TRAING ALGORITHM FOR FORECASTING MULTILAYER NEURAL NETWORK

N. Maniakov, L. Makhnist, V. Rubanov

Department of High Mathematics, Brest State Technical University, Moskovskaja str., 267, Brest, 224017, Belarus, e-mail: *nvmaniakov@bstu.by*

Abstract – This paper discusses one method of building a forecasting neural network, based on dynamical properties of time series. Is explained its construction on basis of theorem of embedding. For this type of feed-forward multilayer neural network proposed recurrent training methods and its matrix algorithmization, which is very helpful in its program realization.

1. Introduction

One of the most important tasks in time series analyses is a prediction of the next points in to the long time interval. For the most real data this problem is unsolvable. That is because of it chaotic behavior. Such time series are highly dependent on initial condition. But another problem arise. Could we understand dynamics of such series? This problem is also very important. When we understand dynamic of motion, we can define domain of attraction of this time series like a pattern of tendency. For the chaotic series it is a stranger attractor, for stochastic – irregular object, for relaxation oscillations – fixed point and so on. So we can understand a tendency and recognize time series behavior.

For this purpose besides linear statistic methods special attention merit nonlinear. In this paper we introduce approach based on heterogeneous nonlinear feed-forward neural networks.

Let us see multilayer heterogeneous neural network [3], which consist of N neural blocks (Fig.1). Any neural block has the structure presenting in Fig.2.

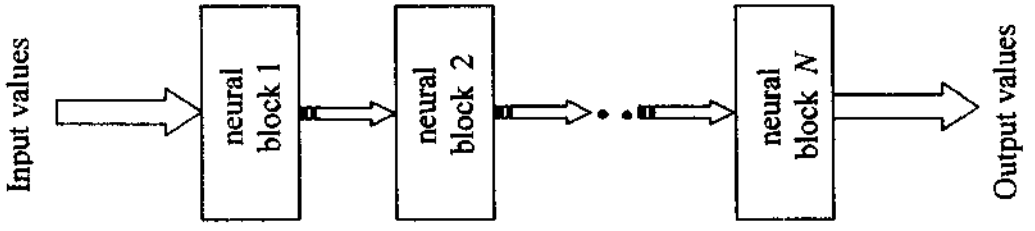

Fig.1 Multilayer neural networks block's representation

The input values for each block are output values of previous neural block; and for the first block – sequence of input figures $\overline{x^k} = \left(x_1^k, ..., x_{m_0}^k\right), \left(k = \overline{1, L}\right)$. Output values of neuron i_n from the block n for element k of taught sequence is defined by recurrent relation:

$$y_{i_n}^{(n),k} = F_n\left(S_{i_n}^{(n),k}\right), \tag{1}$$

where

$$S_{i_n}^{(n),k} = \sum_{i_{n-1}=1}^{m_{n-1}} w_{i_{n-1}i_n}^{(n)} y_{i_{n-1}}^{(n-1),k} - T_{i_n}^{(n)},\ i_n = \overline{1,m_n},\ k = \overline{1,L}. \quad (2)$$

In this way we receive vector $Y^{(n),k} = \left(y_1^{(n),k} \quad y_2^{(n),k} \quad \text{K} \quad y_{m_n}^{(n),k} \quad -1\right)^T$.

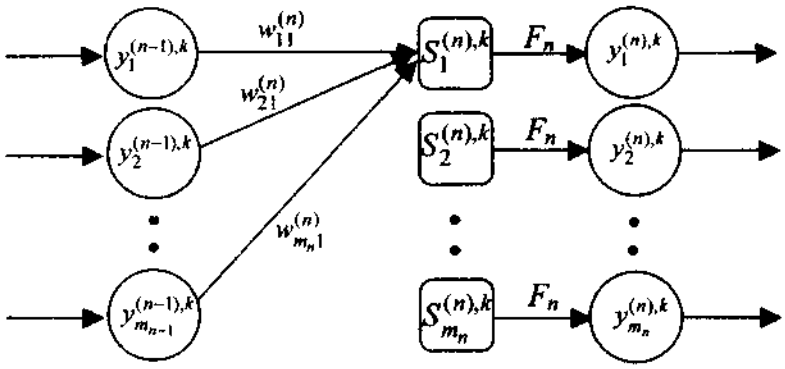

Fig.2 Architecture of neural network block's.

The task of neural network training consist in finding the matrixes of weights

$$W^{(n)} = \begin{pmatrix} w_{11}^{(n)} & w_{21}^{(n)} & \text{K} & w_{m_{n-1}1}^{(n)} \\ w_{12}^{(n)} & w_{22}^{(n)} & \text{K} & w_{m_{n-1}2}^{(n)} \\ \text{K} & \text{K} & \text{K} & \text{K} \\ w_{1m_n}^{(n)} & w_{2m_n}^{(n)} & \text{K} & w_{m_{n-1}m_n}^{(n)} \end{pmatrix}_{m_n \times m_{n-1}}$$

and vectors of thresholds $\overline{T^{(n)}} = \left(T_1^{(n)}, T_2^{(n)}, \ldots, T_{m_n}^{(n)}\right)^T$, $n = \overline{1,N}$, which minimize some network error E_S. We take a mean-square error as a network error $E_S = \frac{1}{2L}\sum_{k=1}^{L}\sum_{i_N=1}^{m_N}\left(y_{i_N}^{(N),k} - t_{i_N}^{k}\right)^2$.

Based on gradient descent method the changes of weights and thresholds are executed in compliance with formulas:

$$w_{j_{n-1}j_n}^{(n)}(t+1) = w_{j_{n-1}j_n}^{(n)}(t) - \alpha^{(n)} \frac{\partial E_s(t)}{\partial w_{j_{n-1}j_n}^{(n)}}, \quad (3)$$

$$T_{j_n}^{(n)}(t+1) = T_{j_n}^{(n)}(t) - \alpha^{(n)} \frac{\partial E_s(t)}{\partial T_{j_n}^{(n)}}. \quad (4)$$

2. Matrix algorithmization of multilayer neural network training process

By using of gradient descent method for multilayer neural network training we receive very complicated formulas, like when we use back-propagation method [7]. So one of the important tasks in training algorithm development consist in finding of easiest ways,

but of cause, without convergence time loss. We proposed matrix algorithmization of training algorithm, which is very helpful in programming realization, especially in such programming environment like MatLab and other.

Theorem Modifications of synaptic connection in multilayer heterogeneous neural network (Fig.1) are produced accordance to the formulas:

$$w^{(n)}_{j_{n-1}j_n}(t+1) = w^{(n)}_{j_{n-1}j_n}(t) - \alpha^{(n)} \cdot \frac{1}{L} \cdot \sum_{k=1}^{L} C^{(n)} \cdot M^{(n)}_{j_n j_{n-1}} \cdot Y^{(n-1),k}, \tag{5}$$

$$T^{(n)}_{j_n}(t+1) = T^{(n)}_{j_n}(t) - \alpha^{(n)} \cdot \frac{1}{L} \cdot \sum_{k=1}^{L} C^{(n)} \cdot M^{(n)}_{j_n(m_{n-1}+1)} \cdot Y^{(n-1),k}, \tag{6}$$

where $C^{(n)}$ calculate recurrently:

$$C^{(n)} = C^{(n+1)} \cdot W^{(n+1)} \cdot MF_n, \; C^{(N)} = \varepsilon^k \cdot MF_N, \tag{7}$$

$$\varepsilon^k = \left(\left(y^{(2),k}_1 - t^k_1 \right) \; \left(y^{(2),k}_2 - t^k_2 \right) \; \text{K} \; \left(y^{(2),k}_{m_2} - t^k_{m_2} \right) \right), \tag{8}$$

and $MF_n = \begin{pmatrix} F_n'\left(S^{(n),k}_1\right) & 0 & \text{K} & 0 \\ 0 & F_n'\left(S^{(n),k}_2\right) & \text{K} & 0 \\ \text{K} & \text{K} & \text{K} & \text{K} \\ 0 & 0 & \text{K} & F_n'\left(S^{(n),k}_{m_n}\right) \end{pmatrix}$ - are $m_n \times m_n$ matrixes, $M^{(n)}_{j_n j_{n-1}}$ - are $m_n \times (m_{n-1}+1)$ matrixes consisting of zero elements with only one element in position $j_n j_{n-1}$ with value equal to one.

Synaptic connection changes begin from the last layer down to first.

Proof Let us calculate the gradient of network error for element k of taught sequence:

$$\frac{\partial E^{(k)}_s}{\partial w^{(n)}_{j_{n-1}j_n}} = \frac{\partial \left(\sum_{i_N=1}^{m_N} \frac{1}{2} \left(y^{(N),k}_{i_N} - t^k_{i_N} \right)^2 \right)}{\partial w^{(n)}_{j_{n-1}j_n}} = \sum_{i_N=1}^{m_N} \left(y^{(N),k}_{i_N} - t^k_{i_N} \right) \cdot \frac{\partial y^{(N),k}_{i_N}}{\partial w^{(n)}_{j_{n-1}j_n}} =$$

$$= \sum_{i_N=1}^{m_N} \left(y^{(N),k}_{i_N} - t^k_{i_N} \right) \cdot F_N'\left(S^{(N),k}_{i_N} \right) \cdot \frac{\partial S^{(N),k}_{i_N}}{\partial w^{(n)}_{j_{n-1}j_n}} =$$

$$= \sum_{i_N=1}^{m_N} \left(y^{(N),k}_{i_N} - t^k_{i_N} \right) \cdot F_N'\left(S^{(N),k}_{i_N} \right) \cdot \sum_{i_{N-1}=1}^{m_{N-1}} w^{(N)}_{i_{N-1}i_N} \cdot \frac{\partial y^{(N-1),k}_{i_{N-1}}}{\partial w^{(n)}_{j_{n-1}j_n}} =$$

$$= \sum_{i_N=1}^{m_N} \left(y^{(N),k}_{i_N} - t^k_{i_N} \right) \cdot F_N'\left(S^{(N),k}_{i_N} \right) \cdot \sum_{i_{N-1}=1}^{m_{N-1}} w^{(N)}_{i_{N-1}i_N} \cdot F_{N-1}'\left(S^{(N-1),k}_{i_{N-1}} \right) \cdot \frac{\partial S^{(N-1),k}_{i_{N-1}}}{\partial w^{(n)}_{j_{n-1}j_n}} =$$

$$= \ldots =$$

$$= \sum_{i_N=1}^{m_N} \left(y^{(N),k}_{i_N} - t^k_{i_N} \right) \cdot F_N'\left(S^{(N),k}_{i_N} \right) \cdot \sum_{i_{N-1}=1}^{m_{N-1}} w^{(N)}_{i_{N-1}i_N} \cdot F_{N-1}'\left(S^{(N-1),k}_{i_{N-1}} \right) \cdot \ldots \cdot$$

$$\cdot \ldots \cdot \sum_{i_n=1}^{m_n} w^{(n+1)}_{i_n j_{n-1}} \cdot F_n'\left(S^{(n),k}_{i_n} \right) \cdot y^{(n-1),k}_{j_{n-1}} \cdot \delta^{i_n}_{j_n} =$$

$$= \varepsilon^k \cdot MF_N \cdot W^{(N)} \cdot MF_{N-1} \cdot \ldots \cdot W^{(n+1)} \cdot MF_n \cdot M^{(n)}_{j_n j_{n-1}} \cdot Y^{(n-1),k} =$$

$$= C^{(n)} \cdot M^{(n)}_{j_n j_{n-1}} \cdot Y^{(n-1),k}, \qquad (9)$$

where

$$C^{(n)} = C^{(n+1)} \cdot W^{(n+1)} \cdot MF_n, \; C^{(N)} = \varepsilon^k \cdot MF_N. \qquad (10)$$

Because of $\frac{\partial E_s}{\partial z} = \frac{\partial\left(\frac{1}{L}\sum_{k=1}^{L} E_s^{(k)}\right)}{\partial z} = \frac{1}{L}\sum_{k=1}^{L}\frac{\partial E_s^{(k)}}{\partial z}$, we can write formulas for synaptic weights changes $w^{(n)}_{j_{n-1}j_n}(t+1) = w^{(n)}_{j_{n-1}j_n}(t) - \alpha^{(n)}\frac{\partial E_s}{\partial w^{(n)}_{j_{n-1}j_n}}$, $j_{n-1} = \overline{1, m_{n-1}}$, $j_n = \overline{1, m_n}$ like:

$$w^{(n)}_{j_{n-1}j_n}(t+1) = w^{(n)}_{j_{n-1}j_n}(t) - \alpha^{(n)} \cdot \frac{1}{L} \cdot \sum_{k=1}^{L} C^{(n)} \cdot M^{(n)}_{j_n j_{n-1}} \cdot Y^{(n-1),k}. \qquad (11)$$

In similar way we can receive formulas for network thresholds changes. ■

Using this theorem we can algorithmize process of multilayer neural network training by reducing modification of synaptic connection to the matrix operations.

3. Multilayer neural networks in modeling of nonlinear dynamics

Neural networks receive wide spread occurrence because of its universal approximation properties [1, 2, 5]. And most of its implementation reduces to using of this property. In construction of forecasting neural networks we also use this.

Suppose we have one dimensional time series *x(t)*. Our basic aim will consist in building such neural networks, which approximate dynamics of this series. This is necessary for finding its tendency. For solution of this problem we built two (three)-layer nonlinear neural networks with m_0 neural units in input layer and only one in output.

The process of forecasting by this type neural network is based on slide window technique. It means, that feeding on to the network input sequence $x(t), x(t+\tau), \ldots, x(t+(m_0-1)\tau)$ we must receive $x(t+m_0\tau)$ in output for consecutively increasing time from $t = 1$. And the training of such network consists in correct choice of synaptic connection for getting in output value closed to the aim. For that purposes we suggest to use algorithm presented above. But we must take particular case with $m_N = 1$.

The mathematical explanation of such network type is next. Accordance to the Takens theorem infinite time series can be embed in to the $\boldsymbol{R}^N$, where $N = 2[d]+1$ ([.] – define real part of number and d – is a fractal dimension of series). Under general condition we can use only limited time series. After building such embedding we receive coordinates of our series like a points $(x(t), x(t+\tau), \ldots, x(t+(N-1)\tau))$ in phase lag-space. All this point will lie on to the smooth manifold; in which by knowing *(N-1)* coordinates we can define exactly one residuary coordinate. So the problem of time series forecasting reduces to approximation of surface, which is given by mapping

$F: R^N \to R: (x(t), x(t+\tau), \ldots, x(t+(N-2)\tau)) \to x(t+(N-1)\tau)$. And for this purpose is very helpful multilayer neural network with $m_0 \geq N-1$ neurons in input layer.

For finding appropriate dimension of embedding we can use any of similar method for chaotic time series: false nearest neighbor, directly Takens theorem or method based on PCA [4]. And for finding time delay τ could be used autocorrelation function or function of mutual information [4].

Used this technique we receive a good result for the chaotic time series like for other. Dealing with stochastic series we couldn't find embedding dimension, because it tend to infinity.

Of cause we don't have a very good forecast because of neural network errors, but we can almost precisely describe dynamics and a tendency of time series.

For more exactly prognosis we may increase number of element in hidden layers (it lowest border for attainment some network error established in [6]), but it cost a training time. So we must make a compromise.

4. Conclusion

We proposed a technique on basis of neural networks, which is helpful in forecasting of different types time series. For the constructed network proposed efficient training method and its matrix algorithmization, which is simple in notation and programming.

5. References

[1] Cybenco G. Approximation by superposition of sigmoidal function // Mathematics of Control, Signals and Systems, 1989, vol. 2, pp. 303-314.

[2] Funahashi K.-I. On the approximate realization of continuous mapping by neural networks // Neural Networks, 1989, vol. 2, pp. 183-192.

[3] Gladkij L., Golovko V. and Makhnist L. Neural network training with use of method of fastest descent // Vestnik BGTU, 2001, vol.5 (11), pp. 56-61. (in Russian)

[4] Golovko V., Savitsky Y. and Maniakov N. Modeling Nonlinear Dynamic using Multilayer Neural Networks // Proceedings of the Workshop Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications. (IDAACS'2001), Foros, Ukraine, July 1-4 2001, pp.197-202.

[5] Hornik K., Stinchcombe M. and White H. Multilayer feedforward networks are universal approximators // Neural Networks, 1989, vol.2, pp. 359-366.

[6] Kurkova V. Approximation of functions by perceptron networks with bounded number of hidden units. // Neural Networks, 1995, vol.8(5), pp.745-750.

[7] Rumelhart D.E., Hinton G.E., Williams R.J. Learning Internal Representations by Error Propagation, In Parallel Distributed Processing, vol.1, chapter 8, Rumelhart D.E. and McClelland J.L., Eds., Cambridge, MA, M.I.T. Press, 1986.