ISSN 1060-992X, Optical Memory and Neural Networks, 2020, Vol. 29, No. 1, pp. 30–36. © Allerton Press, Inc., 2020.

Multi-Start Method with Cutting for Solving Problems of Unconditional Optimization

V. A. Kostenko*

Lomonosov Moscow State University, Moscow, 119991 Russia

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

Received August 26, 2019; revised December 28, 2019; accepted December 30, 2019

Abstract—In the present paper, we present a multi-start method with dynamic cutting of "unpromising" starts of locally optimal algorithms for solving continuous unconditional optimization problems. We also describe our results of testing of the proposed method for solving problems of feed-forward neural network training.

Keywords: problems of unconditional optimization, local optimal algorithm, gradient methods, multi-start method, neural networks, machine learning

DOI: 10.3103/S1060992X20010099

1. INTRODUCTION

To solve the continuous unconditional optimization problems many researchers frequently use the gradient methods. The algorithms that use gradients are local optimal. When using these methods the main problems are:

• The choice of an initial approximation of the solution. This choice determines an optimum to which the algorithm converges.

• The choice of a step size when changing the optimized parameters. The algorithm's rate of convergence and its stability depend on the step size.

When developing different new modifications of the gradient methods and the approaches for solving the listed problems, the goal is to reduce their computational complexity and to increasethe accuracy.

Most fully, the obtained solutions of these problems relate to the training of feed-forward neural networks, however, they are rather special and not suitable for other problems.

In this paper, we propose a universal multi-start method with dynamic cutting of "unpromising" starts that is appropriate to any continuous unconditional optimization problem. We use the problem of neural network training to examine the method efficiency with regard to criteria of its accuracy and complexity.

In paper [1], D. Rumelhart, G. Hinton, and R. Williams developed a backpropagation algorithm for feed-forward neural network training. It is believed that these authors were the first who presented this algorithm but you can find a theoretical background sufficient for implementation of this algorithm in earlier papers [2–4]. The concept of the backpropagation algorithm is a method allowing one to calculate quickly vectors of partial derivatives (gradients) of complex functions of many variables. To increase the efficiency of this method the one can use a number of its modifications. The Levenberg–Marquardt algorithm [5] is one of examples. However, you cannot apply these algorithms for most other continuous unconditional optimization problems.

The main trouble with the gradient algorithms is their strong dependence of the quality of the obtained solution on the choice of the starting approximation point. The multi-start method is a universal tool to eliminate this shortcoming [6]. In the framework of this method, we start the algorithm from different starting approximation. However, this leads to a significant increase of the computational complexity. In the same time, the method does not guarantee the optimal solution.

The authors of [7–10] proposed algorithms for choosing the starting approximation points when training a neural network with one inner layer. They showed that such algorithms improved the solution quality and speed up the training process comparing with a random choice of the starting approximation point. Yam and Chow [11] suggested a method that developed the idea of D. Nguyen and B. Widrow [9] based on dividing the interval of the values of weighted coefficients of hidden neurons in such a way that their outputs fell into their active regions. Contrary to the Nguyen and Widrow algorithm, the Yam and Chow algorithm is suitable for initializing of the weights of feed-forward neural networks with some hidden layers.

In addition, a question about an optimal choice of the step size for the optimized parameters arises. If the step size is too small, the convergence of the algorithm is too slow. In the same time, if the step size is very large, the algorithm can lose its stability [12]. Adaptive algorithms for calculation of a step size for correction of optimized variables (a Quick PROP algorithm [13] and a RPROP algorithm [14]) allow us to adjust the correction value for the particular problem and speeds up the algorithm convergence.

In the present paper, we propose a multi-start method with cutting for solving continuous unconditional optimization problems and present the results of its experimental implementation in the problems of training of the feed-forward neural networks.

2. MULTI-START METHOD WITH DYNAMIC CUTTING

In the framework of a continuous unconditional optimization problem, we have to find the components of the vector $X = (x_1, ..., x_n)$, $X \in R^n$ (the optimized parameters) that minimize the objective function $y = f(X)$.

The main idea of the multi-start method with dynamic cutting is to perform (to initialize) some parallel or pseudoparallel launches (starts) of a local optimal algorithm with different starting approximations. At early stages of its work, the algorithm detects "unpromising" starts and excludes them from the consideration. Then the decision-making process continues on a narrower set of starts. In such a way, we can diminish the number of steps necessary for the decision-making, which we perform from the failed starts and decrease the time of the algorithm work.

Let $M = \{K, D(C, A)\}$ be the dynamic cut method, where

- A is the number of the start initialization, $A \in N$, $A > 1$.
- C is the maximal number of steps for one initialization variant, $C \in N$.
- D is the cutting scheme that we define as a finite set of pairs of positive integers $D(C, A) = \{(c_i, a_i)\}$ satisfying the conditions $\sum_{i=1}^{n} c_i \leq C, \sum_{i=1}^{n} a_i = A - 1$, where n is the number of the algorithm stages, c_i is the number of steps at the i-th stage of each initialization, and a_i is the number of cuttings at the i-th stage.
- K is the cutting criterion estimating the start "promising".

For any continuous unconditional optimization problem universal cutting criterions are

- The value of the objective function $y = f(X)$.
- The rate of decrease of the value of the objective function
- The weighted sum of the first two criteria.

The computational complexity of the values of these criteria is only a small part of the computational complexity of the whole algorithm. We can use criteria that are more complex. For example, we can determine the convergence of various initializations to the same optimum. However, such criterions can increase the algorithm complexity significantly.

As input data for the multi-start algorithm with cutting, we have to define the following:

- an objective function $y = f(X)$;
- a local optimal algorithm L;
- a distribution function of a random variable F.

The cutting method $M = \{K, D(C, A)\}$, determining the way of cutting of the "unpromising" initializations during the training process, defines the set of the algorithm parameters.

Below we describe the work of our multi-start algorithm with dynamic cutting of the "unpromising" starts:

1. The algorithm performs A initializations of the local optimal algorithm. In accordance with the distribution function F, it chooses randomly each of the optimized parameters and numbers all the initializations. When the algorithm runs, an initialization sequence number does not change.

2. The algorithm performs n stages; each i–th stage consists of c_i steps of the remained initializations of the local optimal algorithm. Then a_i initializations with minimal values of the cutting criterion K_i are removed. In this process the algorithm performs the search for solutions in such a way that for any natural

numbers p and q consistent application of p and then q steps is equivalent to application of $p + q$ steps simultaneously.

3. The initialization of the local optimal algorithm left after the n-th cutting stage passes through $C - \sum_{i=1}^{n} c_i$ steps.

4. We consider the obtained remaining initialization as a resulting and its result is the result of the algorithm run.

The total number of the local optimal algorithm steps through all the initialization is equal to $C_\Sigma = \sum_{i=1}^{n}\left(A - \sum_{j=0}^{i-1} a_j\right)c_i + C - \sum_{i=1}^{n} c_i = C + A\sum_{i=1}^{n} c_i - \sum_{i=1}^{n}\sum_{j=0}^{i-1} a_j c_i - \sum_{i=1}^{n} c_i$, where $a_0 = 0$.

We will call a multi-start algorithm a special case of the multi-start algorithm with cutting where the number of cuttings $n = 1$. It is clear that the total number of the steps of the multi-start algorithm is equal to CA.

Statement. The total number of the steps of the multi-start algorithm with cutting through all the initializations and stages at arbitrary values of c_i and a_i is less than the number of steps of the multi-start algorithm without cutting.

The proof.

Let us examine the difference between CA and C_Σ:

$$CA - C_\Sigma = CA - C - A\sum_{i=1}^{n} c_i + \sum_{i=1}^{n}\sum_{j=0}^{i-1} a_j c_i + \sum_{i=1}^{n} c_i = (A-1)\left(C - \sum_{i=1}^{n} c_i\right) + \sum_{i=1}^{n}\sum_{j=0}^{i-1} a_j c_i. \quad (1)$$

Since according to the restrictions imposed on the parameters of the cutting method we have

$$A > 1, \quad \sum_{i=1}^{n} c_i \le C.$$

From Eq. (1) it follows that

$$(A-1)\left(C - \sum_{i=1}^{n} c_i\right) + \sum_{i=1}^{n}\sum_{j=0}^{i-1} a_j c_i \ge \sum_{i=1}^{n}\sum_{j=0}^{i-1} a_j c_i \ge 0.$$

3. EXPERIMENTAL TESTS

We experimentally studied the properties and efficiency of the multi-start method with cutting and compared it with the multi-start method when training the feed-forward neural networks. We used the criteria of the methods accuracy and computational complexity.

3.1. Problem of Neural Network Training

A triad $NET = (G, T, W)$ will be called a neural network where

• G is an oriented acyclic graph defining a topology of the neural network: its vertexes are the neurons and its arcs are the interneuronal connections.

• T is a set of neural network transition functions

• W is a set of weight vectors of the neural network.

The first two components of the triad define the architecture of the neural network.

After attributing an input set of weight coefficients to the neural network with the given architecture, we say that its initialization takes place.

A finite set $S = \{(X_i, Y_i)\}, i = 1, 2, \ldots, P$ of the pairs the arguments – the value of the approximated function" where $X_i \in R^N$, $Y_i \in R^M$, N is the number of the input neurons (the number of the function's arguments), and M is the number of the output neurons we will call a sample. A teaching sample is a sample for which for any i, j: $i \ne j$ the equality $X_i \ne X_j$ is fulfilled.

The problem of the neural network training is to select the values of the weights of the neural network with the given architecture, which maximize the algorithm efficiency when solving an assigned applied problem. In the present paper, we analyze a special case when we train the neural network for approxima-

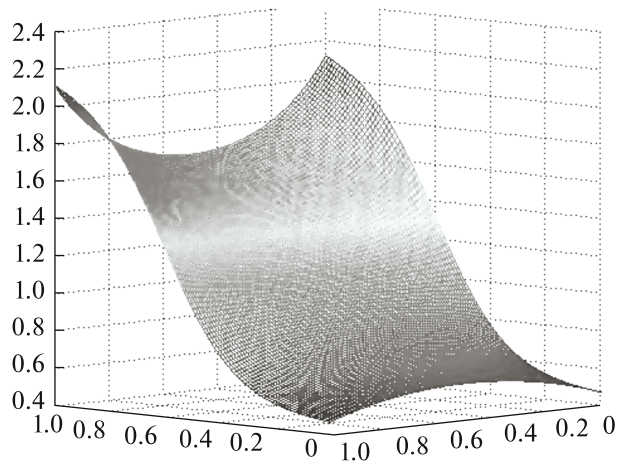

Fig. 1. Function $f_1 = \exp\left(\frac{x}{\sin^2 x + \cos^2 y}\right)$.

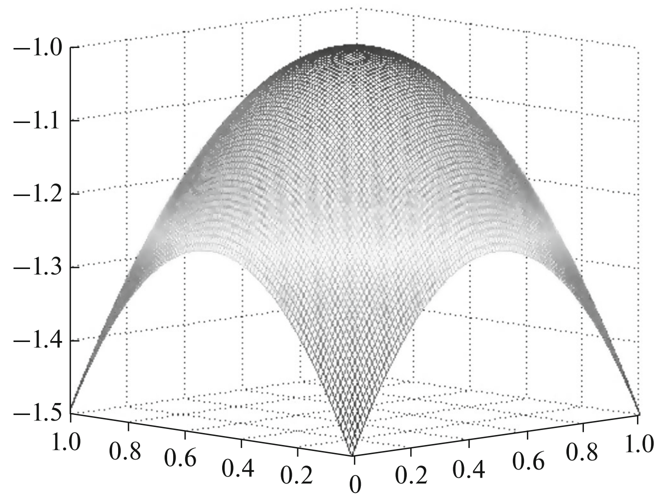

Fig. 2. Function $f_3 = -x^2 - y^2 - 1$.

tion of a continuous table function. To solve this problem, we use the following criteria of the algorithm efficiency:

(1) the accuracy of the approximation provided by the neural network;

(2) the neural network training time.

The approximation accuracy [12] provided by the neural network is defined by the mean-square-root approximation error $MSE(NET, S)$ calculated on a given correct sample for the approximated function.

3.2. Experimental Results

In our computer experiments with the above-described algorithm, we used continuous real functions of two variables defined on the set [0, 1] × [0, 1]. The functions were of different types that is with different number of extremums, different "complexity" and "uniformity" of their reliefs. To obtain a common region of definition for each function we performed a linear transformation of coordinates according the formulas $x = \alpha\left(x_0 - \frac{1}{2}\right), y = \alpha\left(y_0 - \frac{1}{2}\right)$, where $x_0 \in [0, 1]$, $y_0 \in [0, 1]$, and α is a scaling factor chosen separately for each function. Below we list of the selected functions:

1. $f_1 = \exp\left(\frac{x}{\sin^2 x + \cos^2 y}\right)$, the scaling coefficient is 1.5;
2. $f_2 = \frac{1}{1 + e^{-(x+\sin^2 y)}}$, the scaling coefficient is 5;
3. $f_3 = -x^2 - y^2 - 1$, the scaling coefficient is 1;
4. $f_4 = 1 + \frac{1}{1 - e^{x^2+y^2+1}}$, the scaling coefficient is 3;
5. $f_5 = \sin x \sin y$, the scaling coefficient is 3;
6. $f_6 = xe^{\cos y}$, the scaling coefficient is 10;
7. $f_7 = \frac{1}{5}\sin(x^2 + y^2)$, the scaling coefficient is 6;
8. $f_8 = \sin^2 x + \cos^2 y$, the scaling coefficient is 7;
9. $f_9 = \cos\left(\frac{1}{1 + y^2}\right)e^{\frac{y}{10}\sin x} + e^{\frac{x}{10}\cos^2 y}$, the scaling coefficient is 15;
10. $f_{10} = \frac{2}{(x-1)^2 + y^2 + 1} + \sin e^{\frac{x}{2}}$, the scaling coefficient is 10.

We can divide these functions into three groups having in mind “a complexity” of their relief:

• f_1 and f_2 belong to the first group (simple relief);

• f_3, f_4, f_5, and f_6 belong to the second group (more complex relief with extremum);

• f_7, f_8, f_9, and f_{10} belong to the third group (multiextremal and nonuniform relief).

In Appendix 1, we show some examples of the graphs of the functions from each group.

For each of the examined functions we generated a sample of 441 measurements at the points that were the nods of the lattice with the step equal to 0.1 over the region $[0, 1] \times [0, 1]$. We used a half of the generated sample as a training set and the remaining half was a validation set.

We used the following parameters of the multi-start algorithm with cutting:

• the neural network architecture was a two layer perceptron [12] with 10 neurons in each inner layer;

• the number of the first initializations (starts) was $A = 30$;

• the Levenberg–Marquardt algorithm [5] was the local optimal algorithm.

When performing the simulations we found how the cutting procedure influence the computational complexity of the training algorithm. In the course of our experiment, the measurements showed that when running the algorithm uses the processor time n the following way:

(1) On average, about 99.88% of the time the algorithm spends on the running of the local optimal Levenberg–Marquardt training algorithm;

(2) On average, about 0.02% of the time the algorithm spends on the three step cutting procedure;

(3) On average, about 0.01% of the time the algorithm spends on the initialization of the weight coefficients of the learning starts.

Since the most part of the algorithm’s running time of is the running time of the local optimal learning algorithm, in what follows we estimate the algorithm complexity by the number of steps of the local optimalalgorithm (the number of the training steps).

When performing our experiments the number of the total steps of both the multi-start algorithms with cutting and without cutting was the same and it was equal to 1500. The number of starts was also the same and equal to 30. We chose the cutting scheme with the number of training steps equal to 1500 with the maximal number of the training steps per one start equal to 65. The multi-start algorithm without cutting includes 50 training steps per start. We compared the mean-square-root approximation errors obtained when using different cutting schemes and the algorithm without cutting.

Table 1. Comparison between different algorithms

Function type	Number of cutting steps		
	2	3	4
First type	53.45%	55.23%	95.74%
Second type	34.69%	−16.18%	55.49%
Third type	73.04%	20.45%	93.81%
All functions	57.33%	19.83%	81.68%

Table 1 shows how the approximation error (in percent) decreases when we use the cutting schemes with different number of steps comparing with the multi-start algorithm without cutting. We set the error of the algorithm without cutting equal to 100%.

We determined some general properties of the schemes. Below listed the properties:

• on average the number of steps performed after all the starts prior the first cutting is not less than 25 per start and they make up about 38% of the total number of steps per start;

• the number of steps necessary to train (to adjust) the resulting neural network after the last cutting does not exceed 10 and this number is about 15% of the total number of steps;

• at the first cutting step the algorithm cuts off at least 5 starts and that is about 17% of the total number of starts.

4. CONCLUSIONS

The proposed multi-start method with dynamic cutting of the "unpromising" starts of local optimal algorithms is universal when solving continuous unconditional optimization problems. When we examined experimentally the applicability of our method for training of the feed-forward neural networks and compared it with the multi-start method using the criteria of accuracy and computational complexity, we showed that our algorithm was much more efficient.

Using this method for solving other types of the math programming problems for which there are local optimal algorithms it would be necessary to develop new criteria corresponding to the given type of the problem. The chosen criteria have to be suitable when estimating if the start is "promising".

The method is well suited for implementation on multiprocessor systems since the local optimal algorithms and their starts (groups of starts) constitute a main part of their computational complexity. These algorithms can run independently on different processors and the exchange rate between them is low.

FUNDING

The work was done with the financial support of RBRF (grant no. 19-07-00614).

CONFLICT OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

1. Rumelhart, D., Hinton, G., and Williams, R., Learning internal representation by error propagation, in *Parallel Distributed Processing,* Cambridge, MA: MIT Press, 1986, vol. 1, pp. 318–362.
2. Parker, D., Learning logic, in *Invention Report S81-64, File 1,* Stanford, CA: Stanford University, 1982.
3. Werbos, P., Beyond regression: New tools for prediction and analysis in the behavioral sciences, *Master's Thesis,* Harvard University, 1974.
4. Bartsev, S.I. and Okhonin, V.A., Adaptive information processing networks, in *Preprint IF SB USSR,* Krasnoyarsk, 1986, no. 59.
5. Hagan, M. and Menhaj, M., Training feedforward networks with the Marquardt algorithm, *IEEE Trans. Neural Networks,* 1994, vol. 5, no. 6, pp. 989–993.
6. Zhiglyavskii, A.A. and Zhilinskas, A.G., *Metody poiska global'nogo ekstremuma* (Methods for Global Extremum Search), Moscow: Nauka, 1991.

7. Markin, M.I., The choice of the initial approximation in training a neural network using local optimization methods, *Proceedings of the Second All-Russian Scientific and Technical Conference Neuroinformatics-2000,* Moscow, 2000, part 1.
8. Markin, M.I., About one method of increasing the efficiency of learning a direct distribution neural network, *Software Systems and Tools: Thematic Collection of the VMiK Faculty of Lomonosov Moscow State University,* Moscow, 2000, no. 1, pp. 87–97.
9. Nguyen, D. and Widrow, B., 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,* 1990, vol. 3, pp. 21–26.
10. Osowski, S., New approach to selection of initial values of weights in neural function approximation, *Electron. Lett.,* 1993, vol. 29, pp. 313–315.
11. Yam, J.Y.F. and Chow, T.W.S., A weight initialization method for improving training speed in feedforward neural network, *Neurocomputing,* 2000, vol. 30, no. 1, pp. 219–232.
12. Wasserman, P.D., *Neural Computing: Theory and Practice,* Coriolis Group, 1989.
13. Fahlman, S., Faster-learning variations on back-propagation: An empirical study, *Proceedings of the 1988 Connectionist Models Summer School,* 1989, pp. 38–51.
14. Riedmiller, M. and Braun, H., A direct adaptive method for faster backpropagation learning: The RPROP algorithm, *Proceedings of the IEEE International Conference on Neural Networks 1993,* San Francisco, 1993.
15. Minoux, M., *Mathematical Programming: Theory and Algorithms,* Wiley, 1986.