ISSN 1060-992X, Optical Memory and Neural Networks, 2021, Vol. 30, No. 2, pp. 180–186. © Allerton Press, Inc., 2021.

Random Search Algorithm with Self-Learning for Neural Network Training

V. A. Kostenko[a], * and L. E. Seleznev[a], **

[a] *Lomonosov Moscow State University, Moscow, 119991 Russia*

**e-mail: kostmsu@gmail.com*

***e-mail: lev4213@gmail.com*

Received December 12, 2020; revised March 10, 2021; accepted March 11, 2021

Abstract—We discuss a random search algorithm with self-learning designed for solving a problem of training of feedforward neural networks and compare it with gradient algorithms for neural network training with regard to criteria of accuracy and computational complexity.

Keywords: neural networks, machine learning, random search, gradient algorithms for neural network training

DOI: 10.3103/S1060992X2102003X

INTRODUCTION

Gradient algorithms are frequently used for neural network (NN) training. These algorithms are locally optimal. The basic limitation when using the gradient-based algorithms is a strong dependence of quality of the obtained solution on the choice of the point of initial approximation. This problem is the most acute when the number of local minima is large. A universal way to eliminate this shortcoming is application of multi-start methods [1]. They repeatedly run such algorithms from different points of initial approximation; however, this leads to a substantial increase of their computational complexity and does not guarantee that the solution would be optimal.

In [2], we proposed a multi-start algorithm with cutting. The main idea of this algorithm is to launch (to initialize) a number of parallel or pseudoparalell runs (starts) of a local optimal algorithm with different initial approximations. Herewith at the early stages of the algorithm work, we find "unpromising" starts. We cut off these starts and continue the process of solution search on a narrower set of starts. In such a way, we can decrease the number of steps necessary to find the solutions launching at ineffective starts and thus decrease the run time of the algorithm. However, this method does not allow us to solve the problem of algorithm stopping at local optima.

When using the gradient algorithms the second problem is their high computational complexity, which is due to the necessity to calculate the gradients at each step of the algorithm. A back propagation method was developed for training feedforward neural networks. D. Rumelhart and G. Hinton are believed to be the first who proposed this algorithm in [3]; however, a theoretical basis sufficient for an implementation of this algorithm was developed in earlier papers [4–6]. The backpropagation algorithm uses a technique allowing one to calculate quickly the vector of partial derivatives (the gradient) of a complex multivariable function if we know the structure of this function. We can increase the efficiency of this algorithm using some of its modifications; for example, the Levenberg–Marquardt algorithm [7].

The other problem is choosing of a step size for variation of the parameters we have to optimize. If the step size is too small, the algorithm converges very slowly; however, when it is very large, the algorithm can lose its stability [8].

In the present paper, we analyze a possibility to use a random search algorithm with self-learning in place of locally optimal gradient algorithms [9]. It allows us:

- to leave local optima,
- its speed is comparable with speeds of the gradient algorithms [9] in the areas with regular relief of the error function,
- it has a low computational complexity of one step.

2. DIRECTED RANDOM SEARCH ALGORITHMS WITH SELF-LEARNING

In this section, we discuss the main principles of development for the directed random search algorithms with self-learning [9].

The basis of the directed random search method is an interaction process.

$$X_{k+1} = X_k + \alpha_k \frac{\xi}{\|\xi\|}, k = 0,1,\ldots,$$

In this equation α_k is the step size and $\xi = (\xi_1, \ldots, \xi_n)$ is a realization of a random n-dimensional vector ξ.

The directed random search algorithms with self-learning involve a restructuring of probabilistic search characteristics that is in a directed impact on a random vector ξ.

We can do this introducing a memory vector

$$P^k = \left(p_1^k, p_2^k, \ldots, p_n^k\right),$$

where p_j^k is a probability of choosing a positive direction for the jth coordinate at the kth step.

Three elements define the directed random search algorithms with self-learning:

(1) An algorithm for selecting trial states at the current step.

(2) A decision rule according which at each step the algorithm chooses a new current approximation of the solution.

(3) A self-learning algorithm, which refines the memory vector with account for information obtained at the current step.

Let us examine a self-learning algorithm with an arbitrary law of the probability change [9].

Suppose $p_j^{(k)} = p\left(w_j^{(k)}\right)$. The forms of the function $p(w)$ may be different but it has to be

- monotone,
- non-decreasing.

In [9], they propose the following functions $p(w)$:

a linear function

$$p(w) = \begin{cases} 0 & \text{when } w < -1 \\ \frac{1}{2}(w-1) & \text{when } -1 \le w < 1, \\ 1 & \text{when } 1 \le w \end{cases}$$

and an exponential function

$$p(w) = \begin{cases} \frac{1}{2} e^{aw}, & \text{when } w \le 0 \\ 1 - \frac{1}{2} e^{-aw}, & \text{when } w > 0 \end{cases},$$

$w_j^{(k+1)} = w_j^{(k)} - \delta \operatorname{sign}\left(\Delta x_j^{(k)} \Delta f^{(k)}\right)$, δ is a step that defines the learning rate.

Consequently, in the course of its work the algorithm redefine the probabilistic characteristics of the search by means of a goal-directed impact on a random vector ξ. After that, it ceases to be equiprobable and as a result of self-learning it gains a certain advantage in the directions of the best steps. This is achieved by introducing a memory vector. At each iteration the algorithm recurrently corrects the values of the components of this vector depending on how successful/unsuccessful (how much the value of the objective function changed) was the previous step.

3. APPLICATION OF RANDOM SEARCH ALGORITHM WITH SELF-LEARNING FOR NEURAL NETWORK TRAINING

We define a neural network (NN) as a triplet $NET = (G, T, W)$ where

- G is a directed acyclic graph defining the topology of the neural network: its vertexes correspond to neurons and its arcs correspond to interneural connections;
- T is the set of the neural network transition function;

• W is the set of the neural network weight coefficients.

By NN initialization, we mean a NN with a given architecture after assigning it an initial set of weight coefficients.

A finite set $S = \{(X_i, Y_i)\}$, $i = 1, 2, ..., P$, of P pairs "argument/value of function under approximation" will be called a sample. Here $X_i \in R^N$, $Y_i \in R^M$; N is a number of input neurons (the number of function arguments) and M is a number of the output neurons. The sample is correct if $X_i \neq X_j$ if $\forall i, j$: $i \neq j$.

The problem of a supervised learning of a neural network with a given architecture is to select such values of the weight coefficients, which will allow us to maximize the accuracy of solving of an applied problem. When we use the NN to approximate a tabulated continuous function, we can estimate the accuracy of the approximation by the value of the mean square error (MSE) [10]:

$$MSE(NET, S) = \frac{1}{P}\sum_{i=1}^{P}(Y_i - y_i)^2,$$

where y_i is a vector of the values of output neurons of the NN. When we apply the NN for pattern, recognition we estimate the accuracy by the value of MATCH:

$$MATCH(NET, S) = \frac{1}{P}\sum_{i=1}^{P} I_{Y_i = y_i},$$

where y_i is a class number to which the network has assigned the input pattern.

For a given architecture of the NN, the supervised learning algorithm includes the following steps [8]:

(1) To initialize the NN architecture (to assign it an initial set of weight coefficients).

(2) To choose a subsequent training pair.

(3) To calculate the NN output.

(4) Depending on the problem, to calculate the error (MSE) or the accuracy (MATCH) of the NN.

(5) To minimize the error by correcting the NN weights.

(6) To repeat the steps 2–5 for each pair of the sample S until the error reaches an acceptable level on the entire set S.

To initialize the neural network we use the He-at-al method [11]. In the framework of this method we examine fully connected neural networks consisting of L neuron layers with l_k neurons in the kth layer. We define the initialization of the weight of the connection of the ith neuron from the $k-l$th layer by the jth neuron of the kth layer as

$$W_{kij} = \xi\sqrt{\frac{2}{l_k}}, \quad \xi \sim N(0, 1).$$

To perform the step 5 they usually use one of the backpropagation algorithms. The major disadvantages of these methods are their high computational complexity of calculation of the gradient and inability to escape from "bad" local optima.

In the present paper, to eliminate these defects we propose to use the random search algorithm with self-learning at the step 5.

We feed the algorithm with the input NN weight coefficients, which are the parameters we have to optimize; and depending on the problem, we use the error (MSE) or the accuracy (MATCH) as the objective functions.

For the initial initialized neural network, the memory vector is initialized as $P = (0.5, ..., 0.5)$; p_i defines the probability to make a positive step in the direction of changing of the weight w_i in the given neural network. As the parameters of the algorithm, we define the number of steps K, a slowing coefficient α ($\alpha \le 1$), and a limiting probability ε ($\varepsilon \le p_i^k \le 1 - \varepsilon$). As an initial value of the step size of changes of the weights we choose $D = 1.0$. The algorithm works step-by-step:

(1) The counter of the number of iterations k is set to 0.

(2) The value of $E_1 = MSE(NET, S)$ (or $E_1 = 1 - MATCH(NET, S)$ in the case of a pattern recognition problem) is calculated for a given neural network.

(3) The test selection of the direction is performed:

— For each weight w_i with a corresponding probability p_i the direction of the step h_i (positive or negative) is chosen.

— The previous value of the weight $v_i = w_i$ is stored. A new value of each weight $w_i = h_i D$ is calculated.

(4) By analogy with the step 2, the metrics E_2 is calculated.

(5) If $E_1 \le E_2$, then $E_2 = E_1$, and the jump to the step 7. Otherwise, the trial step is considered as unsuccessful.

(6) Step is canceled: $w_i = v_i$. Since in the test direction the error increased, $p_i = \max(\varepsilon, \min(1 - \varepsilon, p_i \Delta_i))$, where Δ_i is equal to 0.5, if $h_i > 0$, and it is equal to 2, if $h_i < 0$. The value of the step changes as $D = D\alpha$. The jump to the step 3.

(7) The counter k increases by 1. If $k = K$, the algorithm terminates. In other case, $D = D0.5$ and the jump to the step 2.

The output of the algorithm is one neural network over which the random search algorithm with self-learning performed K training steps.

4. COMPARISON OF PROPOSED AND GRADIENT ALGORITHMS

To compare our random search algorithm with self-learning described in the previous Section with the backpropagation algorithm we use the following criteria

(1) Accuracy (the value of MSE after training.)

(2) Training time on a specific device.

(3) Total number of testing iterations (including unsuccessful steps for the random search method with self-learning.)

The characteristics of the test medium are:

• Processor Intel® Core™ i7 3520M, 2 × 2.8 GHz,

• RAM: 8 Gb, DDR3.

When testing we performed an averaging over the results of some tests.

Prior testing with the aid of the He-at-al method [11] we generated the initialized neural network.

For testing the training algorithms, we chose the following problems:

(1) MNIST dataset as an example of problems with a large number of parameters.

(2) Approximation of functions of 1, 2 or 3 variables as an example of problems with small number of parameters.

We consider the fully connected feedforward neural networks and choose a sigmoid function as an activation function. We define the neural network by a number of layers and a number of neurons in each layer. For simplicity, we symbolized the neural network topology as $[l_1, l_2, ..., l_L]$, where L is the number of layers, l_i is the number of neurons in the ith layer, l_1 is the size of the input layer and l_L is the size of the output layer.

For the MNIST problem, we performed the comparative testing on two neural networks, which are the Topology I—[768, 100, 10] and the Topology II—[768, 100, 50, 10]. A grayscale image of the size 28 by 28 points was fed to the network input and the maximal value of 10 outputs was interpreted as a selection of one of the numbers by the neural network. On the same device the testing time is measured under the same conditions.

When examining the approximation problem we considered the functions listed in Table 1 of Appendix. In Table 1, we show the topologies of the neural networks in the second and third columns.

In Tables 2–4 of Appendix, we show the averaged test results in the case of one start with the same initial initialization as well as the case of a number of starts with selection of the best result. We also present the averaged test results of multi-start versions of the algorithms for the number of starts equal to 10 with and without test steps; the value of the parameter $K = 8$. The training time and the number of iterations are considered in total for all the starts.

In Tables 2 and 3, we color the best result for each algorithm in light gray and the best result among all the algorithms in dark gray. In the first and second columns, we show the functions from Table 1 and the neural network topologies for these functions, respectively. In the table headers, for each test we define the algorithm and the values of the parameters of this algorithm (η is the learning rate parameter, which defines the rate in the direction of the error gradient, and α is a parameter of the random search).

Table 2 corresponds to the problem of function approximation in the case of one start. As we see from this table, there is only one case when the random search with self-learning and one start is more effective with regard to the error metrics than the gradient algorithm. The random search algorithm with self-learn-

Table 1. Neural network topologies for approximation problems

Function	Topology I	Topology II
$F_1(x) = x\frac{1}{2} + \frac{1}{2}$	[1, 5, 1]	[1, 5, 5, 1]
$F_2(x) = \sin(2\pi x)\frac{1}{2} + \frac{1}{2}$	[1, 7, 1]	[1, 7, 7, 1]
$F_3(x) = \sin(4\pi x)x\frac{1}{2} + \frac{1}{2}$	[1, 10, 1]	[1, 10, 10, 1]
$F_4(x) = \frac{1}{2}$	[1, 5, 1]	[1, 5, 5, 1]
$F_5(x, y) = xy\frac{1}{2} + \frac{1}{2}$	[2, 5, 1]	[2, 5, 5, 1]
$F_6(x, y) = \sin(\pi x)\sin(\pi y)\frac{1}{2} + \frac{1}{2}$	[2, 16, 1]	[2, 16, 16, 1]
$F_7(x, y) = \frac{1}{2}$	[2, 5, 1]	[2, 5, 5, 1]
$F_8(x, y, z) = xyz\frac{1}{2} + \frac{1}{2}$	[1, 5, 1]	[1, 5, 5, 1]
$F_9(x, y, z) = \frac{1}{2}$	[1, 5, 1]	[1, 5, 5, 1]

ing and without trial steps turned out to be the most ineffective. (The algorithm without trial steps is much the same as the random search algorithm with self-learning described above but it performs no computation of the values of E_1 and E_2, and the step 6 follows immediately after the step 4.) From Table 3 we see that sometimes the values of the errors differ only in the second decimal place of the coefficient of ten in negative power (for example, 1.02×10^{-3} and 1.03×10^{-3}). This means that in some sense the random search algorithm with self-learning is comparable with the backpropagation algorithm; however, it is still less effective. In average, the training time for the random search algorithm was 10 times more than the training time for the gradient method.

In Table 3 (the problem of function approximation), we show the results for multi-start versions of the random search and gradient algorithms. As we see from this table, now the random search algorithm with self-learning was better than the gradient method in 8 out of 18 cases. However, it has to be mentioned that in average the training time of the multi-start random search algorithm is 10 times larger the training time of the gradient method, and that may be crucial when choosing the training algorithm.

In Tables 4 and 5 (the problem of recognition of handwritten digits) the best results for each algorithm separately are shown in light gray. The best result among all the algorithms we colored in dark gray. In table headers we indicate the algorithm and the parameters of this algorithm for each test (η is the learning rate parameter, which defines the rate in the direction of the error gradient, and α is a parameter of the random search).

In Table 4 (the problem of recognition of handwritten digits), we show the test results for the random search algorithms with self-learning with and without trial steps and the gradient algorithm. As we see from this Table, the result of the random search is 5 times worse than the result of the gradient method; and, consequently, it does not applicable in this problem. It also has to be mentioned that in average the training time for the random search algorithm with self-learning was 100 times larger the training time for the gradient method.

In Table 5 (the problem of recognition of handwritten digits), we show the test results for multi-start versions of the random search algorithm with self-learning and the gradient method. As we see, the result of the random search is 5 times worse than the result of the gradient method; and, consequently, it does not applicable in this problem. It also has to be mentioned that in average the training time for the random search algorithm with self-learning was 100 times larger the training time for the gradient method.

Table 2. Averaged test results of approximation for gradient method and for random search with and without trial steps

		Gradient			Random search with trial steps		Random search without trial steps
		$\eta = 0.05$	$\eta = 0.1$	$\eta = 0.5$	$\alpha = 0.9$	$\alpha = 1.0$	$\alpha = 0.9; 1.0$
F1	**I**	1.22×10^{-2}	3.60×10^{-3}	4.83×10^{-4}	8.99×10^{-3}	7.23×10^{-3}	5.21×10^{-2}
	II	2.01×10^{-2}	1.63×10^{-2}	5.08×10^{-4}	2.48×10^{-2}	2.27×10^{-2}	4.10×10^{-1}
F2	**I**	1.17×10^{-1}	9.94×10^{-2}	8.76×10^{-2}	1.48×10^{-1}	1.37×10^{-1}	2.32×10^{-1}
	II	1.24×10^{-1}	1.23×10^{-1}	7.57×10^{-2}	2.08×10^{-1}	1.99×10^{-1}	3.09×10^{-1}
F3	**I**	3.49×10^{-2}	3.42×10^{-2}	3.51×10^{-2}	3.90×10^{-2}	3.32×10^{-2}	6.31×10^{-2}
	II	3.51×10^{-2}	3.44×10^{-2}	3.31×10^{-2}	3.70×10^{-2}	3.78×10^{-2}	3.40×10^{-1}
F4	**I**	4.79×10^{-4}	2.26×10^{-4}	4.10×10^{-5}	6.71×10^{-3}	2.31×10^{-3}	2.93×10^{-1}
	II	3.00×10^{-4}	1.44×10^{-4}	2.64×10^{-5}	1.43×10^{-3}	2.66×10^{-5}	1.59×10^{-2}
F5	**I**	1.05×10^{-4}	6.04×10^{-5}	9.71×10^{-7}	1.45×10^{-4}	2.43×10^{-5}	9.55×10^{-3}
	II	7.20×10^{-7}	6.27×10^{-7}	2.17×10^{-7}	6.74×10^{-5}	1.58×10^{-6}	1.53×10^{-1}
F6	**I**	3.63×10^{-5}	6.83×10^{-6}	5.33×10^{-9}	2.35×10^{-4}	1.16×10^{-4}	5.71×10^{-2}
	II	4.19×10^{-5}	3.21×10^{-5}	4.43×10^{-6}	4.13×10^{-5}	4.32×10^{-5}	5.45×10^{-2}
F7	**I**	3.93×10^{-4}	1.87×10^{-4}	3.42×10^{-5}	4.14×10^{-2}	2.59×10^{-2}	3.53×10^{-1}
	II	3.86×10^{-4}	1.84×10^{-4}	3.37×10^{-5}	3.29×10^{-3}	2.89×10^{-3}	1.19×10^{-2}
F8	**I**	1.00×10^{-4}	6.15×10^{-5}	1.78×10^{-6}	1.33×10^{-4}	1.08×10^{-3}	1.32×10^{-4}
	II	7.30×10^{-7}	7.07×10^{-7}	5.43×10^{-7}	5.26×10^{-5}	1.35×10^{-4}	2.79×10^{-2}
F9	**I**	3.54×10^{-4}	1.69×10^{-4}	3.13×10^{-5}	1.25×10^{-2}	6.16×10^{-3}	2.53×10^{-1}
	II	3.98×10^{-4}	1.86×10^{-4}	3.27×10^{-5}	5.70×10^{-3}	$2.35 \times 10-^{3}$	1.14×10^{-1}

Table 3. Averaged test results for approximation problems for gradient and random search with trial steps algorithms with multi-starts

		Gradient			Random search with trial steps	
		$\eta = 0.05$	$\eta = 0.1$	$\eta = 0.5$	$\alpha = 0.9$	$\alpha = 1.0$
F1	**I**	2.95×10^{-3}	1.22×10^{-3}	4.83×10^{-4}	8.99×10^{-3}	7.23×10^{-3}
	II	1.71×10^{-2}	8.58×10^{-3}	3.75×10^{-4}	1.40×10^{-2}	5.02×10^{-3}
F2	**I**	1.11×10^{-1}	9.49×10^{-2}	7.71×10^{-2}	1.17×10^{-1}	1.03×10^{-1}
	II	1.24×10^{-1}	1.23×10^{-1}	7.57×10^{-2}	1.23×10^{-1}	1.19×10^{-1}
F3	**I**	3.37×10^{-2}	3.38×10^{-2}	3.42×10^{-2}	3.33×10^{-2}	3.19×10^{-2}
	II	3.51×10^{-2}	3.44×10^{-2}	3.31×10^{-2}	3.23×10^{-2}	3.13×10^{-2}
F4	**I**	3.44×10^{-4}	1.64×10^{-4}	3.04×10^{-5}	7.66×10^{-4}	8.88×10^{-6}
	II	3.00×10^{-4}	1.41×10^{-4}	2.53×10^{-5}	1.43×10^{-3}	1.18×10^{-5}
F5	**I**	3.58×10^{-7}	2.19×10^{-7}	4.16×10^{-9}	3.51×10^{-6}	2.43×10^{-5}
	II	1.78×10^{-8}	1.72×10^{-8}	1.21×10^{-8}	9.19×10^{-7}	9.40×10^{-7}
F6	**I**	1.29×10^{-6}	3.85×10^{-7}	5.33×10^{-9}	1.12×10^{-7}	6.16×10^{-7}
	II	9.72×10^{-9}	6.99×10^{-9}	7.65×10^{-10}	1.84×10^{-5}	1.54×10^{-6}
F7	**I**	3.45×10^{-4}	1.66×10^{-4}	3.09×10^{-5}	1.21×10^{-4}	7.69×10^{-5}
	II	3.56×10^{-4}	1.67×10^{-4}	2.99×10^{-5}	2.61×10^{-7}	4.27×10^{-8}
F8	**I**	3.60×10^{-7}	2.44×10^{-7}	1.86×10^{-10}	2.24×10^{-6}	5.05×10^{-5}
	II	2.52×10^{-7}	2.47×10^{-7}	2.01×10^{-7}	5.07×10^{-8}	1.05×10^{-5}
F9	**I**	3.47×10^{-4}	1.67×10^{-4}	3.12×10^{-5}	4.91×10^{-6}	3.84×10^{-6}
	II	2.98×10^{-4}	1.44×10^{-4}	2.72×10^{-5}	1.59×10^{-4}	4.86×10^{-6}

Table 4. Averaged test results for MNIST for the gradient, random search with trial steps, and random search without trial steps algorithms

	Gradient			Random search with trial steps		Random search without trial steps
	$\eta = 0.05$	$\eta = 0.1$	$\eta = 0.5$	$\alpha = 0.9$	$\alpha = 1.0$	$\alpha = 0.9; 1.0$
I	9.25×10^{-1}	9.39×10^{-1}	9.54×10^{-1}	1.86×10^{-1}	2.01×10^{-1}	9.33×10^{-2}
II	9.17×10^{-1}	9.33×10^{-1}	9.52×10^{-1}	1.43×10^{-1}	1.85×10^{-1}	1.07×10^{-1}

Table 5. Averaged test results for MNIST for the gradient and random search with trial steps with multi-starts

	Gradient			Random search with trial steps	
	$\eta = 0.05$	$\eta = 0.1$	$\eta = 0.5$	$\alpha = 0.9$	$\alpha = 1.0$
I	9.25×10^{-1}	9.39×10^{-1}	9.55×10^{-1}	1.87×10^{-1}	2.01×10^{-1}
II	9.19×10^{-1}	9.35×10^{-1}	9.52×10^{-1}	1.64×10^{-1}	1.85×10^{-1}

6. CONCLUSIONS

As our simulations show, the efficiency of the multi-start random search algorithm is comparable with the gradient training algorithm only in half of the presented approximation problems and is not comparable with it for any of the MNIST problems. The proposed algorithm can be used for the problems with a small number of the parameters; however even in the problems of approximation of simple functions, it may be inefficient with regard to the error metrics. A separate and frequently important minus of this algorithm is the large time of its performance. With regard to this parameter, it again turned out to be many times worse than the gradient method.

Thus, when using the multi-start method, the random search algorithm with self-learning may be applicable in the approximation problems with a small number of parameters if the training time is not important. It is useless in other cases due it to inefficiency.

FUNDING

The work was supported by RBRF grant no. 19-07-00614.

REFERENCES

1. Zhiglyavsky, A.A. and Zhilinskas, A.G., *Methods for Finding a Global Extremum,* Moscow: Nauka, 1991.
2. Kostenko, V.A., Multi-start method with cutting for solving problems of unconditional optimization, *Opt. Mem. Neural Networks,* 2020, vol. 29, no. 1, pp. 30–36.
3. Rumelhart, D., Hinton, G., and Williams, R., Learning internal reprentations by error propagation, in *Parallel Distributed Processing,* Cambridge, MA: MIT Press, 1986, vol. 1, pp. 318–362.
4. Parker, D., Learning logic, Invention Report S81-64, File 1, Office of Technology Licensing, Stanford, CA: Stanford University, 1982.
5. Werbos, P., Beyond regression: New tools for prediction and analysis in the behavioral sciences, *Masters Thesis,* Harward Univ., 1974.
6. Bartsev, S.I. and Okhonin, V.A., Adaptive information processing networks, *Preprint of Inst. of Philosophy, Sib. Branch USSR Acad. Sci.,* Krasnoyarsk, 1986, no. 59B.
7. Hagan, M. and Menhaj, M., Training feedforward networks with the Marquardt algorithm, *IEEE Trans. Neural Networks,* 1994, vol. 5, no. 6, pp. 989–993.
8. Wasserman, F., *Neurocomputer Technology,* Moscow: Mir, 1992.
9. Rastrigin, L.A., *Statistical Search Methods,* Moscow: Nauka, 1968.
10. MSE. https://en.wikipedia.org/wiki/Mean_squared_error.
11. Priyesh Patel, Meet Nandu, and Purva Raut, Initialization of weights in neural networks, *Int. J. Sci. Eng. Dev. Res.*, 2018, vol. 3, no. 11, pp. 73–79.