Fundamentals of optimization of training algorithms for artificial neural networks

P.A. Kornev^{1,*} and A.N. Pylkin²

¹Finance, Information, Technology, 11a, 1st Khvostov lane, 119180, Moscow, Russia

²Ryazan State Radio Engineering University named after V.F.Utkin, 59/1, Gagarina st., 390005, Ryazan, Russia

Abstract. In the modern IT industry, the basis for the nearest progress is artificial intelligence technologies and, in particular, artificial neuron systems. The so-called neural networks are constantly being improved within the framework of their many learning algorithms for a wide range of tasks. In the paper, a class of approximation problems is distinguished as one of the most common classes of problems in artificial intelligence systems. The aim of the paper is to study the most recommended learning algorithms, select the most optimal one and find ways to improve it according to various characteristics. Several of the most commonly used learning algorithms for approximation are considered. In the course of computational experiments, the most advantageous aspects of all the presented algorithms are revealed. A method is proposed for improving the computational characteristics of the algorithms under study.

1 Introduction

Modern digital life is unthinkable without artificial intelligence technologies. Inanimate helpers can already speak to us in different voices and recognize our speech and appearance. They can advise us on probable good purchases or customize our exercise schedule based on our digitized analyzes. Thanks to robotics, they become completely humanoid entities with their own complex behavior and digital intelligence. An artificial brain as a complex system consists of tiny interconnected objects, often called artificial neurons. Researchers have already taught such neurons to organize themselves into artificial neural networks in the middle of the 20th century. Any neural network (system) needs training, otherwise the numerical parameters with which it was originally created will not correspond to the exact desired solution to the problem.

To date, several dozens algorithms for training artificial neural systems have been developed. Each of these algorithms was created individually or became a logical development of an existing one. Scientists are still looking for ways to improve the algorithmic base for training artificial neural networks [1-5]. It is proposed to consider one of such methods in more detail in this work.

* Corresponding author: k-p-al@yandex.ru

2 Theory

All the set of existing training algorithms for artificial neural networks (ANN) can be conventionally represented as $A_{NN} = A_{NN1} \cup \dots A_{NNi} \dots \cup A_{NNI}$, where $i \in [1; I]$ – computational task type identifier, A_{NNi} – a group of algorithms used mainly for the i -th type of problems. To improve the computational capabilities of the algorithm, it is necessary to single out some typical problem and consider in detail the model of its solution.

As a test model, consider a model for solving the approximation problem. Given matrices X, T with dimension $M \times N$. It is necessary to calculate the dependence as accurately and quickly as possible $T = f(X)$. Such calculations consist in building an artificial neural network that performs the basic transformation in the form:

$$T = f_1(X) \Rightarrow Y = f_2(W, X) \quad (1)$$

where X – input data vector, T – target data vector, Y – output data vector, W – settings vector ANN.

For solving this problem, it is recommended to use three standard training algorithms:

- 1) Levenberg-Marquardt algorithm [1];
- 2) Bayesian regularization algorithm [1];
- 3) Conjugate gradient algorithm [1].

Let's consider each of them in more detail. Let's accept W_i as a variable matrix of parameters describing the state of ANN at the next step of calculations. Then the basic variants of the algorithms will take the following form.

2.1 Algorithm 1

Step 0. Choose an arbitrary starting point W_1 . Calculate the error functional $E_1 = E(W_1)$. If $E_1 = 0$, then W_1 – the desired solution, else go to step 1.

Step 1. Calculate $dW_1 = -[J^T J + \mu I]^{-1} \cdot J^T E_1$, where J is the Jacobian of the function $f(W, X)$, I – the unit matrix, E_1 – error vector, a μ – selectable setting parameter [1]. Calculate $W_2 = W_1 + dW_1$. If $E_2 = 0$, then W_2 – the desired solution, else go to step k.

Step k. Calculate $dW_k = -[J^T J + \mu I]^{-1} \cdot J^T E_k$. Calculate $W_{k+1} = W_k + dW_k$. If $W_{k+1} \approx 0$, then W_{k+1} – the desired solution.

2.2 Algorithm 2

Step 0. Choose an arbitrary starting point W_1 . Calculate the error functional $E_1 = E(W_1)$. If $E_1 = 0$, then W_1 – the desired solution, else go to step 1.

Step 1. Calculate $dW_1 = -[\beta \cdot J^T J + (\mu + \alpha)I]^{-1} \cdot (\beta \cdot J^T E_1)$, where J – the Jacobian of the function $f(W, X)$, I – the unit matrix, E_1 – error vector, a α, β, μ – selectable setting parameters [1]. Calculate $W_2 = W_1 + dW_1$. If $E_2 = 0$, then W_2 – the desired solution, else go to step k.

Step k. Calculate $dW_{k+1} = -[\beta \cdot J^T J + (\mu + \alpha)I]^{-1} \cdot (\beta \cdot J^T E_1 + \alpha \cdot dW_k)$. Calculate $W_{k+1} = W_k + dW_k$. If $E_{k+1} = 0$, then W_{k+1} – the desired solution.

2.3 Algorithm 3

Step 0. Choose an arbitrary starting point W_1 . Calculate error gradient $G_1 = -\nabla E(W_1)$. If $G_1 = 0$, then W_1 – the desired solution, else go to step 1.

Step 1. Calculate $dW_1 = G_1$. Calculate $W_2 = W_1 + dW_1$. If $E_2 = 0$, then W_2 – the desired solution, else go to step k.

Step k. Calculate $dW_k = G_k + b \cdot dW_{k-1}$. Calculate $W_{k+1} = W_k + a \cdot dW_k$. If $W_{k+1} \approx 0$, then W_{k+1} – the desired solution.

3 Experiment

Consider a way to optimize the operation of artificial neural networks using a standard example of approximation of analytical dependence in the MATLAB package [1-4]. The input and output data are matrices X, T with various configurations. Dependence needs to be identified as $Y = f_2(W, X)$.

To solve the problem, in the standard case, an artificial neural network with the following layers will be built:

- 1) one input layer;
- 2) one hidden layer containing 10 neurons;
- 3) one output layer.

To obtain the correct parameters, the newly created ANN should be trained for each of the three above algorithms. Here are typical results of solving this problem for each of the algorithms for three different data sets.

After performing training according to the first algorithm with standard parameters, the following results were obtained (Table 1).

Table 1. Computational characteristics of standard algorithm trainlm.

Data set	Characteristic	Algorithm 1
Data set 1	mse	2.0935×10^{-4}
	gradient	0.0039
	epochs	16
	time	0.7260
	$Y = R \cdot T + d$	$Y = T + 0.0034$
Data set 2	mse	4.216
	gradient	2.9255
	epochs	11
	time	0.3309
	$Y = R \cdot T + d$	$Y = 0.57T + 4.2$
Data set 3	mse	0.0023
	gradient	0.0020
	epochs	71
	time	2.5623
	$Y = R \cdot T + d$	$Y = 0.85T + 0.061$

On the first dataset, training was completed in 16 iterations. The total training time (time) on the bench computer was 0,726 seconds. The error functional (EF) was taken in the form of the mean square error (performance = mse) and amounted to $2,0935 \times 10^{-4}$. The gradient value was 0,0039. The EF-value did not improve on the validation set during the last 6 training epochs, that is, the most favorable values of the coefficients W, b were obtained after the 10th era of study. This element of learning can be carefully examined according to a special schedule (Figure 1). To study the distribution of the error by characteristic values,

you can use the histogram of errors (Figure 2). When constructing a histogram, the 20 most common difference values are selected $e_i = t_i - y_i$. For an approximate determination of the dependence $Y = \lambda(T)$ apply linear regression plot analysis (Figure 3). In the case under consideration, the dependence is obtained in the form $Y = R \cdot T + d \Rightarrow Y = 1 \cdot T + 0,0034$. To correctly assess the contribution of all points to the construction of the approximation curve, the construction of the resulting graph is used (Figure 4).

Fig. 1. Graph for determining the best validation EF (trainlm).

Fig. 2. Histogram of errors of ANN (trainlm).

Fig. 3. Linear Regression Plot (trainlm).

Fig. 4. Graph of approximation of dependence (trainlm).

Let us next consider the main computational results of the Bayesian regularization algorithm. The main indicators of this algorithm are presented in the following form (Table 2).

Table 2. Computational characteristics of standard algorithm trainbr.

Data set	Characteristic	Algorithm 2
Data set 1	mse	6,6769x10 ⁻¹²
	gradient	2,0952x10 ⁻⁷
	epochs	1000
	time	2,5814
	$Y = R \cdot T + d$	$Y=T+1,8x10^{-7}$
Data set 2	mse	4,0395
	gradient	0,0207
	epochs	607
	time	6,4729
	$Y = R \cdot T + d$	$Y=0,61T+3,8$
Data set 3	mse	0,0022
	gradient	8,5468x10 ⁻⁶
	epochs	186
	time	6,9964
	$Y = R \cdot T + d$	$Y=0,86T+0,058$

Let us conclude our consideration of the standard ANN tuning capabilities by analyzing the results of the conjugate gradient algorithm. The main performance indicators of this algorithm are summarized in Table 3.

Table 3. Computational characteristics of standard algorithm trainsg.

Data set	Characteristic	Algorithm 3
Data set 1	mse	0,1962
	gradient	0,0039
	epochs	19
	time	0,5532
	$Y = R \cdot T + d$	$Y=0,98T+0,27$
Data set 2	mse	4,5098
	gradient	2,9255
	epochs	81
	time	5,4834
	$Y = R \cdot T + d$	$Y=0,57T+0,43$
Data set 3	mse	0,0029
	gradient	0,0007
	epochs	111
	time	0,4512
	$Y = R \cdot T + d$	$Y=0,82T+0,073$

Examining all the above tables and graphs, one can come to a conclusion. Algorithm №2 is slower than Algorithm №1, but more accurate as a result. Algorithm №3 is faster than Algorithm №1, but less accurate as a result.

Thus, we can conclude that the Levenberg-Marquardt algorithm is currently considered the most optimal for solving such problems. Next, you can start modifying the selected most winning algorithm. Let us apply the decomposition-superposition method to optimize the Levenberg-Marquardt algorithm model with the possible use of the following modifications:

- 1) changing the base coefficients;
- 2) introduction of scaling factors.

Consider the formula for the main computational step in the form of a decomposition into individual terms and factors. For example, it can be done like this:

$$dWB = \begin{cases} g_1(J^T J), \\ g_2(\mu I), \\ g_3(J^T E). \end{cases} \tag{2}$$

It is necessary to perform certain analytical transformations in order to perform the desired optimization and possibly obtain some superposition of one or several functions $g_{1-3}(\ast)$. In the course of the experiments, various variants of superpositions were obtained. Variations of the function were chosen as one of the promising directions $g_2(\ast)$:

$$g_2(\ast) = f(\mu) \cdot I = I \cdot (k_1\mu^n + \dots + k_n\mu^1 + k_{n+1}\mu^0). \tag{3}$$

In particular, the following modification of the function was obtained:

$$g_2(\mu) = f(\mu) \cdot I = I \cdot \left(\mu^{\frac{1}{p}}\right). \tag{4}$$

where the control parameter p is given as $2 \leq p \leq 5$.

3.1 Algorithm 4 (modified)

Step 0. Choose an arbitrary starting point W_1 . Calculate the error functional $E_1 = E(W_1)$. If $E_1 = 0$, then W_1 – the desired solution, else go to step 1.

Table 4. Computational characteristics of standard algorithm trainlm1mod.

Data set	Characteristic	Algorithm 4
Data set 1	mse	6,6402x10 ⁻⁶
	gradient	0,0004
	epochs	99
	time	0,3509
	$Y = R \cdot T + d$	$Y=0,98T+0,27$
Data set 2	mse	4,1997
	gradient	0,3978
	epochs	17
	time	0,1950
	$Y = R \cdot T + d$	$Y=0,57T+0,43$
Data set 3	mse	0,0023
	gradient	0,0019
	epochs	35
	time	1,3431
	$Y = R \cdot T + d$	$Y=0,82T+0,073$

Step 1. Calculate $dW_1 = -[J^T J + \mu I]^{-1} \cdot J^T E_1$, where J – jacobian of function $f(W, X)$, I – identity matrix, E_1 – error vector, a μ – selectable setting parameter [1]. Calculate $W_2 = W_1 + dW_1$. If $E_2 = 0$, then W_2 – the desired solution, else go to step k.

Step k. Calculate $dW_k = -\left[J^T J + \mu^{\frac{1}{p}} I\right]^{-1} \cdot J^T E_k$, where $2 \leq p \leq 5$. Calculate $W_{k+1} = W_k + dW_k$. If $W_{k+1} \approx 0$, then W_{k+1} – the desired solution.

This modification (Algorithm 4) gives a certain gain in the accuracy of ANN calculations while maintaining the minimum training time. These facts are confirmed by the following data (Table 4).

Let's consider all the main graphical tools for analyzing the performance of the new algorithm (Figure 5-8).

All main characteristics of the new algorithm are either equivalent to the characteristics of standard algorithms or exceed them by 5-10%.

Fig. 5. Graph for determining the best validation EF (trainlm1mod).

Fig. 6. Histogram of errors of ANN (trainlm1mod).

Fig. 7. Linear Regression Plot (trainlm1mod).

Fig. 8. Graph of approximation of dependence (trainlm1mod).

4 Conclusion

The article discusses the most commonly used algorithms for training artificial neural networks to approximate analytical dependencies of varying degrees of complexity. Much attention is paid to the consideration of the features of calculations at all stages of training. The parameters of the quality of training are analyzed to identify the most probable lines of optimization of algorithms. To improve such an analysis, a specific method is proposed, focused both on the decomposition of large blocks of the algorithm and on improving the computational characteristics of each step of the algorithm. The above method can be considered promising for the computational optimization of learning algorithms for modern ANNs used to solve problems of approximation of any degree of complexity.

The article discusses the most commonly used algorithms for training artificial neural networks to approximate analytical dependencies of varying degrees of complexity. The parameters of the quality of training are analyzed to identify the most probable lines of optimization of algorithms. To improve such an analysis, a specific method is proposed, focused both on the decomposition of large blocks of the algorithm and on improving the computational characteristics of each step of the algorithm. The above method can be considered promising for computational optimization of modern ANN training algorithms used to solve approximation problems of any degree of complexity.

References

1. Haykin S 2019 *Neural networks* (M.: I.D. Williams) p 1104
2. 2017 *Neural networks in MATLAB* (B.S.T.U. SPb) p 165
3. Parkhomenko S S, Ledenyov T M 2014 *VGU bulletin, Series: system analysis and information technologies* **2** 98
4. Filippov F V 2019 *Modeling of neural networks of deep training* (SPb) p 79
5. Izmailov A F, Solodov M V, Uskov E I 2019 *Computational Optimization and Applications* **72** 215
6. Vasin V V 2012 *Automation and Remote Control* **73** 440
7. Kenji Ueda, Nobuo Yamashita 2012 *Journal of Optimization Theory and Applications* **152** 450
8. Hanke Martin 2010 *J. Integral Equations Applications* **22** 259
9. Iqbal J, Iqbal A, Arif M 2015 *Journal of Computational and Applied Mathematics* **282** 134
10. Jiann-Ming Wu 2008 *IEEE Transactions on Neural Networks* **19** 2032
11. Suliman A, Omarov B 2018 *International Journal of Engineering and Technology(uae)* **7** 1194
12. Bilski J, Kowalczyk B, Grzanek K 2018 *Lecture Notes in Computer Science* **10841** LNAI 15
13. Sakamoto H, Matsumoto K, Kuwahara A, Hayami Y 2005 *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* **E88-A** 1971
14. Levenberg K 1944 *The Quarterly of Applied Mathematics* **2** 164
15. Marquardt D W 1963 *Journal of the Society for Industrial and Applied Mathematics* **11(2)** 431
16. Kornev P A 2020 *Control systems and information technologies* **1(79)** 36