UDC 004.032.26

SOME CASES OF MLFF NETWORKS MODELING USING GRADIENT LEARNING ALGORITHMS

[1]Mykytenko N., [2]Sedov Ye.

[1,2]South Ukrainian National Pedagogical University, Odessa, Ukraine

The paper considers the speed and the exactness of multi-layer feed-forward (MLFF) neural network learning using various learning algorithms and presentation of input data. The choise of the learning algorithm and the expediency of normalizing input data and its type depend on the initial sequence and the configuration of the network. The speed of network is determined by network configuration using different values of input data and learning techniques. As a result the possibility to compare the rightness degree of forecasting each variants of learning algorithm and to define the most preferable variant for the sequence are shown.

KEY WORDS: neural networks, learning algorithms, multi-layer feed-forward neural network.

НЕКОТОРЫЕ СЛУЧАИ МОДЕЛИРОВАНИЯ MLFF СЕТЕЙ С ИСПОЛЬЗОВАНИЕМ ГРАДИЕНТНЫХ ОБУЧАЮЩИХ АЛГОРИТМОВ

Микитенко Н., Седов Е.

В представленной работе исследуются скорость и точность многослойной обучающей нейронной сети с прямой связью (MLFF) с использованием различных обучающих алгоритмов и представлением входных данных. Выбор обучающего алгоритма, целесообразности нормализации входных данных и их типа зависят от начальной последовательности и конфигурации сети. Скорость сети определяется конфигурацией сети с использованием различных значений входных данных и обучающих алгоритмов. В результате показаны возможности сравнения правильности предсказания каждого типа обучающего алгоритма, и определения наиболее предпочтительного варианта начальной последовательности.

КЛЮЧЕВЫЕ СЛОВА: нейронные сети, обучающие алгоритмы, многослойные нейронные сети с прямой связью.

ДЕЯКІ ВИПАДКИ МОДЕЛЮВАННЯ MLFF МЕРЕЖ З ВИКРИСТАННЯМ ГРАДІЄНТНИХ НАВЧАЮЧИХ АЛГОРИТМІВ

Микитенко Н., Сєдов Є.

В даній роботі досліджуються швидкість і точність багатошарової навчальної нейронної мережі з прямим зв'язком (MLFF) з використанням різних навчаючих алгоритмів та поданням вхідних даних. Вибір навчаючого алгоритму, доцільності нормалізації вхідних даних та їх типу залежать від початкової послідовності і конфігурації мережі. Швидкість мережі визначається конфігурацією мережі з використанням різних значень вхідних даних і навчальних алгоритмів. В результаті показані можливості порівняння правильності передбачення кожного типу навчального алгоритму, і визначення найбільш кращого варіанту початкової послідовності.

КЛЮЧОВІ СЛОВА: нейронні мережі, навчальні алгоритми, багатошарові нейронні мережі з прямим зв'язком.

1. Introduction. It has long been discussed that time series forecasting widely used in many fields of science. The forecasting is a result of determination of future values of the sequence, based on some mathematical model with the use of data present in the moment of forecasting. Forecasting of future events is less likely for risk decision.

The actuality of forecasting problem is confirmed by the amount of the written monographs and articles in different magazines, for example, International Journal of Forecasting, Journal of Forecasting and others. Forecasting uses different methods, depending on the kind of the sequence and the final target. These models are widely used to produce high-quality and reliable forecasts.

Time series is the well-organized sequence of numerical indexes that characterizes the levels of studied process in successive moments or time periods.

Time series y_t can be presented as $y_t = \chi_t + S + C + \varepsilon_t$, where χ_t is a trend that characterizes the substantial dynamics of process development, S is a seasonal constituent, C is a cyclic constituent, ε_t is a stochastic component of process, that represents casual vibrations and noises of process. Stationary time series feature is the absence of trend and periodic constituent, systematic changes of vibrations scope and systematic changing dependences between the time series elements. Non-stationary time series feature is a presence of the above-stated components.

There are many forecasting methods, each of them can be applied to certain kinds of sequences. The books and articles of different authors [1–4] describe forecasting experiments using linear and nonlinear methods, identified the advantages and disadvantages of each method. In [3] authors generalized results of the numerous researches conducted within the different rows (3003 rows) analysis and comparison by different methods (24 methods). The results are alike with the results of other researchers: the difficult or complex forecasting methods do not give the best prognoses comparatively with more simple; the estimation of forecasting quality depends on the fact, what description of quality is chosen by standard and depends on forecasting length. In [4] authors concluded that linear methods give better prognoses as compared to nonlinear one. For example, prognoses with the use of neural networks, in all cases appear worse than the prognoses with the use of autoregressive models. However in other sources [5] nonlinear methods, in particular neural networks, give better results as compared with linear ones.

The neural network capacities to forecast straight ensue from its capacity for generalization and selection of the hidden dependences between input and output data. After teaching a network is able to predict the future values of the sequence on the basis of a few previous values or some existing presently factors. It should be noted that forecasting is possible when previous changes determine the future values.

In [6] the need to scale the initial values of the sequence was described, so that they fall into the scope of the network. In [7] it was showed that the speed and the exactness of neural networks teaching for the forecasting tasks can depend on the type of input data presentation. The expediency of normalizing and its type depend on the initial sequence and the configuration of the network. In this paper we compare the results of forecasting time series values, where input data represented by one of the normalizing type, using different learning algorithms.

2. Neural network structure. A neural network is an interconnected network of simple processing elements (neurons) with a different weight associated with each connection.

The basic building block of a neural network is the neuron. The neuron consists of a propagation function g and an activation function f, where f takes the output of g as an argument. The propagation function g is the weighted sum of inputs. The activation function f can be a linear function or non-linear function and it determines dependence of signal on the neuron output from the self-weighted sum of signals on his entrances. Thus, a neuron can be represented in the general form as $y(x) = f(g(x; w))$. Such neurons are assembled in layered structure to construct the artificial neural network (ANN).

Development of ANN model for any system involves three important issues: topology of the network, proper training algorithm, activation function.

For the output units an activation function, that is suited to the distribution of the target values, should be chosen. For binary targets the step function can be used. For targets with a bounded range the sigmoid and tanh functions can be used, provided either scale the outputs to the range of the targets or scale the targets to the range of the output activation function. For targets with no known bounds, the linear activation function is used.

An ANN involves an input layer and an output layer connected through one or more hidden layers. Multiple layers of neurons, usually interconnected in a feed-forward way, implement multi-layer feed-forward neural network (MLFF).

The network learns by adjusting the interconnections between the layers. When the learning procedure is completed, a suitable output is produced at the output layer. The learning procedure may be supervised or unsupervised. In prediction problem supervised learning is adopted, where a desired output is assigned to network beforehand.

3. Learning techniques. ANN learning is the process of changing the weights in the network to achieve the desired result. Usually neural network training is carried out on a sample. After learning by some algorithm, the network should get better and better respond to input values. Consider some supervised learning algorithms.

Back-propagation algorithm (BPA) is an iterative gradient algorithm which is used to minimize the ANN error and the desired output. The basic idea of this method is to extend the error signals from the output of the network to its inputs in the direction opposite to the forward propagation of signals in normal operation. It is the most useful for feed-forward networks. Back-propagation requires that the activation function, used by the artificial neurons, must be differentiable.

The error function defined as

$$E\left(\left\{w_{i,j}\right\}\right) = \frac{1}{2}\sum_{i=1}^{p}(t_i - y_i)^2 ,$$

where y_i is the network output, t_i is the target, p is the training set, $w_{i,j}$ is the weights between i and j neurons.

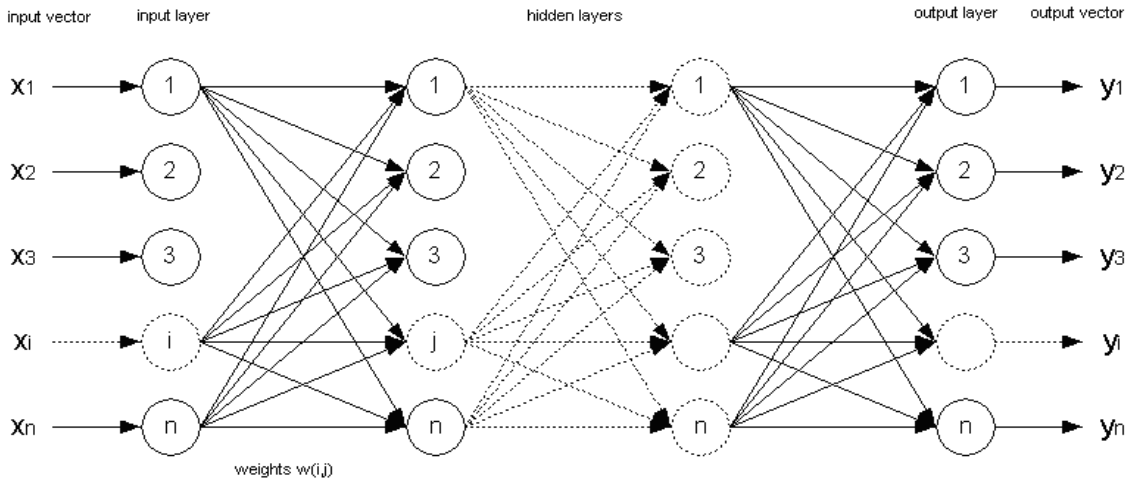

Fig. 1. MLFF neural network structure.

Change of weights can be done after each training set either once after the presentation of all training vectors. Learning goal is to determine the neurons weights of each layer of the network to a given input vector to get the output values satisfying the required accuracy. Learning defines as

$$w_{i,j} = w_{i,j} + \Delta w_{i,j},$$

where $\Delta w_{i,j} = -\eta \frac{\partial E}{\partial w_{i,j}}$, $0 < \eta < 1$ is the learning rate.

While in the back-propagation algorithm network weights are adjusted after each observation, in the Quickprop method the average gradient of the error surface around the training set calculated, and the weights are adjusted once at the end of each period. In the first epoch Quickprop adjusts the weights as well as the back-propagation algorithm. Then, changing the weights are computed as

$$w_k(n) = \frac{\nabla E(w_k)}{\nabla E(w_{k-1}) - \nabla E(w_k)} w_k(n-1).$$

In resilient back propagation (RPROP) changing the weights considered only the sign of the gradient, not its value

$$\Delta w_{ij}(k) = -\eta_{ij}(k)\,\mathrm{sign}\left(\frac{\partial E(w(k))}{\partial w_{ij}}\right),$$

where sign() function returns the gradient sign.

Gradient learning algorithms are associated with the Taylor series expansion of the function $E(w)$ in the neighborhood of w in the p direction. Such expansion is described by

$$E(w+p) = E(w) + [g(w)]^T p + \frac{1}{2} p^T H(w) p + \ldots, \quad (1)$$

where $g(w) = \nabla E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n}\right]^T$ is the gradient vector, $H(w)$ is the matrix of second derivatives (Hessian), p is the direction vector. The dependence (1) can be considered as an approximation of the function $E(w)$ in the vicinity of point w with an accuracy $O\left(h^3\right)$, $h = \|p\|$. In the search for the minimum $E(w)$ the values of the direction p and the step h are selected in such a way that for each new point $w_{k+1} = w_k + \eta_k p_k$ performed the condition $E(w_{k+1}) < E(w_k)$. The search continues until the norm of the gradient reaches the desired accuracy or until it exceeds maximum calculation time.

Steepest descent method is the gradient method, in which the decomposition of the function in a Taylor series can restrict its linear approximation. The direction vector is determined as

$$p_k = -g(w_k).$$

Changing the weights as follows

$$\Delta w_k = \eta_k p_k + \alpha(w_k - w_{k-1}),$$

where α is the momentum coefficient in [0, 1] interval. The higher value of α, the more important momentum for the selection of weights. Selection momentum requires many experiments.

Using the Levenberg-Marquardt algorithm the Hessian replaced by the value of $G(w)$, which is calculated on the basis of a regularization factor. Represent the target function to the existence of a single training set

$$E(w) = \frac{1}{2}\sum_{i=1}^{M} [e_i(w)]^2, \quad (2)$$

where $e_i = (y_i(w) - d_i)$. Denote:

$$e(w) = \begin{bmatrix} e_1(w) \\ e_2(w) \\ \ldots \\ e_M(w) \end{bmatrix}, \quad J(w) = \begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \frac{\partial e_1}{\partial w_2} & \ldots & \frac{\partial e_1}{\partial w_n} \\ \frac{\partial e_2}{\partial w_1} & \frac{\partial e_2}{\partial w_2} & \ldots & \frac{\partial e_2}{\partial w_n} \\ \ldots & \ldots & \ldots & \ldots \\ \frac{\partial e_M}{\partial w_1} & \frac{\partial e_M}{\partial w_2} & \ldots & \frac{\partial e_M}{\partial w_n} \end{bmatrix}.$$

The gradient vector and the approximated Hessian matrix, corresponding to (2), are defined as

$$g(w)=[J(w)]^T e(w),$$
$$G(w)=[J(w)]^T J(w)+R(w),$$

where $R(w)$ identifies the components of the Hessian, including derivatives relative to w. So the direction vector is determined as

$$p_k=-[G(w_k)]^{-1} g(w_k).$$

When choosing the minimizing direction in conjugate gradient algorithm Hessian is not used. The direction p_k is chosen such that it is orthogonal and conjugate to all previous directions $p_0, p_1, ..., p_{k-1}$

$$p_k=-g_k+\beta_{k-1} p_{k-1},$$

where $g_k=g(w_k)$ is the gradient vector, β_{k-1} is the conjugate coefficient.

In this paper we compare the results of forecasting time series values using the considered learning algorithms:

- back-propagation algorithm;
- Quickprop algorithm;
- steepest descent algorithm;
- conjugate gradient algorithm.

After experiments it is possible to determine which algorithm performed the best training.

4. Standardizing input data. Standardizing either input or target variables tends to make the training process better behaved by improving the numerical condition of the optimization problem and ensuring that various default values involved in initialization and termination are appropriate. Standardizing targets can also affect the objective function.

In theory normalizing or standardizing inputs are not necessary. The reason is that any rescaling of an input vector can be effectively undone by changing the corresponding weights, leaving the exact same outputs as had before. However, there are a variety of practical reasons why standardizing the inputs can make training faster and reduce the chances of getting stuck in local optima. Also, weight decay can be done more conveniently with standardized inputs.

In this paper the input sequence for neural network learning presents in such ways: sequence becomes a sequence of bits transferring each number; sequence becomes a sequence of bits by transferring each number in Gray code.

In [7] we showed that for stationary time series with a small dispersion of values in the case of MLFF networks with supervised learning for forecasting the sequence can be represented by binary or Gray code. Using this presentation of sequence elements we can compare the learning speed and accuracy of the network's using different learning techniques.

When using binary code, the data present as a combination of two characters, numbered 0 and 1. In general, the number of combinations n-bit binary code is equal to the number of placements with repetition $A(2,n)=A_2^n=2^n$, where $A(2,n)=A_2^n$ is the number of codes, n is the number of binary digits.

To convert the original sequence of decimal numbers in a sequence of binary numbers, you must first present each number in the form of a nonnegative decimal number. Fractional part is removed by multiplying each of the original sequence by $10*n$, where n is maximum number of digits in the fractional part of numbers. Nonnegativity is achieved by adding a module to a minimum number of initial sequence numbers. These transformations are possible, because they do not affect the learning network.

Once obtained a nonnegative integer sequence, each number in the sequence can be represented in binary. The number of bits in each of those should be the same. To do this, choose the maximum order, convert it to binary code and fix the number of received bits. Since the network can predict the numbers are larger than input, one more bit should be reserved in binary form for such forecasts. As a result, we obtain the maximum number of bits for each binary representation of the number.

But this code is not without drawbacks. The main disadvantage is that adjacent numbers differ in the values of a few bits that could hamper operation. To avoid this problem it is better to use an encoding where adjacent numbers differ fewer positions in the ideal value of one bit. This source code is Gray.

The Gray code may be got out of the binary representation, so you need to perform all operations to the original sequence for the binary case. Gray code can easily be obtained from the binary number by bitwise XOR with the same numbers, shifted right by one bit. Thus, i-th bit Gray code G_i is expressed through the binary B_i as $G_i=B_i\oplus B_{i+1}$, where $\oplus$ – XOR operation; the bits are numbered from right to left, starting with the youngest.

5. Computer experiment. For the research it was examined several types of neural network structures. The better results have been shown by network with two of the hidden layers of 30 and 15 neurons in the layer. For the network learning each of four considered training methods has been used.

For learning the step activation function was used. The step function may be used because the output takes on values of 0 or 1. The threshold of the function is 0,5. At the use of step function until the self-weighted signal on the neuron entrance does not arrive to some level T — a signal on an output is equal to the zero. As soon as a signal on the neuron entrance exceeds the indicated level — an output signal saltatory changes to one.

To form the training set, the sliding window method was chosen. The sliding window method constructs a window classifier h_w that maps an input window of width w into an individual output value y. The window classifier h_w is trained by converting each sequential training example $(x_i,y_i,)$ into windows and then applying a standard supervised learning algorithm. A new sequence x is classified by converting it to windows, applying h_w to predict each y_t and then

concatenating the y_t's to form the predicted sequence y. The obvious advantage of this sliding window method is that permits any classical supervised learning algorithm to be applied.

Consider a sequence, which values variation is in a range [25, 5...103, 5].

Parameters of the network training are the same for all cases: learn rate=0.1; momentum = 0.6; window width = 3.

Determine a speed of network learning with this network configuration using the different values of input data and learning techniques. As a result, it is possible to compare the rightness degree of forecast of each variants of learning algorithm and to define the most preferable variant for the sequence.

Table 1. Network errors with BPA algorithm

Iterations	***Network error***	
	Gray code	***Binary code***
1000	0,123371	0,131531
2000	0,113918	0,128573
3000	0,106972	0,125269

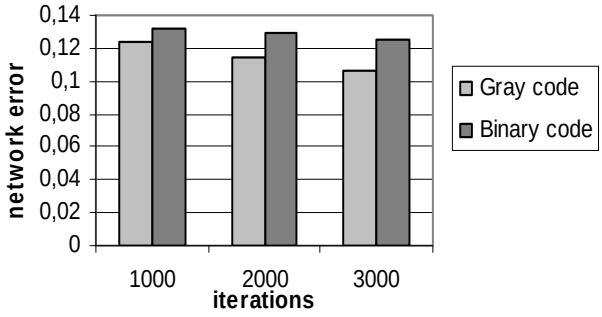

Fig. 2. Comparison of the error values with various input data with BPA.

Table 2. Network errors with Quickprop algorithm

Iterations	***Network error***	
	Gray code	***Binary code***
1000	0,216747	0,160975
2000	0,178947	0,14095
3000	0,128734	0,139837

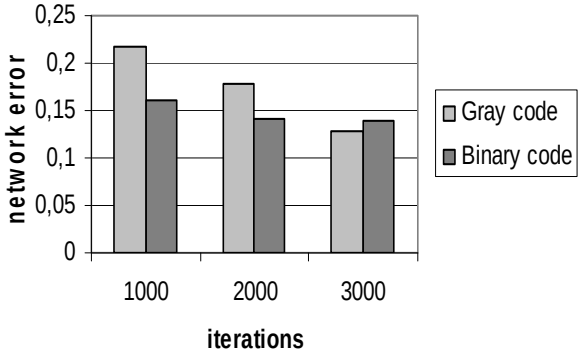

Fig. 3. Comparison of the error values with various input data with Quickprop.

Table 3. Network errors using steepest descent algorithm

Iterations	***Network error***	
	Gray code	***Binary code***
1000	0,109043	0,123454
2000	0,100348	0,110232
3000	0,084501	0,109423

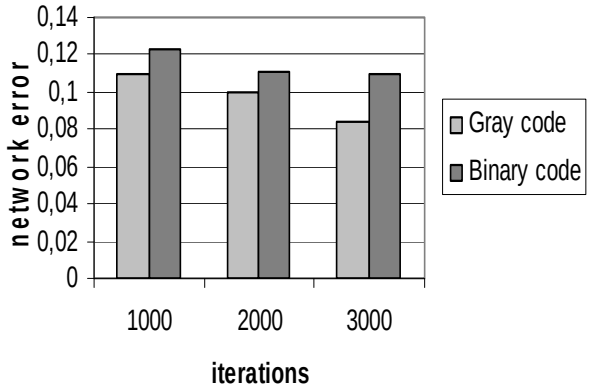

Fig. 4. Comparison of the error values with various input data with steepest descent.

Table 4. Network errors using conjugate gradient algorithm

Iterations	***Network error***	
	Gray code	***Binary code***
1000	0,108043	0,119234
2000	0,109262	0,111098
3000	0,087434	0,110665

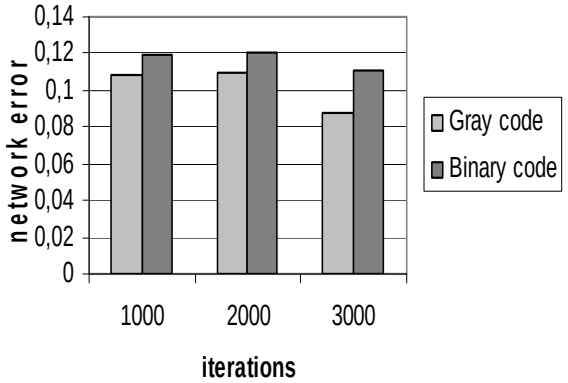

Fig. 5. Comparison of the error values with various input data with conjugate gradient.

6. Conclusions. Each of the tables shows the value of the neural network error using one of considered learning algorithm during the passage of a certain number of iterations for input data, which translated into binary and Gray code to the neural network. The maximum number of iterations is 3000. Charts display the value of a network error when passing iterations.

The following conclusions can be driven:

- we have implemented the program for neural network learning with different learning algorithms, that use different strategies for speedy promotion to the minimum;
- we have found that the use of different learning algorithms, that use different strategies for speedy promotion to the minimum, does not much affect the final prediction result for a sequence that is represent in the binary and Gray code. The prefer algorithm should be chosen depending on the task and the various sets of input data, because the behavior of the algorithms affect a large amount of initial data, their redundancy, fault and other. Well trained network using the steepest descent method and conjugate gradient algorithm. Strategy of choosing momentum in the steepest descent method is key. In conjugate gradient algorithm the key is conjugate coefficient, which contains information about the previous directions;
- it follows, that in research of such problems, a simple algorithm in terms of computing can be chosen.

REFERENCES

1. Hank J.E., Wichern D.W., Reitsch A.G. *Business forecasting.* 7th edition Upper Saddle River, Pearson Prentice Hall, NJ. – 2009. – 498 p.
2. Thomakos D.D., Nave G.J. ARIMA, Nonparametric Transfer Function and VAR Models: a Comparison of Forecasting Performance *Intern. J. Forecast*. – 2004. – N 20. – P. 53–67.
3. Makridakis S., Hibon M. The M3_Competition: Results. Conclusions and Implications *Intern. J. Forecast.* – 2000. – N 16. – P. 451–476.
4. Stock J.H., Watson M.W. *A Comparison of Linear and Non-Linear Univariate Models for Forecasting Macroeconomic Time Series* NBER WP. No. 6607. – 1998.
5. Baestaens D., Van den Bergh W. *Neural Network Solutions for Trading in Financial Market.* Pitman Publ. Inc. Marshfield, MA, USA. – 1994. – 288 p.
6. Callan R. The essence of neural network. *Prentice Hall Europe*. – 1999. – 248 p.
7. Mykytenko N., Sedov Ye. Some particular cases of MLFF networks modelling *Copmputer Model. and New Technol*. – 2011. – v. 15, N 4. – P.28–34.
8. Haykin S. *Neural Networks: a Comprehensive Foundation*. 2nd ed. Prentice-Hall, Englewood Cliffs, NJ. – 1998. – 842 p.
9. Heaton J. *Introduction to Neural Networks for C#*, 2nd ed. Heaton Research, Incorp. – 2008. – 428 p.