be responsible for an interval of length $2/H$ on the average. Since sigmoid$(w_i x + w_{bi})$, where w_i and w_{bi} are initial weights while x represent the input, is approximately linear over values

$$-1 < w_i x + w_{bi} < 1 \tag{3}$$

this yields the interval

$$-1/w_i - w_{bi}/w_i < x < 1/w_i - w_{bi}/w_i \qquad \text{for } w_i > 0 \tag{4}$$

$$-1/w_i - w_{bi}/w_i > x > 1/w_i - w_{bi}/w_i \qquad \text{for } w_i < 0 \tag{5}$$

which has the length $2/|w_i|$. Therefore

$$2/|w_i| = 2/H \qquad \Longrightarrow \qquad |w_i| = H \tag{6}$$

However, it is preferable to have slightly overlapping intervals, and so Nguyen and Widrow recommend to use $|w_i| = 0.7H$. Then w_{bi} are picked so that the intervals are located randomly in the region $-1 < x < 1$. The center of each interval is located at

$$x = -w_{bi}/w_i = \text{uniform random value between } -1 \text{ and } 1 \tag{7}$$

and so we will set

$$w_{bi} = \text{uniform random value between } -|w_i| \text{ and } |w_i| \tag{8}$$

Let us consider now a single layer of m neurons with p synapses, each including the bias. Then the output of the jth neuron using transfer function φ is defined by relation

$$y_j = \varphi(v_j), \qquad \text{where } v_j = \mathbf{w}_j \cdot \mathbf{x} \tag{9}$$

For an activation function $\varphi(v)$ we can specify its active region $\overline{v} = (v_{\min},\ v_{\max})$ outside which the function is saturated. For example, for the hyperbolic tangent we can assume that

$$\overline{v} = (-2,\ +2), \qquad \tanh(v) \in (-0.96,\ 0.96)$$

In addition we need to specify the range of input values,

$$\overline{x_i} = (x_{i,\min},\ x_{i,\max}) \qquad \text{for } i = 1, \ldots, p-1 \tag{10}$$

Assume at first that the range of input signals and non-saturating activation potential is (-1, +1). The initial weight vectors will now have evenly distributed magnitudes and random directions: for $p = 2$ the initialization is to generate m random numbers $a_j \in (-1,\ +1)$ for $j = 1, \ldots, m$ and then set up weights

$$\mathbf{W}(j,1) = 0.7\frac{a_j}{|a_j|}, \quad \mathbf{W}(:,2) = 0 \tag{11}$$

For $p > 2$ the initialisation assumes specification of the magnitude of the weight vectors as

$$G = 0.7m^{\frac{1}{p-1}} \tag{12}$$

then to generate m random unity vectors a_j, that is, generate an $m \times (p-1)$ array $\mathbf{A}$ of random numbers, $a_{ji} \in (-1,\ +1)$ and normalise it in rows:

$$\mathbf{a}_j = \frac{\mathbf{A}(j,:)}{\|\mathbf{A}(j,:)\|} \tag{13}$$

Using these values it is possible to set up weights

$$\mathbf{W}(j, 1:p-1) = G \cdot \mathbf{a}_j, \qquad \text{for } j = 1, \ldots, m \tag{14}$$

and bias values

$$\mathbf{W}(j,p) = \text{sgn}(\mathbf{W}(j,1)) \cdot G \cdot \beta_j, \qquad \text{for } \beta_j = -1 : 2/(m-1) : 1 \tag{15}$$

Finally, the weights are linearly rescaled to allow different ranges of activation potentials and input signals. Details can be found in the Matlab script, `initnw.m`.

Our modification of Nguyen-Widrow method was in the use of active input range $\overline{v} = \langle -2,\ +2 \rangle$ of the given transfer function, in our case hyperbolic tangents and maximal and minimal value of network's input. In the Matlab code the `P` is a matrix of inputs values, `S1` and `S2` is the number of neurones in the first and in the second layer. The main parts of the proposed algorithm include the following steps

% evaluation of minimal and maximal values of network's input

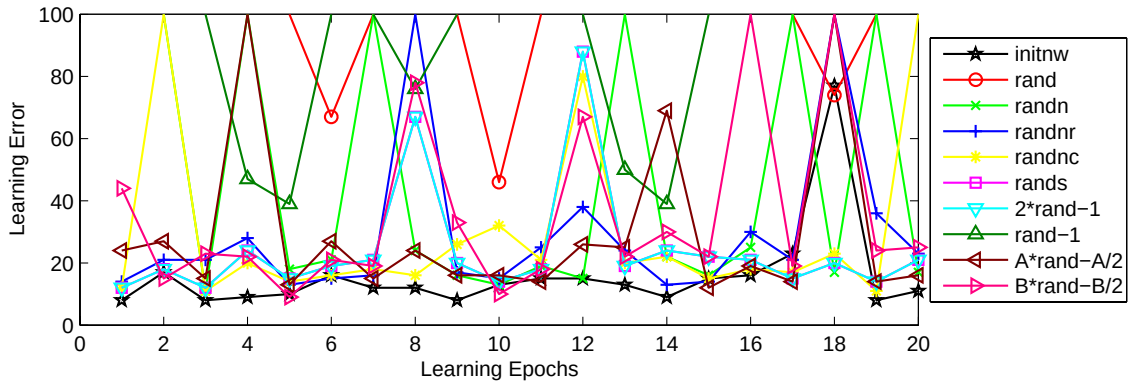

Figure 7: The number of learning epochs after the decrease of the learning error under the thresholding value of 10

```
mmP = minmax(P);
```

% creation of original neural network

```
net00 = newff(mmP,[S1 S2],{'tansig','purelin'});
```

% the selection and modification (parameter AB) of active input range of the transfer function

```
active = feval(net00.layers{1}.transferFcn,'active')*AB;
```

% specification of the modifying parameter from active input range of transfer function and maximal and minimal value of network's input

```
rnd_lim(1,1:2) = active(1)./(1+mmP(1,:));
rnd_lim(1,3:4) = active(2)./(1+mmP(1,:));
rnd_lim_w = max(rnd_lim)-min(rnd_lim);
```

% generation and modification of weights and biases for a specific random state

```
rand('state',ijk)
net09.IW{1} = rnd_lim_w*rand(S1,R)-rnd_lim_w/2;
net09.LW{2} = rnd_lim_w*rand(1,S1)-rnd_lim_w/2;
net09.b{1}  = rnd_lim_w*rand(S1,1)-rnd_lim_w/2;
net09.b{2}  = rnd_lim_w*rand(1,1)-rnd_lim_w/2;
```

4 Results

Initialization methods described above have been tested and analyzed for real data sets that represented observations of daily gas consumption measured with the sampling rate of one day. The verification set consisted of measurements taken in the Czech republic during the winter period from October the 1st, 1993 till June 30th, 2004. The reference model has been formed by the two layer neural network with architecture 5-8-1 having sigmoidal and linear transfer functions in the first and the second layers respectively trained in 100 learning epochs.

Ten algorithms for initialization of weights and bias of such a neural network has been used with twenty different initial random states. The learning process has been always interrupted after the epoch in which the learning error decreased under value of 10 with results summarized in the Table 1 and visualized in Fig. 7. Using this criterium and two types of scoring methods (see Fig. 8) it was possible to find that the best results have been achieved by the Nguyen-Widrow method while the worst ones have been obtained by application of the simple unmodified RAND function.

Another very efficient method for network weights initialization is based upon the modification of RAND function using function $2*RAND-1$ providing the same results as the RANDS function in the Neural network toolbox. Our modification resulting in relation $A*rand-A/2$,

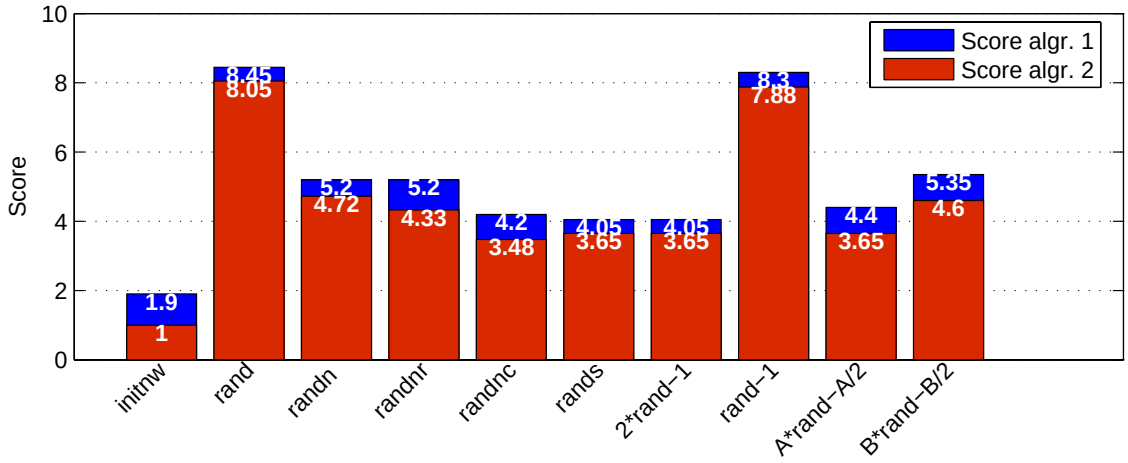

Figure 8: The score achieved by two types of scoring algorithm for comparison of results from Table 1 (less is better)

where $A = 0.72$ yields sufficient results as well. On contrary the similar case using network initialization by function $B * rand - B/2$, where B=1.43 result in significantly worse network outputs. Obviously the main reason for this result is the use of $RAND - 1$ modification where the range of random numbers is $\langle -1,\ 0\rangle$ and results of this nonsymmetric random range are summarized in Table 1 as well. According to these observations it is possible to recommend the use of symmetric range for the generation of random numbers for initial weights and biases of neural networks.

Table 1: The number of learning epoch after decreasing of the learning error under value 10

rand('state')	initnw	rand	randn	randnr	randnc	rands	2*rand-1	rand-1	A*rand-A/2	B*rand-B/2
1	8	100	100	14	12	12	12	100	24	44
2	17	100	100	21	100	18	18	100	27	15
3	8	100	11	21	11	12	12	100	15	23
4	9	100	100	28	20	24	24	47	100	22
5	10	100	18	13	14	15	15	39	13	9
6	16	67	21	15	16	19	19	100	27	21
7	12	100	100	16	18	21	21	100	15	19
8	12	100	24	100	16	67	67	76	24	78
9	8	100	16	17	26	20	20	100	16	33
10	13	46	13	15	32	14	14	100	16	10
11	15	100	19	25	21	18	18	100	14	18
12	15	100	15	38	80	88	88	100	26	67
13	13	100	100	24	20	19	19	50	25	22
14	9	100	22	13	22	24	24	39	69	30
15	15	100	16	14	15	22	22	100	12	22
16	16	100	25	30	18	21	21	100	19	100
17	23	100	100	21	17	15	15	100	14	19
18	77	74	17	100	23	20	20	100	100	100
19	8	100	100	36	11	14	14	100	14	24
20	11	100	17	23	100	21	21	100	16	25

5 Summary

To select initial weights and biases for a given neural network it is possible to recommend their initialization by an around zero symmetric random numbers. Even better results can be achieved by the use of the Nguyen-Widrow method for the weights initialization.

References

[1] Anonymous. *The MathWorks-Online Documentation*. The MathWorks, Inc., Natick, MA, release 14; online only) edition, 2004. http://www.mathworks.com.

[2] Howard Demuth and Mark Beale. *Neural Network Toolbox, User's Guide, Version 4*. The MathWorks, Inc., Natick, MA, revised for version 4.0.4 edition, October 2004. http://www.mathworks.com.

[3] Derrick Nguyen and Bernard Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. *Proceedings of the International Joint Conference on Neural Networks*, 3:21–26, 1990.

Ing. Aleš Pavelka, Prof. Aleš Procházka
Institute of Chemical Technology, Prague
Department of Computing and Control Engineering
Technická 1905, 166 28 Prague 6
Phone.: 00420-2-2435 4198, Fax: 00420-2-2435 5053
E-mail: ales.pavelka@volny.cz, A.Prochazka@ieee.org
WWW: http://dsp.vscht.cz