Comparing the efficiency of serial and parallel algorithms for training artificial neural networks using computer clusters

Oleg V. Kryuchin
Alexander A. Arzamastsev
Klaus G. Troitzsch

Nr. 13/2011

Arbeitsberichte aus dem Fachbereich Informatik

Arbeitsberichte des Fachbereichs Informatik

ISSN (Print): 1864-0346
ISSN (Online): 1864-0850

Herausgeber / Edited by:

Der Dekan:
Prof. Dr. Grimm

Kontaktdaten der Verfasser

Oleg V. Kryuchin, Alexander A. Arzamastsev, Klaus G. Troitzsch,
Institut für Wirtschafts- und Verwaltungsinformatik
Fachbereich Informatik
Universität Koblenz-Landau
Universitätsstraße 1
D-56070 Koblenz
Email : kryuchov@gmail.com, arz_sci@mail.ru, kgt@uni-koblenz.de

Comparing the efficiency of serial and parallel algorithms for training artificial neural networks using computer clusters

Oleg V. Kryuchin, Alexander A. Arzamastsev, Klaus G. Troitzsch

Abstract

An estimation of the number of multiplication and addition operations for training artificial neural networks by means of consecutive and parallel algorithms on a computer cluster is carried out. The evaluation of the efficiency of these algorithms is developed. The multilayer perceptron, the Volterra network and the cascade-correlation network are used as structures of artificial neural networks. Different methods of non-linear programming such as gradient and non-gradient methods are used for the calculation of the weight coefficients.

Keywords: parallel algorithms, artificial neural networks, estimation of algorithm efficiency, computer clusters.

1 Introduction

1.1 Background and aims

Nowadays artificial neural networks (ANNs) are used in different branches of science and engineering, such as systems of artificial intelligence, image identification tasks, mathematical simulation (in situations when it is necessary to develop a model of an object when the structure of this object cannot be easily defined by simple rules of this science branch), computer vision systems, analysis and forecast of time series, simulation of social objects etc. [1, 2, 3].

Thus an ANN is a universal instrument for building mathematical models for objects which are defined by empirical data. The main limitations of ANNs lie in the compilation of algorithms for structure selection and in the large size of the vector of variables of the target function which is to be minimized by the algorithm training the ANN. These limitations define that the development of an ANN-model which is sufficient to model a certain object may take a large amount of time.

One of the most advanced methods of solving this problem is the development of parallel algorithms of building an ANN-model using computer clusters [4, 5, 6]. The traditional techniques of estimating the speed of algorithms count the multiplication and addition operations executed by these algorithms. So the aim of this paper is the calculation of the numbers of multiplication and addition operations for serial and parallel algorithms training the ANN. These numbers may be used for examining parallel algorithms .

1.2 Used network structure types

This subsection will consider a few types of ANNs. These are the multilayer perceptron (MLP) shown in Figure 1, a cascade-correlation network shown in Figures 2–5) and a Volterra network shown in Figure 6. These structure types were selected because they are very often used for the approximation and the forecasting of time series [3, 7].

A multilayer perceptron (MLP) is a feedforward ANN model that maps sets of input data into a set of appropriate output. An MLP consists of multiple layers of nodes in a directed graph which is fully connected from one layer to the next. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function. Usually an MLP utilizes a supervised learning technique called backpropagation for training the network but it may also use other algorithms [8, 9]. The MLP is a modification of the standard linear perceptron, which can distinguish data which are not linearly separable [10].

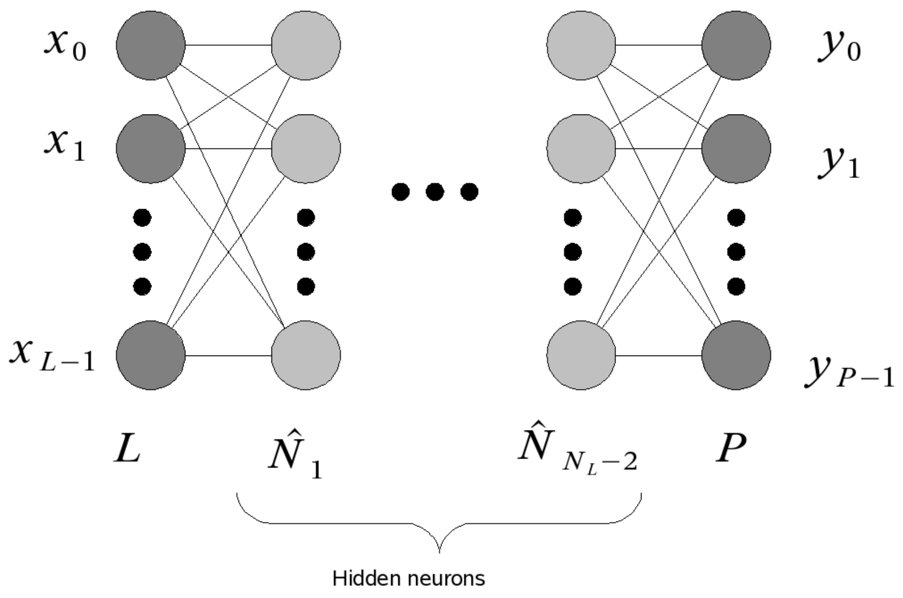

Figure 1: The multilayer perceptron.

In Figure 1 $\vec{x}$ is the vector of input data and $\vec{y}$ is the output data vector. This network has N_L layers. The first layer is the input layer and consists of L neurons. The last layer is the output layer and consists of P neurons. The other layers are hidden and the i-th hidden layer consists of $\hat{N}_i$ neurons.

A cascade-correlation network is a particularized multilayer neural network construction which was recommended by Fahlman. In this network the selection of the structure and the search for weight coefficients are executed in parallel. In each training step one new hidden neuron is added. So training such network always consists of the selection of the structure and of updating weight coefficients (and sometimes searching neuron activation functions). The architecture of the cascade-correlation network is the consolidation of neurons by links, and the view of this consolidation is the progressive cascade. Each added neuron connects to output nodes and to all hidden neurons. All input neurons and hidden neurons connect to each output neuron, too [11, 12].

The training of this network begins before hidden neurons are added. Weight coefficients (and activation functions) are searched for minimizing the target function value (Fig. 2).

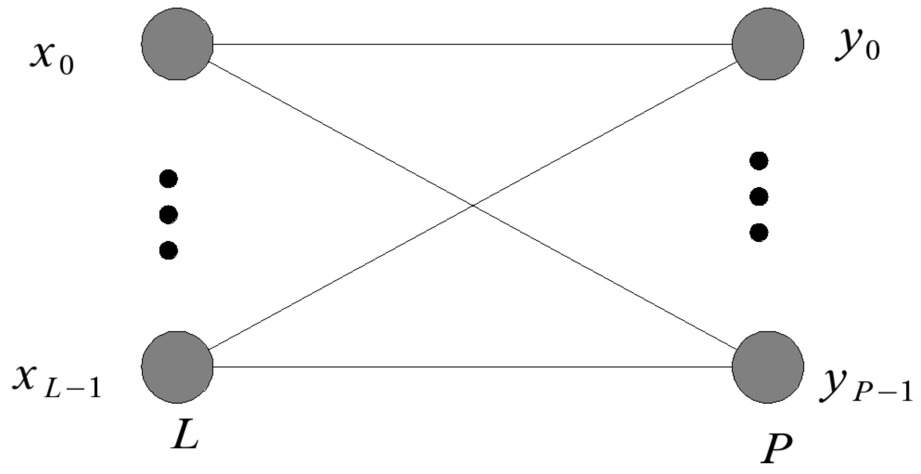

Figure 2: The cascade-correlation network without hidden neurons.

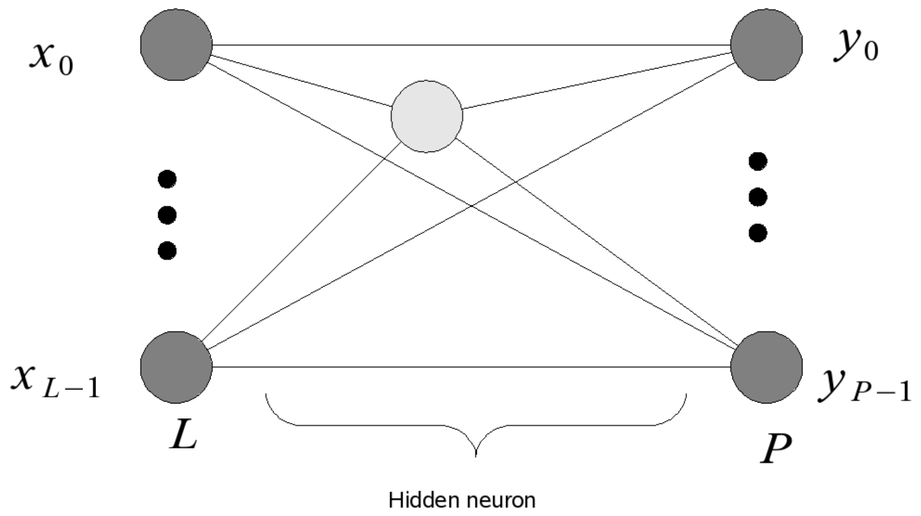

Figure 3: The cascade-correlation network with one hidden neuron.

If after training a result is considered reasonable with respect to the feasible or desired accuracy then the process of the training and the network structure formed finishes. Otherwise it is necessary to add a new hidden neuron (Fig. 3). For this a special procedure is used. At first input weights of the new neuron are formed, initialized, updated and fixed. Then the output weights of this neuron are connected to all output neurons. After this step new weight coefficients are searched. If the result of the training

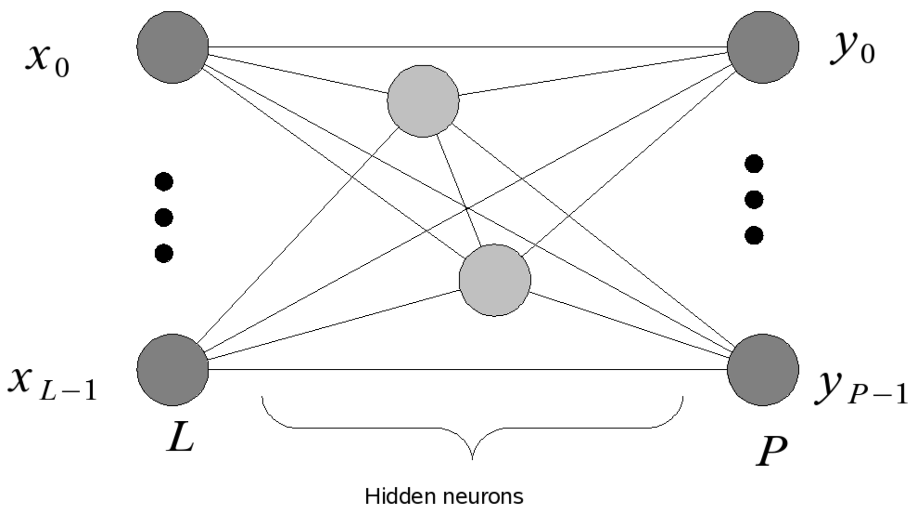

Figure 4: The cascade-correlation network with two hidden neurons.

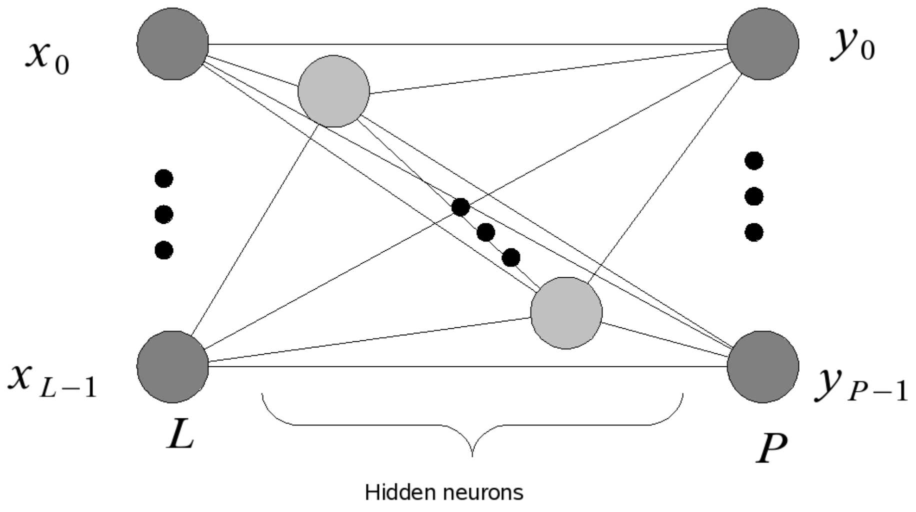

Figure 5: The cascade-correlation network with few hidden neurons.

is reasonable then the training is finished, otherwise again a new neuron is added. This process continues until the desired result of the training is obtained (pic. 4-5) [8, 14].

A Volterra network is a dynamic network for the nonlinear adaptation of an array of signals following each other. The input vector $\vec{x}(t)$ (which is shown in formula (1)) activates the network at moment i. As the Volterra series is defined, the output signal

$y(i)$ is calculated by formula (2).

$$\vec{x}(t) = (x_t, x_{t-1}, \ldots, x_{t-L}) \tag{1}$$

$$y(\iota) = \sum_{i_1=1}^{L}(w_{i_1} x_{\iota-i_1}) + \sum_{i_1=1}^{L}\left(\sum_{i_2=1}^{L}(w_{i_1,i_2} x_{\iota-i_1} x_{\iota-i_2})\right) + \ldots + + \sum_{i_1=1}^{L}\left(\sum_{i_2=1}^{L}\left(\sum_{i_3=1}^{L}\cdots\sum_{i_K=1}^{L}(w_{i_1,i_2,\ldots,i_K} x_{\iota-i_1} x_{\iota-i_2} \ldots x_{\iota-i_K})\right)\right) \tag{2}$$

where weights w_{i_1}, w_{i_1,i_2}, w_{i_1,i_2,i_3}, w_{i_1,i_2,i_3,i_4} etc. are called Volterra kernels and correspond to the reaction of the highest factors. This polynomial degree is called the Volterra series degree [3, 15].

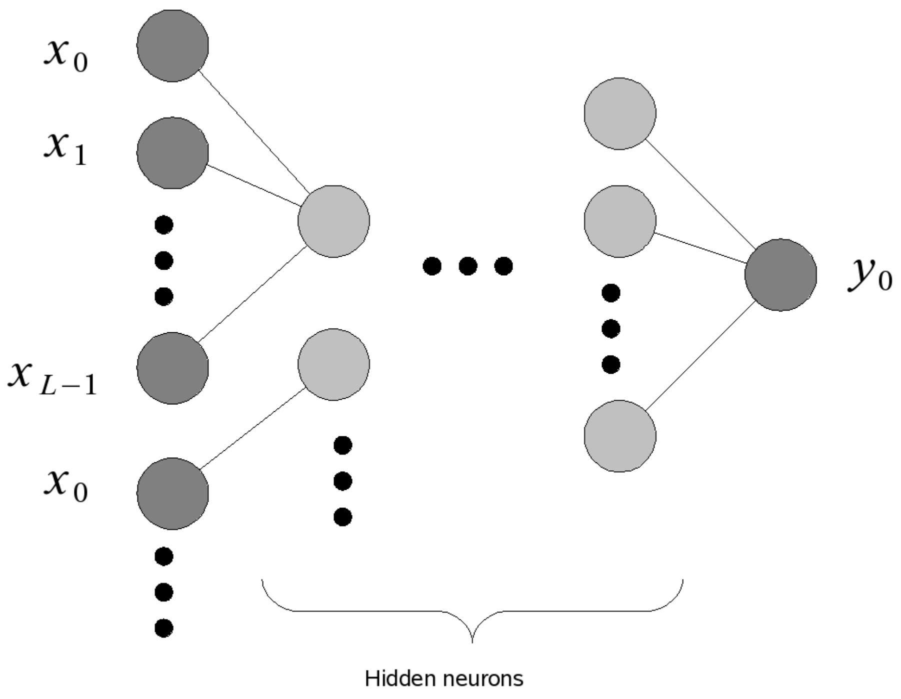

Figure 6: The Volterra network.

2 Number of operations for calculating output values

2.1 Activation functions of neurons

An artificial neuron model is an individual identity element which must sum up signals which enter with associated weight coefficients (this is an analog of the biological prototype of the synaptic force). Afterwards an activation function (which is usually nonlinear) is executed and formulates the output signal of the neuron.

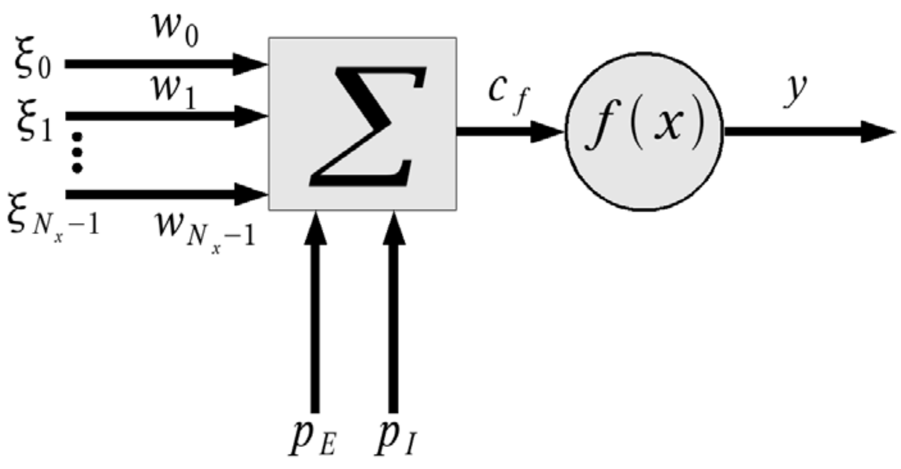

Figure 7: The model of the McCulloch-Pitts neuron.

The model of McCulloch-Pitts was one of the first models of the artificial neuron. For the signal generation it uses the threshold function (3).

$$y = \begin{cases} 1, & x_f > 0; \\ 0, & x_f \leq 0; \end{cases} \tag{3}$$

The function (3) is an activation function. The argument of this function is calculated by formula (4).

$$x_f = \sum_{i=0}^{N_x-1} \xi_i w_i \tag{4}$$

where $\vec{w}$ is the weight coefficients vector, $\vec{\xi}$ is the vector of input signals which has lenght N_x.

The McCulloch-Pitts model with its training strategy is called "the simple perceptron" [16]. There are other activation function flavors. The functions used most often are the sigmoid and the tangent.

As it is shown in figure 7 a general neuron has two impulses (adjugate weight coefficients). One of these impulses (which is called "the internal impulse" (p_I)) is fixed when weight coefficients are trained and the second impulse (which is called "the external impulse" (p_E)) is updated together with the former. The activation function argument has a coefficient and so the neuron output value can be calculated by formula (5).

$$y = f\left(c_f\left(\sum_{i=0}^{N_x-1}(\xi_i w_i) + p_I + p_E\right)\right) \tag{5}$$

The usage of impulses (p_I and p_E) and a coefficient (c_f) allows to customize activation functions flexibly.

It symbolizes a number of multiplication operations which are executed for the calculation of the i-th neuron output value (by formula (5)) as ζ_n, and it symbolizes a addition operations number as $\hat{\zeta}_n$. Values ζ_n and $\hat{\zeta}_n$ of different activation functions are showed in table 1.

Table 1: Numbers of operations which are executed for calculation of output neuron value.

The function flavor	**The number of addition operations ($\hat{\zeta}_n$)**	**The number of multiplication operations (ζ_n)**
$y = c_f x_f$	2	1
$y = c_f^2 x_f^2$	2	2
$y = c_f^3 x_f^3$	2	3
$y = c_f^4 x_f^4$	2	3
$y = sin(c_f x_f)$	2	28
$y = cos(c_f x_f)$	2	28
$y = tan(c_f x_f)$	2	24
$y = ctg(c_f x_f)$	3	24
$y = \frac{1}{1+c_f x_f}$	3	2
$y = \frac{1}{1+e^{c_f x_f}}$	3	12
$y = \frac{1}{1+e^{\lvert c_f x_f \rvert}}$	3	12

2.2 Calculation of output values of different network types

The number of multiplication operations which are necessary for the calculation of the ANN output values depends on the number of multiplication operations which are executed for the calculation of output values of neurons and the number of synapses:

$$\zeta_y = \sum_{i=0}^{l_\mu - 1} \zeta_n(\mu_i) + l_w \tag{6}$$

where l_μ is the number of neurons in the ANN, l_w is the number of synapses (weights). Analogously the number of addition operations which are necessary for the calculation of ANN output values is calculated:

$$\hat{\zeta}_y = \sum_{i=0}^{l_\mu - 1} \hat{\zeta}_N(\mu_i) + l_w + \hat{\zeta}_I \tag{7}$$

where $\hat{\zeta}_I$ is the number of addition operations which are necessary for changing the marker of the obstruction cocycle.

It is possible to modify formulas (6)-(7) for different types of ANNs. So formulas (8)-(9) formulate operations numbers for the MLP:

$$\zeta_{yM} = \sum_{i=0}^{l_\mu - 1} \zeta_n(\mu_i) + \sum_{i=1}^{N_L - 1} \hat{N}_i \hat{N}_{i-1} \tag{8}$$

$$\hat{\zeta}_{yM} = \sum_{i=0}^{l_\mu - 1} \hat{\zeta}_n(\mu_i) + 2 \sum_{i=1}^{N_L - 1} \hat{N}_i \hat{N}_{i-1} + 2N_L + P + L \tag{9}$$

In a similar way one can calculate numbers of operations for the cascade-correlation network (calculated by formulas (10) and (11)) and the Volterra network (calculated by formulas (12) and (13)):

$$\zeta_{yC} = \sum_{i=0}^{l_\mu-1} \zeta_n(\mu_i) + \sum_{i=0}^{\hat{N}_1-1} (\hat{N}_0 + i) + P(\hat{N}_0 + \hat{N}_1) \tag{10}$$

$$\hat{\zeta}_{yC} = \sum_{i=0}^{l_\mu-1} \hat{\zeta}_n(\mu_i) + 2\sum_{i=0}^{\hat{N}_1-1} (\hat{N}_0 + i) + 2P(\hat{N}_0 + \hat{N}_1) + \hat{N}_1 + 2P + L \tag{11}$$

$$\zeta_{yV} = \sum_{i=0}^{l_\mu-1} \zeta_n(\mu_i) \tag{12}$$

$$\hat{\zeta}_{yV} = \sum_{i=0}^{l_\mu-1} \hat{\zeta}_n(\mu_i) + 2\sum_{i=1}^{N_L-1} \hat{N}_i\hat{N}_{i-1} + L(N_L - 1) \tag{13}$$

2.3 Reduction of addition operations to multiplication operations

For reduction of addition operations to multiplication operations the coefficient σ is used. This coefficient value is directly proportional to the time spent for one addition operation and inversely related to the time spent for one multiplication operation. So one multiplication operation needs the time which is necessary for σ addition operations and one addition operation can be changed to σ multiplication operations. So it is possible to formulate equations (14)-(16).

$$\begin{aligned} z_{yM} &= \zeta_{yM} + \sigma\hat{\zeta}_{yM} = \sum_{i=0}^{l_\mu-1} \zeta_n(\mu_i) + \sum_{i=1}^{N_L-1} \hat{N}_i\hat{N}_{i-1} + \\ &\quad + \sigma\left(\sum_{i=0}^{l_\mu-1} \hat{\zeta}_N(\mu_i) + 2\sum_{i=1}^{N_L-1} \hat{N}_i\hat{N}_{i-1} + 2N_L + P + L\right) \end{aligned} \tag{14}$$

$$\begin{aligned} z_{yC} &= \zeta_{yC} + \sigma\hat{\zeta}_{yC} = \sum_{i=0}^{l_\mu-1} \zeta_n(\mu_i) + \sum_{i=0}^{\hat{N}_1-1} (\hat{N}_0 + i) + \\ &\quad + P(\hat{N}_0 + \hat{N}_1) + \sigma\left(\sum_{i=0}^{l_\mu-1} \hat{\zeta}_N(\mu_i) + 2\sum_{i=0}^{\hat{N}_1-1} (\hat{N}_0 + i)\right) + \\ &\quad + \sigma\left(2P(\hat{N}_0 + \hat{N}_1) + \hat{N}_1 + 2P + L\right) \end{aligned} \tag{15}$$

$$\begin{aligned} z_{yV} &= \zeta_{yV} + \sigma\hat{\zeta}_{yV} = \sum_{i=0}^{l_\mu-1} \zeta_n(\mu_i) + \\ &\quad + \sigma\left(\sum_{i=0}^{l_\mu-1} \hat{\zeta}_N(\mu_i) + 2\sum_{i=1}^{N_L-1} \hat{N}_i\hat{N}_{i-1} + L(N_L - 1)\right) \end{aligned} \tag{16}$$

3 Number of operations of algorithms of training

3.1 The general coefficient of the efficiency

The coefficient of efficiency depends on the numbers of operations which are calculated by the serial and the parallel algorithm. Its value is calculated by formula (17):

$$\alpha(Z) = \frac{z}{nZ} \tag{17}$$

where z is the number of operations in the serial algorithm, Z is the number of operations in the parallel algorithm (in the processor which executes the maximum number of operations) and n is the number of processors.

3.2 Level of calculating the value of the target function

The value of the target function is calculated by formula (18). In this equation the calculation of ε for the i-th row of the pattern needs $\zeta_y + 1$ multiplication operations (ζ_y operations for the calculation of ANN output values and one operation for the squaring) and $\hat{\zeta}_y + 2P$ addition operations.

$$\varepsilon = \sum_{i=0}^{N-1} \sum_{j=0}^{P-1} (d_{i,j} - F(\vec{x}_i, \vec{w}, \vec{\mu})_j)^2 \tag{18}$$

where x and d are patterns (input and output) which consist of N rows, $\vec{w}$ is the weights coefficients vector, $\vec{\mu}$ is the vector of activation functions of neurons, F is the function which calculates output values of the ANN, P is the number of output neurons in the ANN.

So for the calculation of the full inaccuracy ε $N(\zeta_y+1)+1$ multiplication and $\hat{\zeta}_y+2P$ addition operations are needed. And it is necessary to execute N addition operations for the summation.

So it is possible to formulate equations (19)-(21) which show the numbers of operations which are executed by the algorithms of calculating the value of the target function.

$$\zeta_\varepsilon = N(\zeta_y + 1) + 1 \tag{19}$$
$$\hat{\zeta}_\varepsilon = N(\hat{\zeta}_y + 2P) \tag{20}$$
$$z_\varepsilon = \zeta_\varepsilon + \sigma\hat{\zeta}_\varepsilon = N(\zeta_y + 1 + \sigma\hat{\zeta}_y + 2\sigma P) + 1 \tag{21}$$

If there are n processors and the pattern consists of N rows then each processor can calculate the inaccuracy for the part of the pattern (equation (23)) for processor number k and equation (22) for the lead processor. Then the zero processor sums up (formula (24)). So the lead processor sends M rows to each processor, calculates the inaccuracy by $\hat{M}$ rows, receives the inaccuracies from other processors and calculates the result. Values of M and $\hat{M}$ are calculated by formulas (25)-(26) [4, 17].

$$\begin{aligned}
\varepsilon_0 &= \sum_{i=0}^{\hat{M}-1}\sum_{j=0}^{P-1}(d_{i,j} - F(\vec{x}_i, \vec{w}, \vec{\mu})_j)^2 && (22)\\
\varepsilon_k &= \sum_{i=0}^{M}\sum_{j=0}^{P-1}(d_{i+(k*M)+\hat{M},j} - F(\vec{x}_{i+(k*M)+\hat{M}}, \vec{w}, \vec{\mu})_j)^2, \quad k > 0 && (23)\\
\varepsilon &= \frac{1}{N}\sum_{k=0}^{n-1}\varepsilon_k && (24)\\
M &= \begin{cases} \left[\frac{N}{n}\right], & N \; mod \; n = 0;\\ \left[\frac{N}{n-1}\right], & N \; mod \; n \neq 0; \end{cases} && (25)\\
\hat{M} &= \begin{cases} M, & N \; mod \; n = 0;\\ N - M(n-1), & N \; mod \; n \neq 0; \end{cases} && (26)
\end{aligned}$$

It is necessary for the parallel algorithm:

- to divide the pattern into n parts and send it to processors before the training;
- to send the vector of weights coefficients $\vec{w}$ to each nonzero processor and receive the inaccuracy value ε_k in each iteration of the calculation of the value of the target function.

For sending or receiving one element one multiplication and two addition operations are necessary. So for sending the pattern which will be worked on by the k-th processor the lead processor executes $M(P+L)$ multiplications and $2M(P+L)$ addition operations and it executes $M(P+L)(n-1)$ multiplications and $2M(P+L)(n-1)$ addition operations for sending patterns to all nonzero processors. Thus the number of pattern elements which are sent is $\hat{P}$ ($\hat{P} = MP + ML$). The segmentation of the pattern into n parts needs two multiplication and N addition operations and thus for accomplishing the first step in the lead processor $2+\hat{P}(n-1)$ multiplication and $N+2\hat{P}(n-1)$ addition operations and $\hat{P}$ multiplication and $2\hat{P}$ addition operations in nonzero processors are needed. The non-lead processors cannot begin to receive data until the lead processor sends it so k-th processor ($k > 0$) waits $2 + k\hat{P}$ multiplication and $N + 2k\hat{P}$ addition operations which are executed by the lead processor for sending patterns to the k-th processor. Thus the k-th processor ($k > 0$) executes $2 + k\hat{P} + \sigma N + 2\sigma k\hat{P}$ vacuous operations which conform with the preparatory operations in the lead processor and $\gamma(\hat{P}, v)$ operations of the sending (the time of making these operations conforms with the time of the interprocessor sending $\hat{P}$ numbers where the time rate of the interconnection is v). So the lead processor executes $C^{(0)}_{\varepsilon 0}$ multiplication operations, and the k-th nonzero processor executes $C^{(0)}_{\varepsilon k}$. The values of $C^{(0)}_{\varepsilon 0}$ and $C^{(0)}_{\varepsilon k}$ are calculated by formulas (27)–(28)).

$$\begin{aligned}
C^{(0)}_{\varepsilon 0} &= 2 + \hat{P}(n-1) + \sigma N + 2\hat{P}\sigma(n-1) && (27)\\
C^{(0)}_{\varepsilon k} &= 2 + k\hat{P} + \sigma N + 2\sigma k\hat{P} + \gamma(\hat{P}, v) + \hat{P} + 2\sigma\hat{P} && (28)
\end{aligned}$$

As figure 8 shows, the parallel calculation of the inaccuracy consists of few steps:

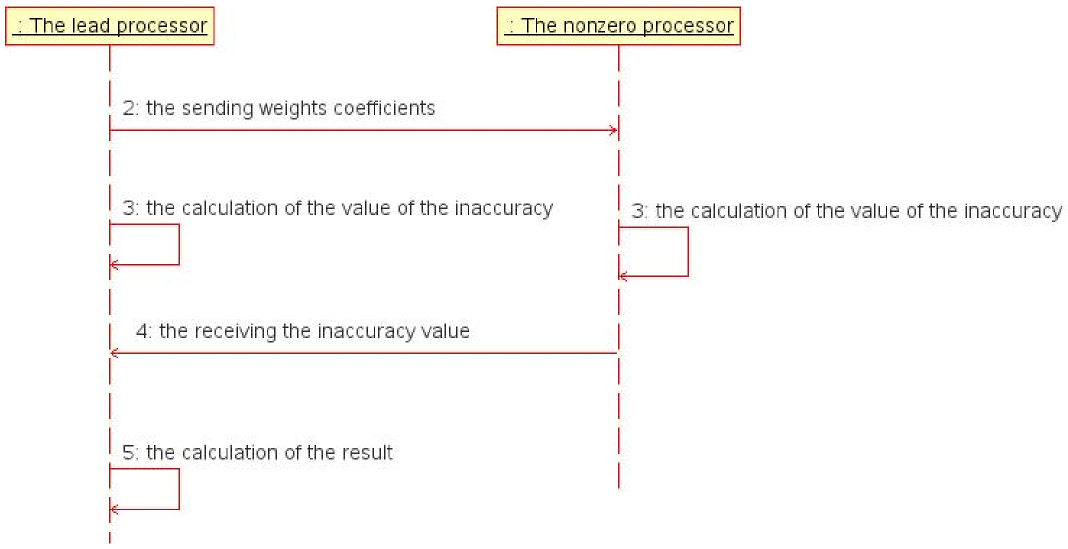
: The lead processor
: The nonzero processor
2: the sending weights coefficients
3: the calculation of the value of the inaccuracy
3: the calculation of the value of the inaccuracy
4: the receiving the inaccuracy value
5: the calculation of the result

Figure 8: The diagram of steps of the parallel calculation of the target function value.

1. sending elements of the vector of weight coefficients (the length of this vector is l_w) to all nonzero processors. For this the lead processor executes $l_w(n-1)$ multiplication and $2l_w(n-1)$ addition operations and the other processors execute l_w multiplication and $2l_w$ addition operations for receiving the data. So the k-th processor executes $C_{\varepsilon k}^{(0)}$ and the lead processor executes $C_{\varepsilon 0}^{(1)}$ operations. The values of $C_{\varepsilon 0}^{(0)}$ and $C_{\varepsilon k}^{(1)}$ are calculated by formulas (29)–(30).

$$C_{\varepsilon 0}^{(1)} = (n-1)(l_w + 2\sigma l_w) \quad (29)$$
$$C_{\varepsilon k}^{(1)} = kl_w + 2\sigma k l_w + \gamma(l_w, v) + l_w + 2\sigma l_w \quad (30)$$

2. calculating the value of the inaccuracy: each processor uses its part of the pattern, so the numbers of operations are calculated by formula (31) for the lead processor and (32) for the nonzero processors.

$$C_{\varepsilon 0}^{(2)} = \hat{M}(\zeta_y + 1 + \sigma\hat{\zeta}_y + 2\sigma P) + 1 \quad (31)$$
$$C_{\varepsilon k}^{(2)} = M(\zeta_y + 1 + \sigma\hat{\zeta}_y + 2\sigma P) + 1 \quad (32)$$

3. receiving the inaccuracy value by the lead processor. For this, the nonzero processors execute one multiplication and two addition operations, and the zero processor executes $n-1$ multiplication and $2n-2$ addition operations for receiving and $1+2\sigma+\gamma(1,v)$ operations for waiting. So the lead processor executes $C_{\varepsilon 0}^{(3)}$ operations (the value of $C_{\varepsilon 0}^{(3)}$ is calculated by formula (33)) and other processors execute $C_{\varepsilon k}^{(3)}$ (this value is calculated by formula (34)).

$$C_{\varepsilon 0}^{(3)} = n - 1 + 2\sigma(n-1) + \gamma(1, v) + 1 + 2\sigma \quad (33)$$
$$C_{\varepsilon k}^{(3)} = 1 + 2\sigma \quad (34)$$

4. calculating the result by the zero processor. For this, it is necessary to execute $n-1$ addition operations and one multiplication operation.

So the number of operations which are necessary for these steps are shown in table 2.

Table 2: Numbers of operations which are necessary for the steps of the parallel calculation of the inaccuracy.

Step	**Zero processor**	**Nonzero (k-th) processor**
1	$l_w(n-1)(1+2\sigma)$	$kl_w + 2\sigma k l_w + \gamma(l_w, v) + l_w + 2\sigma l_w$
2	$\hat{M}(z_y + 1 + 2\sigma P) + 1$	$M(z_y + 1 + 2\sigma P) + 1$
3	$n + 2\sigma n + \gamma(1, v)$	$1 + 2\sigma$
4	$\sigma(n-1) + 1$	

Before the lead processor begins to receive the results of the operations which were calculated by other processors it sends weight coefficients to all other processors and calculates ε_0 (for ¥this it executes $C_{\varepsilon 0}$ operations) and the other processors receive the vector $\vec{w}$, calculate the value of the inaccuracy ε_k and send the result to the lead processor.

$$\begin{aligned} \hat{C}_{\varepsilon 0} &= l_w(n-1)(1+2\sigma) + \hat{M}(\zeta_y + 1 + \sigma\hat{\zeta}_y + 2\sigma P) + 1 && (35) \\ \hat{C}_{\varepsilon k} &= (1+2\sigma)(kl_w + l_w + 1) + \gamma(l_w, v) + && (36) \\ &\quad + M(\zeta_y + 1 + \sigma\hat{\zeta}_y + 2\sigma P) + 1 \end{aligned}$$

It is necessary to consider the time of sending the inaccuracy value ($\gamma(1, v)$). Thus the zero processor can begin to receive data from the k-th processor after $\max\left(\hat{C}_{\varepsilon 0}, \hat{C}_{\varepsilon k} + \gamma(1, v)\right)$ operations. The receiving finishes after the slowest processor has sent its inaccuracy value, and this is why the parallel calculation of the inaccuracy needs Z_ε operations (calculated by formula (37)).

$$\begin{aligned} Z_\varepsilon &= \max_{k=1..n-1}\left(\max\left(\hat{C}_{\varepsilon 0} + k(2\sigma+1), \hat{C}_{\varepsilon k} + \gamma(1, v)\right)\right) + \\ &\quad + \sigma n - 1\sigma + 1 && (37) \end{aligned}$$

So it is possible to conclude (38) which shows the efficiency of the algorithm which executes the parallel calculation of the value of the target function.

$$\alpha_\varepsilon(Z) = \frac{I_\varepsilon z_\varepsilon + \lambda_\varepsilon}{nI_\varepsilon Z_\varepsilon + C_{\varepsilon k}^{(0)} + \lambda_\varepsilon} \qquad (38)$$

where I_ε is the number of operations for the calculation of the inaccuracy, λ_ε is the number of other operations of algorithms (which do not belong to the calculation of the inaccuracy).

3.3 Full enumeration method

This method searches all variants of weight coefficients values. So the value of the i-th weight coefficient in the I-th iteration is calculated by formula (39). The method iterates I_F times (the value of I_F is calculated by formula (41)).

$$w_i^{(I)} = l_0^{down} + s_i \times \left(\left[\frac{I}{\hat{s}(i)} \right] \; mod \; \left(\left[\frac{l_i^{up} - l_i^{down}}{s_i} \right] + 1 \right) \right) \tag{39}$$

$$\hat{s}(i) = \begin{cases} 1, & i = l_w - 1; \\ \prod_{j=i+1}^{l_w-1} \left(\left[\frac{l_j^{up} - l_j^{down}}{s_j} \right] + 1 \right), & i < l_w - 1; \end{cases} \tag{40}$$

$$I_F = \prod_{i=0}^{l_w-1} \left(\left[\frac{l_j^{up} - l_j^{down}}{s_j} \right] + 1 \right) \tag{41}$$

where l_i^{up}, l_i^{down} are the upper and lower limits of the i-th weight coefficient, s_i is the step of the i-th scanning of the weight coefficient, and l_w is the weights count.

It is possible to calculate the number of multiplication operations which are executed by the serial algorithm of the full enumeration by formula (42).

$$z_{wF} = z_\varepsilon I_F + 2\sigma I_F \tag{42}$$

where z_ε is the number of operations which are needed for the calculation of the inaccuracy value.

If there are n processors then each nonzero processor iterates J_F times (formula (43)), and the lead processor iterates $\hat{J}_F$ times (formula (44)) [17].

$$J_F = \begin{cases} \left[\frac{I_F}{n} \right], & I_F \; mod \; n = 0; \\ \left[\frac{I_F}{n-1} \right], & I_F \; mod \; n \neq 0; \end{cases} \tag{43}$$

$$\hat{J}_F = \begin{cases} J_F, & I_F \; mod \; n = 0; \\ I_F - J_F(n-1), & I_F \; mod \; n \neq 0; \end{cases} \tag{44}$$

For the calculation of the number of multiplication operations which are executed by the parallel algorithm of the full enumeration it is necessary to analyse the steps of this algorithm. As figure 9 shows, the parallel algorithm consists of five steps:

1. initialization;
2. sending data from the lead processor to other processors;
3. enumeration of values of weight coefficients which belong to this processor;
4. sending data from all processors to zero processor;
5. selection of the best configuration;

In the first step the lead processor executes $2l_w$ addition operations (l_w for the organization of the obstruction cocycle and l_w for the awarding). So $C_{wF0}^{(1)} = 2\sigma l_w$ and $C_{wFk}^{(1)} = 0$. In the second step the lead processor executes $l_w(n-1)$ multiplication and $2l_w(n-1)$ addition operations. The other processors execute l_w multiplication and $2l_w$ addition operations but they can begin before the zero processor sends data, so the nonzero processors execute $C_{Fk}^{(2)}$ multiplication operations (calculated by formula (45)).

$$C_{wFk}^{(2)} = l_w(k + 2k\sigma + 2\sigma + 1) + \gamma(l_w, v) \tag{45}$$

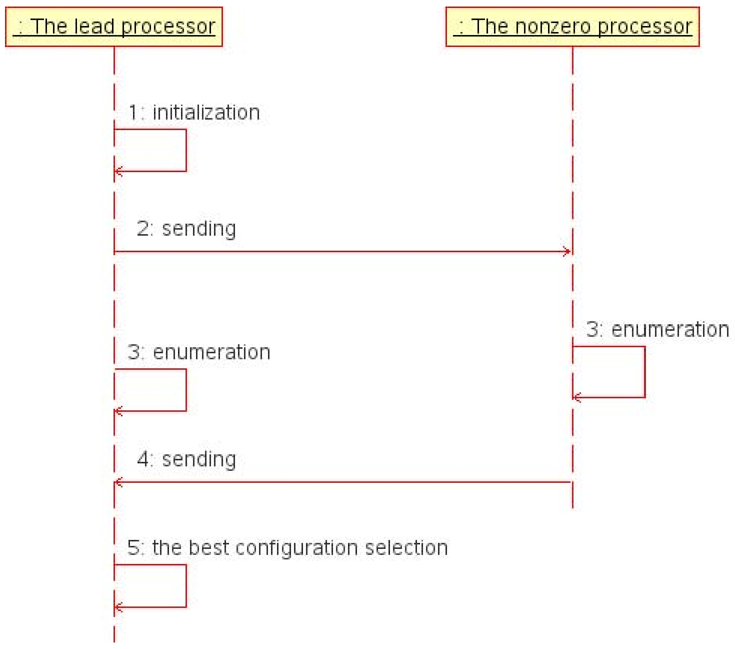

Figure 9: The diagram of the parallel full enumeration method steps.

Table 3: Numbers of multiplication operations in steps of the parallel full enumeration algorithms.

Step	**Zero processors**	**Nonzero (k-th) processor**
1	$2\sigma l_w$	
2	$l_w(n-1)(1+2\sigma)$	$l_w(k+2k\sigma+2\sigma+1)+\gamma(l_w,v)$
3	$z_\varepsilon \hat{J}_F + 2\sigma \hat{J}_F$	$z_\varepsilon J_F + 2\sigma J_F$
4	$n(l_w+1)(1+2\sigma)+\gamma(l_w+1)$	$(l_w+1)(1+2\sigma)$
5	$2\sigma n$	

The number of operations in the third step is analogous to the number of operations which are executed by the serial algorithm.

In the fourth step the lead processor needs to get the best weights coefficients and the inaccuracy value which belongs to these coefficients. It needs to get this information from each processor, so it executes $(n-1)(l_w+1)$ multiplication and $2(n-1)(l_w+1)$ addition operations, and the nonzero processors execute l_w+1 multiplication and $2l_w+2$ addition operations. And the lead processor has to wait for the sending.

In the fifth step zero processor executes $2n$ addition operations. The reduction of addition operations to multiplication operations allows to get values which are shown in table 3.

Before the the lead processor begins to receive weight coefficients and inaccuracy values, it executes three steps (the total number of operations is $\hat{C}_{F0}$), and the other processors execute four steps (the total number of operations is $\hat{C}_{Fk}$). The alues of $\hat{C}_{F0}$

and $\hat{C}_{Fk}$ are calculated by formulas (46)–(47).

$$\begin{aligned} \hat{C}_{wF0} &= 2\sigma l_w + l_w(n-1)(1+2\sigma) + z_\varepsilon \hat{J}_F + 2\sigma \hat{J}_F && (46)\\ \hat{C}_{wFk} &= l_w(k+2k\sigma+\sigma+1) + \gamma(l_w, v) + z_\varepsilon J_F + 2\sigma J_F + && (47)\\ &\quad +(l_w+1)(1+2\sigma) \end{aligned}$$

And it is necessary to consider the time for sending the inaccuracy value and the weight coefficients. The receiving finishes after the the slowest processor has sent the inaccuracy, and this is why the parallel calculation of inaccuracy needs to execute Z_{wF} operations. This value is calculated by formula (48).

$$\begin{aligned} Z_{wF} &= \max_{k=1..n-1}\left(\max\left(\hat{C}_{wF0} + k(l_w+1)(2\sigma+1), \hat{C}_{wFk} + \gamma(l_w+1, v)\right)\right) + \\ &\quad +(l_w+1)(1+2\sigma) + 2\sigma n && (48) \end{aligned}$$

So the efficiency of the parallel algorithm can be defined in formula (49):

$$\alpha_{wF}(Z) = \frac{z_{wF}}{nZ_{wF}} = \frac{z_\varepsilon I_F + 2\sigma I_F}{nZ_{wF}} \tag{49}$$

3.4 Gradient methods

Gradient methods are based on the calculation of a gradient vector and change the values of the weight coefficients in the opposite direction:

$$\vec{w}^{(I)} = \vec{w}^{(I-1)} - \vec{s}^{(I)}\nabla\varepsilon^{(I)} \tag{50}$$

where $\vec{s}$ is the coefficient of training and $\nabla\varepsilon$ is the gradient which is calculated by formula (51) [18, 19]. The values of $\vec{p}(\vec{w})$ and the coefficient vector $\vec{s}$ are selected and it is necessary for $\varepsilon^{(I)}$ to be less than $\varepsilon^{(I-1)}$ for each element of the coefficient vector (as is shown in formula (52)). The searching of the minimum continues until the reduction of the gradient norm is below the value which was set before (or the finishing the time which is allowed for the training).

$$\begin{aligned} \nabla\varepsilon &= \left(\frac{\partial\varepsilon}{\partial w_0}, \frac{\partial\varepsilon}{\partial w_1}, \ldots, \frac{\partial\varepsilon}{\partial w_{l_w-1}}\right) && (51)\\ \vec{w}^{(I)} &= \vec{w}^{(I-1)} + \vec{s}^{(I)}\vec{p}(\vec{w}^{(I-1)}) && (52) \end{aligned}$$

Three methods which are the most useful will be considered. These are the steepest descent method, QuickProp and RPROP. They use different formulas for updating the weight coefficients.

The steepest descent method uses formula (53) or sometimes (54).

$$\begin{aligned} \vec{p}(\vec{w}^{(I)}) &= -\nabla\varepsilon^{(I-1)} && (53)\\ w_i^{(I)} &= w_i^{(I-1)} + s_i^{(I)}\frac{\partial\varepsilon(\vec{w}^{(I-1)})}{\partial w_i^{(I-1)}} + q\Delta w_i^{(I-1)} && (54) \end{aligned}$$

Table 4: Numbers of operations which are necessary for one iteration of gradient methods.

Method	**The multiplicative operations number**	**The additive operations number**
The steepest descent	$l_w(\zeta_\varepsilon + 4)$	$l_w(\hat{\zeta}_\varepsilon + 5)$
QuickProp	$l_w(\zeta_\varepsilon + 4)$	$l_w(\hat{\zeta}_\varepsilon + 5)$
RPROP	$l_w(\zeta_\varepsilon + 4)$	$l_w(\hat{\zeta}_\varepsilon + 4)$

where q is the coefficient of the moment which lies in the interval [0, 1]. As can be seen, this algorithm executes three multiplication and two additive operations for the calculation of updating one element of the weight coefficient vector.

The algorithm QuickProp which was suggested by Falman uses formula (55) [20, 21].

$$\Delta w_i^{(I)} = \begin{cases} q_i^{(I)} \Delta w_i^{(I-1)}, & \Delta w_i^{(I-1)} \neq 0; \\ s_i^{(I)} \nabla \varepsilon_i^{(I-1)}, & \Delta w_i^{(I-1)} = 0; \end{cases} \tag{55}$$

$$q_i^{(I)} = \min\left(\frac{\nabla \varepsilon_i^{(I-1)}}{\nabla \varepsilon_i^{(I-2)} - \nabla \varepsilon_i^{(I-1)}}, q_{MAX} \right) \tag{56}$$

The maximal value of the coefficient of the moment q_{MAX} is 1.75 [22]. This algorithm executes three multiplicative and two additive operations too.

Another heuristic algorithm which is called RPROP (Resilient back PROPagation) was developed by Riedmiller and Brawn [23, 24]. It calculates the elements of the vector of coefficients by formula (57).

$$s_i^{(I)} = \begin{cases} min(q_a s_i^{(I-1)}, q_{MAX}), & \nabla \varepsilon_i^{(I-1)} \nabla \varepsilon_i^{(I-2)} > 0 \\ max(q_b s_i^{(I-1)}, q_{MIN}), & \nabla \varepsilon_i^{(I-1)} \nabla \varepsilon_i^{(I-2)} < 0 \\ s_i^{(I-1)}, & \nabla \varepsilon_i^{(I-1)} \nabla \varepsilon_i^{(I-2)} = 0 \end{cases} \tag{57}$$

where q_{MAX} and q_{MIN} are maximal and minimal values of the coefficient of the training (q_{MAX}=50, $q_{MIN} = 10^{-6}$ [24]), q_a and q_b are constant (q_a=1.2, q_b=0.5) [25]. The speciality of this method is that it ignores the gradient value. The vector of the direction $\vec{p}(\vec{w})$ is determined by the sign (-1, 0, 1) [26]. This algorithm executes three multiplication and one addition operations.

It is necessary to execute $\xi_\varepsilon + 1$ multiplicative and $\xi_\varepsilon + 3$ additive operations for the calculation of one element of the gradient. So it is possible to formulate that one operation needs the operations numbers which are showed in table 4.

As shown in figure 10, each parallel iteration of the gradient method consist of few steps:

- sending the vector of weight coefficients $\vec{w}$ to all nonzero processors by the lead processor;

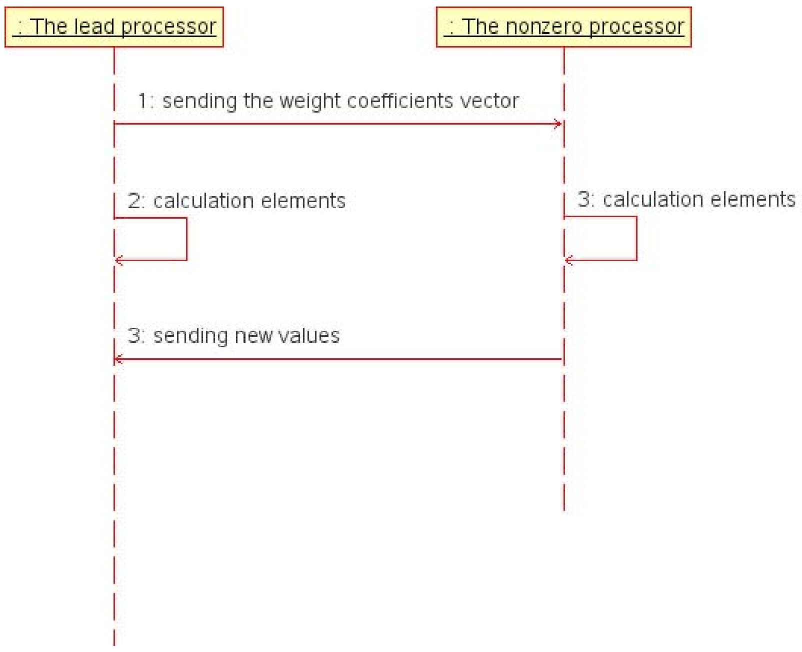

Figure 10: The diagram of the parallel gradient method steps.

- calculating the elements of the part of weight coefficients and the gradient which belongs to the current processor;
- sending new values of weight coefficients to the lead processor by all nonzero processors.

The first step needs $l_w(n-1)$ multiplicative and $2l_w(n-1)$ additive operations in the lead processor and $l_w(k+1)+\gamma(l_w,v)$ operations in the k -th processor (k). The number of operations which are necessary for the second step are shown in table 5. For the third step it executes $\hat{l}_w$ multiplicative and $2\hat{l}_w$ additive operations in the non-lead processors and it executes $\hat{l}_w n+\gamma(l_w,v)$ ($\hat{l}_w(n-1)$ multiplicative ($\hat{l}_w(n-1)$ operations for the receiving and $\hat{l}_w$ operations for the waiting) and $2\hat{l}_w(n-1)$ additive operations in the lead processor.

The first step needs $C^{(1)}_{wG0}$ operations in the lead processor and $C^{(1)}_{wGk}$ in other processors (these values are calculated by formulas (58)-(59)). The second step needs $C^{(2)}_{wG0}$ and $C^{(2)}_{wGk}$ operations (see the data from table 5) and the third step needs $C^{(3)}_{wGk}$operations (calculated by formula (60)) in the nonzero processors. So the lead processor has to wait for $\hat{C}_{wG}$ operations before it starts receiving. The value of $\hat{C}_{wG}$ is calculated by formula (61).

$$C^{(1)}_{wG0} = l_w(n-1)(1+2\sigma) \tag{58}$$
$$C^{(1)}_{wGk} = l_w(k+1)(1+2\sigma)+\gamma(l_w,v) \tag{59}$$
$$C^{(3)}_{wGk} = \hat{l}_w+2\sigma\hat{l}_w \tag{60}$$
$$\hat{C}_{wG} = \max\left(C^{(1)}_{wG0}+C^{(2)}_{wG0}, C^{(1)}_{wGk}+C^{(2)}_{wGk}+C^{(3)}_{wGk}+\gamma(\hat{l}_w,v)\right) \tag{61}$$

So the parallel algorithm executes Z_{wG} operations (calculated by formula (62)), and the efficiency of parallel gradient algorithms which is calculated by formula (63) may

Table 5: Numbers of operations which are necessary for the second step of the parallel iteration of gradient methods.

Method	**Number of mult. operations in the lead processor**	**Number of additive operations in the lead processor**	**Number of mult.. operations in the non-lead processor**	**Number of additive operations in the non-lead processor**
The steepest descent	$\hat{N}_w\ (\zeta_\varepsilon + 4)$	$\hat{N}_w\ (\hat{\zeta}_\varepsilon + 5)$	$\hat{l}_w\ (\zeta_\varepsilon + 4)$	$\hat{l}_w\ (\hat{\zeta}_\varepsilon + 5)$
QuickProp	$\hat{N}_w\ (\zeta_\varepsilon + 4)$	$\hat{N}_w\ (\hat{\zeta}_\varepsilon + 5)$	$\hat{l}_w\ (\zeta_\varepsilon + 4)$	$\hat{l}_w\ (\hat{\zeta}_\varepsilon + 5)$
RPROP	$\hat{N}_w\ (\zeta_\varepsilon + 4)$	$\hat{N}_w\ (\hat{\zeta}_\varepsilon + 4)$	$\hat{l}_w\ (\zeta_\varepsilon + 4)$	$\hat{l}_w\ (\hat{\zeta}_\varepsilon + 4)$

be formulated as (64) for the method of the steepest decent and QuickProp and (65) for RPROP.

$$Z_{wG} = I_G(\hat{C}_{wG} + k(\hat{l}_w + 1)(2\sigma + 1) + 2\sigma\hat{l}_w(n-1)) + \lambda_G \tag{62}$$

$$\alpha_{wG}(Z) = \frac{z_{wG}}{nZ_{wG}} \tag{63}$$

$$\alpha_{wGQ}(Z) = \frac{I_G l_w(\zeta_\varepsilon + 4 + \sigma\hat{\zeta}_\varepsilon + 5\sigma) + \lambda_G}{nZ_{wG}} \tag{64}$$

$$\alpha_{wGR}(Z) = \frac{I_G l_w(\zeta_\varepsilon + 4 + \sigma\hat{\zeta}_\varepsilon + 4\sigma) + \lambda_G}{nZ_{wG}} \tag{65}$$

where I_G is the number of iterations and, λ_G is the number of other operations of the algorithm.

4 Results

4.1 Experiments

For checking the formulas which were developed here an experiment was done. In this experiment a forecast of currency exchange rates €/$was attempted at. It uses two ANN structures, namely a multilayer perceptron (Fig. 11) and a Volterra network. The pattern consisted of 300 rows and each row consisted of 12 input and one output values. The results of this experiment are shown in table 6.

Another experiment was the prediction of shrimp mass caught in the Indian ocean. For this experiment the MLP (Fig. 12) was used. Results are shown in table 7.

Thus the efficiency of the parallel algorithm of the full enumeration is about 95%, the efficiency of the gradient algorithms is about 93% and the efficiency of the parallel algorithms of the target function minimizing is about 91%.

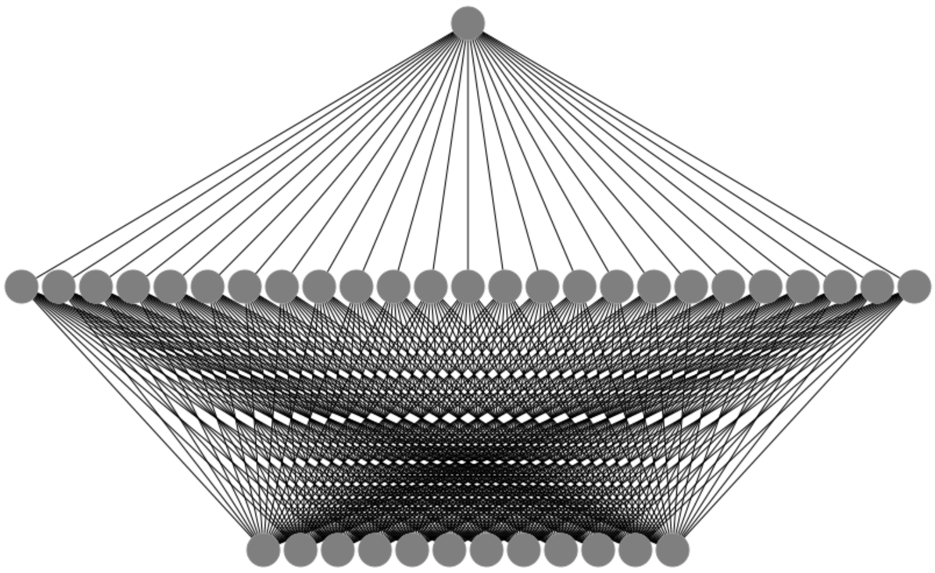
Figure 11: The MLP which was used for the prediction of currency pair quotations.

Table 6: Values of efficiency coefficient for prediction of currency pair quotations.

4	0.9127	0.9521	0.9319
6	0.9111	0.9509	0.9302
8	0.9101	0.9492	0.9293

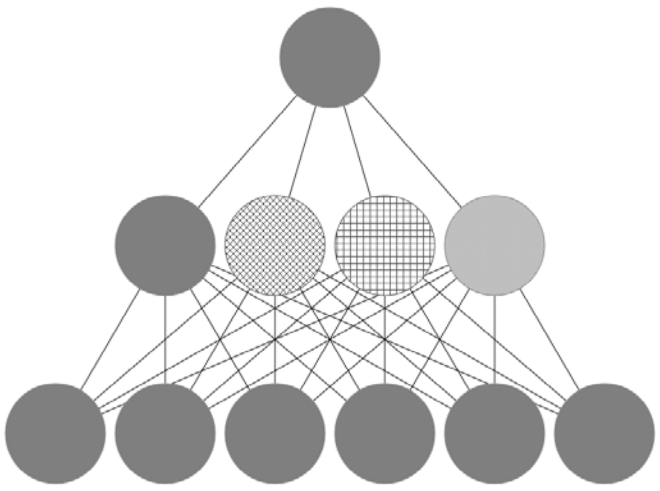
Figure 12: The MLP which was used for the forecast of the shrimp mass which was caught.

4.2 Conclusion

The time which is necessary for training the ANN by the parallel algorithm can be calculated by formula (66):

$$\tau \approx \frac{t}{n\alpha(Z)} \tag{66}$$

Table 7: Values of the efficiency coefficient for predicting the shrimp mass which was caught.

Processors number	α_{ε}	α_{wF}	α_{wG}
4	0.9124	0.9518	0.9312
6	0.9112	0.9501	0.9304
8	0.9103	0.9494	0.9297

where t is the time which is necessary for training the ANN by the seral algorithm.

So it is possible to conclude that the usage of parallel algorithms of training ANNs allows to lower capacity and time expenses. So parallel algorithms are very effective.

References

[1] Горбань А.Н., Дунин-Барковский В.Л., Кирдин А.Н., Миркес Е.М., Новоходько А.Ю., Россиев Д.А., Терехов С.А., Сенашова М.Ю., Царегородцев В.Г. и др. Нейроинформатика // Новосибирск: Наука, 1998, – 296 с. *(Gorban A.N., Dunin-Barkovskiy V.L., Kirdin A.N., Mirkes E.M., Novahodko A.Y., Rossiev D.A., Terehov S.A., Senashova M.Y., Tcaregorodtcev V.G. and etc. Neural computer science // Novosibirsk: Nauka, 1998 – 296 p.)*

[2] Зенкова Н.А. Моделирование на основе аппарата искусственных нейронных сетей как метод исследования в психологической науке. // Вестн. Тамб. ун-та. Сер. Естеств. и техн. науки. Тамбов: 2009. Т. 14, Вып. 3. – С. 577-591. *(Zenkova, N.n»„. The simulation based on artificial neural networks how discussion method in psychology science. // Tambov University Reports. Series: Natural and Technical Sciences. Tambov, 2009. Vol. 14, Issue. 3 P. – 577-591.)*

[3] Osowsky S. Sieci neuronowe w ujeciu algorytmicznym // Warszawa. 1996.

[4] Крючин О.В. Использование кластерных систем для обучения искусственных нейронных сетей при применении параллельного вычисления значения невязки // Наука и образование в развитии промышленной, социальной и экономической сфер регионов России [Электронный ресурс]: II Всероссийские научные Зворыкинские чтения. Сб. тез. докладов II Всероссийской межвузовской научной конференции. -Муром: полиграфический центр МИ ВлГУ, 2010. – 802 с. *(Kryuchin O.V. The usage of computer clusters for the artificial neural networks training using the parallel calculation of inaccuracy // Science and education in the evolution of the industrial, social and economic sphere of Russian regions: II All-Russian interacademic science conference – Murom: polygraphic center MI VlSU, 2010 – 802 p.)*

[5] Крючин О.В. Параллельные алгоритмы обучения искусственных нейронных сетей // Информационные технологии и математическое моделирование

(ИТММ-2009) : Материалы VIII Всероссийской научно-практической конференции с международным участием . - Томск: Изд-во Том. Ун-та, 2009. – Ч. 2. С. 241-244. *(Kryuchin O.V. Parallel algorithms of artificial neural networks training // Information technologies and mathematic simulation (ITMS-2009) : Material of VIII All-Russian science-practical conference with international participation – Tomsk: publishing of Tomsk University, 2009 – Part 2. P. 241-244.)*

[6] Крючин О.В. Разработка параллельных градиентных алгоритмов обучения искусственной нейронной сети // Электронный журнал "Исследовано в России", 096, С. 1208-1221, 2009 // http://zhurnal.ape.relarn.ru/articles/2009/096.pdf. *(Kryuchin O.V. The development of parallel gradient algorithms of the artificial neural networks training // Electronic journal "Investigated in Russia" 096, P. 1208-1221, 2009) //* `http://zhurnal.ape.relarn.ru/articles/2009/096.pdf`

[7] Солдатова О.П., Семенов В.В. Применение нейронных сетей для решения задач прогнозирования // Электронный журнал "Исследовано в России", 136, С. 1270-1270, 2009 *(Soldatova O.P., Semenov V.V. The usage of neural networks for forecast problems solving // Electronic journal "Investigated in Russia" 136, P. 1270-1270, 2009) //* `http://zhurnal.ape.relarn.ru/articles/2006/136.pdf`

[8] Rosenblatt F. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. // Washington DC: Spartan Books, 1961

[9] Rumelhart D.E., Geoffrey E.H., Williams R.J. Learning Internal Representations by Error Propagation. // Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundations. MIT Press, 1986.

[10] Cybenko G. Approximation by superpositions of a sigmoidal function Mathematics of Control, Signals, and Systems (MCSS), 2(4), P. 303-314. [Electronic resource] // `http://www.springerlink.com/content/n873j15736072427/`

[11] Fahlman S.E., Lebiere C. The cascade-correlation learning architecture. Tech. Rep. CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, August 1991.

[12] Fahlman, S.E. The recurrent cascade-correlation architecture. Tech. Rep. CMU-CS-91-100, School of Computer Science, Carnegie Mellon University, 1991.

[13] Hoefeld M., Fahlman S.E. Learning with limited numerical precision using the cascade-correlation algorithm. Tech. Rep. CMU-CS-91-130, School of Computer Science, Carnegie Mellon University, 1991.

[14] Osowsky S., Siwek K. Study of generality ability of neural networks // III Konferencja "Sieci Neuronowe i ich zastosowania". Kule, 1997

[15] Крючин О.В, Арзамасцев А.А. Прогнозирование котировок валютных пар при помощи искусственной нейронной сети. // Вестн. Тамб. ун-та. Сер. Естеств. и техн. науки. Тамбов: 2009. Т. 14, Вып. 5. Џ С. 591-596. *(Kryuchin O.V., Arzamastsev A.A. The forecasting of currency pairs by artificial neural network. // Tambov University Reports. Series: Natural and Technical Sciences. Tambov, 2009. Vol. 14, Issue. 5. – P. 591-596.)*

[16] Hertz J., Kroght A., Palmer R. Wstep do teorii obliczen neuronowych. Wyd. II.– Warszawa: WNT, 1995.

[17] Kryuchin O.V., Arzamastsev A.A., Troitzsch K.G. A universal simulator based on artificial neural networks for computer clusters // Fachbereich Informatik Nr. 2/2011 // [Electronic resource]. `http://www.uni-koblenz.de/f̃b4reports/2011/2011_02_Arbeit-sberichte.pdf`

[18] Gill P., Murray W. Wrights M. Practical Optimisation. – N.Y.: Academic Press, 1981.

[19] Widrow B., Stearns S. Adaptive signal processing. – N.Y.: Prentice Hall, 1985.

[20] Fahlman S.E. Faster-learning variations on back-propagation: An empirical study. In 1988 Connectionist Models Summer School (San Mateo, CA, 1988), T. J. S. G. E. Hinton and D. S. Touretzky, Eds., Morgan Kaufmann.

[21] Veith A.C., Holmes G.A. A modified quickprop algorithm // Neural Computation, Vol. 3, 1991. – P. 310-311.

[22] Fahlman S.E. An empirical study of learning speed in back-propagation networks. Technical report, CMU-CS-88-162, Carnegie-Mellon University, 1988.

[23] Riedmiller M., Brawn H. RPROP – a fast adaptive learning algorithms. Technacal Report, Karlsruhe: University Karlsruhe, 1992.

[24] Zadeh L.A., The concept of linguistic variable and its application to approximate reasoning. Part 1-3 // Information Sciences, 1975. – P. 199-249.

[25] Riedmiller M. Untersuchungen zu konvergenz und generalisierungsverhalten uberwachter lernverfahren mit dem SNNS. In Proceedings of the SNNS. 1993.

[26] Крючин О.В. Разработка параллельных эвристических алгоритмов подбора весовых коэффициентов искусственной нейронной сети // М.: Информатика и ее применение, Т. 4, Вып. 2, 2010. – С. 53-56. *(Kryuchin O.V. The development of parallel heuristic algorithms of selection of weight coefficients of the artificial neural network // Moscow : Computer science and its adaptation, V. 4, Issue 2, 2010. – P. 53-56.)*

Glossary

$C^{(\iota)}_{\varepsilon k}$	the number of operations which are executed in the ι step of parallel algorithm of the calculation of the inaccuracy in the k-th processor, 12
$C^{(\iota)}_{wFk}$	the number of operations which are executed in the ι step of parallel full enumeration algorithm in the k-th processor, 16
$C^{(\iota)}_{wGk}$	the number of operations which are executed in the ι step of parallel gradient algorithm in the k-th processor, 21
F	the function which calculates output values of the ANN, 10, 11
I_F	the number of iterations which are done by full enumeration algorithm, 15
I_G	is the number of iterations which are done by the gradient algorithm, 21
I_ε	the number of calculation of the inaccuracy in the parallel algorithm of the calculation of the target function value, 14
J_F	the number of iterations which are done by parallel full enumeration algorithm in the non-lead processor, 15
L	the number of neurons in the input layer, 2
M	the number of pattern rows which are used by the non-lead processor for the calculation of the inaccuracy in the parallel calculation of the target function value, 11
N	the number of rows in a pattern, 10
N_L	the number of layers in the network, 2
N_x	the number of neuron input, 7
P	the number of neurons in the output layer, 2
Z	the number of operations which are executed by the parallel algorithm (additive operation are recuted to multiplicative operations), 10
Z_ε	the number of operations which are necessary for the parallel calculation of the calculation of the target function value (additive operation are recuted to multiplicative operations), 11
Z_{wF}	the number of operations which are executed by the parallel full enumeration algorithm (additive operation are recuted to multiplicative operations), 17

Z_{wG}	the number of operations which are executed by the parallel gradient algorithm (additive operation are recuted to multiplicative operations), 21
Δw_i	the value of the change of the i-th weight coefficient, 18
$\alpha(Z)$	the coefficient of the efficiency of the parallel algorithm (additive operation are recuted to multiplicative operations), 10
$\alpha_\varepsilon(Z)$	the coefficient of the efficiency of the parallel algorithm of the calculation of the target function value, 14
$\alpha_{wF}(Z)$	the coefficient of the efficiency of the parallel full enumeration algorithm (additive operation are recuted to multiplicative operations), 17
$\alpha_{wGQ}(Z)$	the coefficient of the efficiency of the parallel gradient algorithm of QuickProp and the steepest descent (additive operation are recuted to multiplicative operations), 21
$\alpha_{wGR}(Z)$	the coefficient of the efficiency of the parallel gradient algorithm of RPROP (additive operation are recuted to multiplicative operations), 21
γ	the function which returns the number of operations which are necessary for the sending data from one processor to other, 12
$\hat{C}_{\varepsilon 0}$	the number of operations which are executed by the lead processor for the sending weight coefficients to all other processors and calculates inaccuracy ε_0 by the algorithm which parallel calculates the target function value, 14
$\hat{C}_{\varepsilon k}$	the number of operations which are executed by non-lead processors for the receiving the vector of weight coefficients, the calculating the value of the inaccuracy ε_k and the sending result to the lead processor by the algorithm which parallel calculates the target function value, 14
$\hat{C}_{wF0}$	the number of operations which are executed by the lead processor in first three steps $(C_{wF0})^{(1)} + C_{wF0})^{(2)} + C_{wF0})^{(3)}$ by the parallel full enumeration algorithm, 17

$\hat{C}_{wFk}$	the number of operations which are executed by non-lead processors in first four steps $(C_{wFk})^{(1)} + C_{wFk})^{(2)} + C_{wFk})^{(3)} + C_{wFk})^{(4)}$ by the parallel full enumeration algorithm, 17
$\hat{C}_{wG}$	the number of operations which are waited by the lead processor before the beginning receiving weight coefficients in parallel gradient algorithms, 21
$\hat{J}_F$	the number of iterations which are done by parallel full enumeration algorithm in the lead processor, 15
$\hat{M}$	the number of pattern rows which are used by the lead processor for the calculation of the inaccuracy in the parallel calculation of the target function value, 11
$\hat{N}_i$	the number of neurons in the i-th layer, 2
$\hat{N}_w$	the number of weight coefficients (and elements of the gradient vector) which are calculated in the lead processor by the parallel gradient algorithm, 20
$\hat{P}$	the number of pattern elements which are sent by the lead processor to each nonzero process in the parallel calculation of the target function value, 12
$\hat{\zeta}_\varepsilon$	the number of additive operations which are necessary for the calculation of the target function value, 11
$\hat{\zeta}_n$	the number of additive operations which are necessary for the calculation of the output values of a neuron, 7
$\hat{\zeta}_n(\mu_i)$	the number of additive operations which are necessary for the calculation of the output values of the i-th neuron, 9
$\hat{\zeta}_y$	the number of additive operations which are necessary for the calculation of the output values of an ANN, 9
$\hat{\zeta}_{yC}$	the number of additive operations which are necessary for the calculation of the output values of a cascade-correlation network, 9
$\hat{\zeta}_{yM}$	the number of additive operations which are necessary for the calculation of the output values of a multilayer perceptron, 9

$\hat{\zeta}_{yV}$	the number of additive operations which are necessary for the calculation of the output values of a Volterra network, 9
$\hat{l}_w$	the number of weight coefficients (and elements of the gradient vector) which are calculated in the non-lead processor by the parallel gradient algorithm, 20
λ_G	is the number of operations which are done by the gradient algorithm and which are not operations of the parallel calculation of weight coefficients, 21
λ_ε	the number of operations of the parallel algorithms (of the calculation of the target function value) which are not belong to the calculation of the inaccuracy, 14
σ	the coefficient of the reduction of additive operations to multiplicative operations, 10
$\vec{\mu}$	the activation functions vector, 9
$\vec{\nabla}\varepsilon$	the vector of the gradient, 18
$\vec{p}$	the vector of functions of the changing weight coefficients, 18
$\vec{s}$	the vector of the training, 18
$\vec{w}$	the vector of weight coefficients, 5
$\vec{x}$	the vector of input values, 2
$\vec{y}$	the vector of output values, 2
ζ_ε	the number of multiplicative operations which are necessary for the calculation of the target function value, 11
ζ_n	the number of multiplicative operations which are necessary for the calculation of the output values of a neuron, 7
$\zeta_n(\mu_i)$	the number of multiplicative operations which are necessary for the calculation of the output values of the i-th neuron, 9
ζ_y	the number of multiplicative operations which are necessary for the calculation of the output values of an ANN, 9
ζ_{yC}	the number of multiplicative operations which are necessary for the calculation of the output values of a cascade-correlation network, 9
ζ_{yM}	the number of multiplicative operations which are necessary for the calculation of the output values of a multilayer perceptron, 9

ζ_{yV}	the number of multiplicative operations which are necessary for the calculation of the output values of a Volterra network, 9
c_f	the coefficient of an activation function, 7
d	an output pattern matrix, 10
l_i^{down}	the lower limits of the i-th weight coefficient, 15
l_i^{up}	the upper limits of the i-th weight coefficient, 15
l_μ	the total number of neurons in a network, 9
l_w	the number of weights coefficients, 12
n	the number of used processors, 11
p_E	the external impulse, 7
p_I	the internal impulse, 7
q	the coefficient of the moment, 18
t	the points of time, 5
v	the rate of an interconnect, 12
z	the number of operations which are executed by the serial algorithm (additive operation are recuted to multiplicative operations), 10
z_ε	the number of operations which are necessary for the calculation of the serial calculation of the target function value (additive operation are recuted to multiplicative operations), 11
z_n	the number of operations which are necessary for the calculation of the output values of a neuron (additive operation are reduced to multiplicative operations), 7
$z_n(\mu_i)$	the number of operations which are necessary for the calculation of the output values of the i-th neuron (additive operation are recuted to multiplicative operations), 9
z_y	the number of operations which are necessary for the calculation of the output values of an ANN (additive operation are recuted to multiplicative operations), 9
z_{wF}	the number of operations which are executed by the serial full enumeration algorithm (additive operation are recuted to multiplicative operations), 17
z_{wG}	the number of operations which are executed by the serial gradient algorithm (additive operation are recuted to multiplicative operations), 21

z_{yC}	the number of operations which are necessary for the calculation of the output values of a cascade-correlation network (additive operation are recuted to multiplicative operations), 9
z_{yM}	the number of operations which are necessary for the calculation of the output values of a multilayer perceptron (additive operation are recuted to multiplicative operations), 9
z_{yV}	the number of operations which are necessary for the calculation of the output values of a Volterra network (additive operation are recuted to multiplicative operations), 9

Bisher erschienen