Professor Gheorghe RUXANDA, PhD
Department of Economic Cybernetics
The Bucharest Academy of Economic Studies

LEARNING PERCEPTRON NEURAL NETWORK WITH BACKPROPAGATION ALGORITHM

Abstract. *In this paper, the multilayer perceptron neural network is described and the architecture, performances and the possibilities of using it to solve certain concrete problems are analyzed. At the same time, the backpropagation algorithm of learning multilayer perceptron neural network is described in detail and its software implementation in Eviews 6 language is presented. There are also analyzed the specific problems related to the software implementation of the backpropagation algorithm and there are presented some numerical results obtained with the aid of software implementation, respectively the use of neural network for simulating XOR circuit, for calculating the probability density χ^2 and for the prediction of exchange rate time series.*
Keywords: *neural network, artificial intelligence, multilayer perceptron, backpropagation algorithm, prediction with neural network, supervised pattern recognition, discriminant analysis.*

JEL classification: C45, C53, C63, C87, E17, G17

1. INTRODUCTION

In their majority, the scientific interests and approaches in the field of artificial intelligence have as goal the attempt to reproduce the informational and decision-making processes taking place in the human brain. The human brain's capacity to abstract, to make generalizations and to deduce principles and laws, to know, to order and to classify by association, to make accurate and correct informational identifications on the basis of certain fuzzy and approximate criteria in the conditions of the existence of some uncertain, partial information, to learn both theoretically and practically, to store a huge amount of extremely varied information, has always inspired and influenced both the general modalities of approaching, developed by humanity in order to know materially and spiritually and the creation, the development and the improvement of procedures, instruments and techniques in the field of scientific knowledge.

One of the most interesting characteristics of the human reasoning is that related to the stability and robustness of brain activity, even in the conditions of information powerful distortion or informational penury. The neural mechanisms have the

extraordinary possibility to reconstruct in its integrality a certain informational context, only on the basis of some partial, distorted, collateral or indirect information.

As a result of the attempts to develop methods and techniques of knowledge which to reproduce the informational and decision-making mechanisms taking place at human brain level, a new important and wide field of human knowledge, the *artificial intelligence* has been developed. A prime result of the initial interests in the field of artificial intelligence was concretized under the form of *expert systems*. The expert systems are logical-symbolist models which try to simulate the sequence of reasoning that the human brain develops when it makes a certain decision or when it solves a certain problem. The expert systems which include a set of elements, knowledge, procedures, rules and principles analogously to the human reasoning have as purpose to make certain decisions or to deduce certain responses. To make decisions or to formulate responses by an expert are based on a sequence of analogies, associations and deductions and suppose to develop certain complex inferential procedures and to perform certain symbolic calculations. Although, the expert systems development has recorded a series of successes in certain fields as for instance, medicine, over time, we have noticed that the expert systems represent a too fuzzy and too poor approximation of the complex and subtle processes and mechanisms which characterize the human reasoning. In spite all the advantages related to the fact that an expert system refers only to a segment strictly limited by reality, the expert system remains more than a partial response of the human thinking mechanisms.

Another approaching direction in the field of artificial intelligence is that based *on artificial neural networks.* So far, this approaching manner proves to be more effective and closer to the mode in which the human brain works, compared with the direction represented by expert systems. Both the analogy between artificial neural networks and the human thinking mechanism, as well as the fact that the artificial neural networks are able to reproduce the human intelligence, result from the fact that neural network has a structure much more closer to the biologic architecture of the human brain.

In the last decades, the artificial neural network concept based approach has known a powerful development, more and more frequently, numerous problems in the field of classification, control or prediction started to be solved with the aid of neural calculation based techniques. The applications of neural networks are met in large variety fields, as for instance, economics, finances, informatics, medicine, engineering, biology etc., and the results obtained in case of neural calculation based approaches are really spectacular. Both the development of the neural calculation techniques and the their extended use in more and more varied fields are mostly due to the simplicity of these instruments and to the efficiency of their utilization.

The neural networks are powerful and effective instruments of modeling, used to solve problems of a large complexity, problems which often cannot be solved satisfactorily, easily and effectively, by using other methods and techniques. In spite of their power and effectiveness, the neural networks are characterized by extraordinary algorithmic simplicity and by easiness in using them. The neural networks,

are probably, the only modeling instruments which logically can be set from the simplest to the most complex pattern, depending on the nature and the complexity of the approached problem. The capacity of neural to be set according to the approached problem nature provides solving possibilities for the most varied types of problems, creating in this way modeling instruments with an almost universal feature.

The neural techniques power is also due to the fact that, by their structure and functionality, the neural networks can be assimilated to *non-linear models.* Consequently, in neural modeling, is no longer necessary to state the simplifying hypothesis and to accept certain more or less realistic approximations, as in the case of linear modeling. In nonlinear modeling of the traditional approaches, the models are represented by means of a large number of algebraic or differential equations, whose construction supposes a laborious process of analyzing and which arises numerous difficulties concerning the analytic or numerical solving. Unlike the traditional nonlinear modeling, the neural modeling is characterized by flexibility and naturalness, the neural model construction is very simple and requires a minimal effort and problems solving is reduced to perform certain elementary numerical calculations.

Due to neural network simplicity, the inherent difficulties of the complexity problem within the traditional nonlinear modeling are powerfully diminished or even eliminated in case of modeling with the aid of neural networks. Another important advantage resulting from neural models simplicity consists in the fact that modeling with the aid of neural networks does not require a very high level of user' s scientific training in the field of mathematical modeling. From this point of view, we may say that the access to use the neural techniques is much easier, compared with any other methods and techniques of statistical-mathematical modeling.

Maybe, the most important feature of neural networks that distinguished them fundamentally from other modeling instruments is that of dynamic adaptability, of their ability to learn, to self-train. The neural networks have the ability *to learn* from primary data information that define the inputs of these networks. The learning processes within the neural networks are defined by means of certain specific algorithms, *called learning algorithms*, and which allow neural networks to learn to recognize, better and better, the informational structures contained in an invisible pattern of input data.

The beginning of neural model concept based approach is recorded since 1943 year, with the appearance of the first neuron model, model proposed by neurophysiologist W.S. McCulloch and by mathematician W. Pitts. A special interest for neural model was noticed especially after the publication of the first papers concerning the mathematical modeling of learning process. In 1947 year, D.O. Hebb opened unsuspected directions in the field of neural calculation. In 1957 year, another important step was made when Frank Rosenblatt's work (Cornell Aeronautical Laboratory) was published, work dedicated to a simplified neural model of probabilistic nature, known under the name of *perceptron.* We can also mention the appearance of neural model *ADALINE* (**ADA**ptive **L**inear **N**euron), model developed in the early 60's by B. Widrow.

The *backpropagation algorithm* of learning a multilayer perceptron neural network (MLP – **M**ulti**L**ayers **P**erceptron) was described for the first time in 1969 year by Arthur E. Bryson and Yu-Chi Ho. With the version created in 1986 year by David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams, the backpropagation algorithm has gained notoriety and interest to neural network field.

The artificial neural networks have been created in order to try to reproduce the neurophysiologic mechanisms taking place at the level of the human brain. By combining a very large number of calculation units, called *neurons,* under the form of some very high degree interconnecting systems, we try to reproduce certain complex phenomena, specific to human brain, as for instance, retention and intelligence.

Regardless the fact that the neural networks can or cannot be considered to be endowed effectively with "intelligence" , they are very useful for activities of analysis and prediction, as statistical models. Generally, the neural networks can be considered as representing the classes of regression models, of discriminant analysis or of reducing the dimensionality. As with neural networks we can detect data relationships and complex structures, they can be also used as prediction instruments for many categories of real phenomena.

The neural networks can be used to solving some complex problems of very varied fields: cluster analysis, supervised pattern recognition (human speech recognition, images recognition), automatic translation of different languages, detection of wireless systems position, bankruptcy prediction, time series prediction etc.

Compared with other approaching manners, the neural networks used to solve certain complex problems provides a series of important advantages. For instance, the advantage provided by using the neural networks to make predictions in the field of time series results from the fact that, for this purpose, it is not strictly necessary to state a certain explicit mathematical model, that to describe the links among an input's set and an output's set. On the other hand, the neural network based approach is indicated in many situations, as a result of the fact that, most times, the prediction activity is a more important activity from a scientific point of view, than the explanation of the links among phenomena. Although, in principle, the prediction activity supposes to know and to formalize the causality links, in neural modeling case, the predictions can be made without a prior knowledge of the causality links.

The most important category of neural networks is that represented by "*feed-forward"* networks. In case of learning or training this type of neural networks, a specific neural process called "*backpropagation*" is defined. It refers to the mode in which the gradient of error function is calculated, according to which, the synaptic weights updating of neural networks takes place. *Standard backpropagation* is, in fact, an alternative name for the rule known in neural networks theory under the name of "*Generalized Delta Rule*", that is a learning technique suggested by Rumelhart, Hinton and Williams in 1986 year and that is the most spread supervised learning technique of neural networks. There are two variants of standard backpropagation: *batch backpropagation* and *incremental backpropagation*.

Batch backpropagation is an optimization technique where the weights updating is based on the cyclic use of the entire learning set. Unlike the *batch* backpropagation, the *incremental* backpropagation is based on weights updating after the utilization of each pattern of the learning set.

2. GENERAL STRUCTURE AND THE FUNCTIONALITY OF A NEURAL NETWORK

Neural network is a set of processing elementary units, called *neurons* or *nodes,* whose processing capability is stored at the level of neural connections, under the form of *synaptic weights*, as a the result of an iterative process of adaptation, according to the information of a sample, called *learning set* or *training set* of neural network.

A neural network is composed of two fundamental elements: a set of *functional units* called *neurons* and a set of *connections* among neurons called *synaptic connections.* The fundamental element of any neural network is the *artificial neuron*, whose functionality is shown in the figure below.

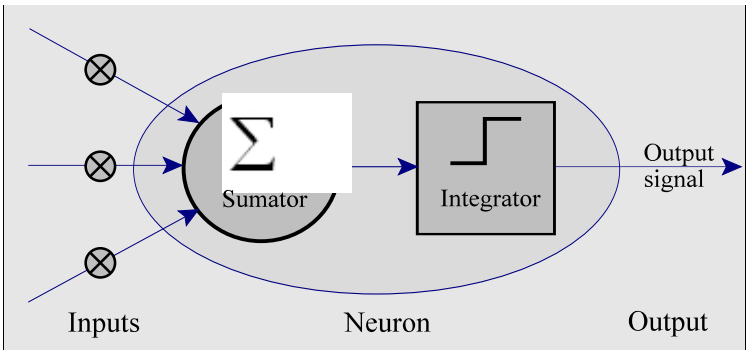

Figure 1: **Functional structure of a neuron**

The neural network neurons have different functions, they are specialized to make certain types of activities. From this point of view, a neural network contains three basic types of neurons: *input* neurons which collect the values of input variables or the standardized values of input variables which form the so-called *input layer*; *intermediate* neurons or *hidden* neurons, which are neurons located between the input layer and the output layer, whose functionality is purely to perform nonlinear calculations and are organized under the form of one or more layers, called *hidden* layers; *output* neurons calculating the predicted values with the aid of neural network and which compare these values with certain *target values* or *reference values.* According to the comparison result, the synaptic link weights are or are not updated.

Each basic unit of a neural network, respectively, each neuron, has an internal state and an output. Neuron functionality consists in the fact that it produces a single output, represented by a single numeric value, depending on the nature or the state of the respective unit, state determined according to input information of the respective

neuron. The network neurons behavior can be characterized with the aid of an output activation mechanism, of the general form:

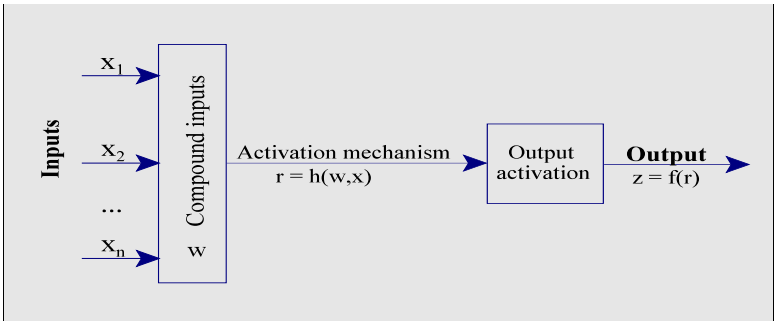

Figure 2: Structure and orientation of a general neuron

In the figure above, the function *h*(.) is called the *compound function* and defines the adding of inputs under the form of a magnitude whose value can determine the output activation, i.e. a neuron to produce an output. The inputs contain a series of parameters, denoted by $w_1, w_2, ..., w_n$, which are called *synaptic weights* and which weight the neurons input from different layers of neural network. As the compound function of a neuron has in most cases a linear form, the activation part of a neuron, related to the receive of inputs is known under the name of linear component. Function *f*(.) is called *activation function* and defines the formation of neuron output, according to the achieved inputs composition level.

The internal state of a neuron, denoted by *j* is achieved by adding the weighted inputs of this neuron, adding described by a relation of the form:

$$s_j = w_{0j} + \sum_{i=1}^{n} w_{ij} \cdot x_i,$$

where value w_{0j} is called the *neuron excitation threshold*. In case when the threshold w_{0j} is included in the input values, the internal state or the neuron potential can be described also under the form:

$$s_j = \sum_{i=0}^{n} w_{ij} \cdot x_i,$$

where $x_0 = 1$ is considered a fictitious input.

If we shall denote by **x** the possible inputs vector of a generic neuron denoted by *j* including also the fictitious input x_0 and by $w^{(j)}$ the extended vector of the *n+1* weights of neuron inputs *j*, namely:

$$x = \begin{pmatrix} x_0 \\ x_1 \\ \\ x_n \end{pmatrix}; \qquad w^{(j)} = \begin{pmatrix} w_{0j} \\ x_{1j} \\ \\ x_{nj} \end{pmatrix}, \qquad j = 1,2,...,n,$$

then, the neuron internal state *j* could be written under the form of dot product:

$$s_j = \left(\mathbf{w}^{(j)}\right)^t \cdot \mathbf{x}.$$

Assuming that besides neurons in input layer, the neural network includes also a number of *n* neurons, the internal state of all *n* neurons of neural network can be written under the following matrix form:

$$\begin{pmatrix} s_1 \\ s_2 \\ \cdots \\ s_j \\ \cdots \\ s_n \end{pmatrix} = \begin{pmatrix} w_{01} & w_{11} & \cdots & w_{i1} & \cdots & w_{n1} \\ w_{02} & w_{12} & \cdots & w_{i2} & \cdots & w_{n2} \\ \cdots & & & & & \\ w_{0j} & w_{1j} & \cdots & w_{ij} & \cdots & w_{nj} \\ \cdots & & & & & \\ w_{0n} & w_{1n} & \cdots & w_{in} & \cdots & w_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ \cdots \\ x_i \\ \cdots \\ x_n \end{pmatrix}.$$

If we shall consider that the *n* weight vectors, $\mathbf{w}^{(j)}$, j=1,2,...,n, are the columns of a matrix **W** under the form:

$$\mathbf{W}_{(n+1)\times n} = \begin{pmatrix} w_{01} & w_{02} & \cdots & w_{0j} & \cdots & w_{0n} \\ w_{11} & w_{12} & \cdots & w_{1j} & \cdots & w_{1n} \\ \cdots & & & & & \\ w_{i1} & w_{i2} & \cdots & w_{ij} & \cdots & w_{in} \\ \cdots & & & & & \\ w_{n1} & w_{n2} & \cdots & w_{nj} & \cdots & w_{nn} \end{pmatrix}$$

then, the internal state of all *n* network neurons can be written under the condensed form:

$$\mathbf{s} = \mathbf{W}^t \cdot \mathbf{x},$$

where **s** is the vector of the internal states of *n* network neurons.

Further on, we shall denote by $f_j(\cdot)$ the activation or transfer function of the *j*-th neuron, such as its output will be:

$$x_j = f_j(s_j).$$

Assuming that the processing of neuron inputs and the generation of this neuron output imply a certain period of time, we shall achieve a time lag of a neuron input in relation to its inputs. By taking into consideration this lag, the output of the *j*-th neuron can be written as follows:

$$x_j(t+1) = f_j(s_j(t)),$$

known under the name of *evolution equation* of the *j*-th network neuron.

To deduce the *evolution equation* of the entire neural network, we shall denote by $\mathbf{f}(\mathbf{s})$ the vector of transfer functions, defined as follows:

$$\mathbf{f}(\mathbf{s}) = \begin{pmatrix} f_1(s_1(t)) \\ f_2(s_2(t)) \\ \dots \\ f_n(s_n(t)) \end{pmatrix}.$$

Consequently, the *evolution equation* of neural network, namely, the equation describing the modification of neural network states, can be written:

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{W}^t \mathbf{x}(t)).$$

3. BACKPROPAGATION LEARNING ALGORITHM

In the following, we shall define and analyze the most important processes specific to neural networks, respectively, the learning process of a neural network. These processes will be taken into consideration for the specific case of (MLP) *multilayer perceptron* neural networks, the *backpropagation algorithm* being the most known training process. The structure of a perceptron neural network with a single hidden layer is illustrated in the figure below.

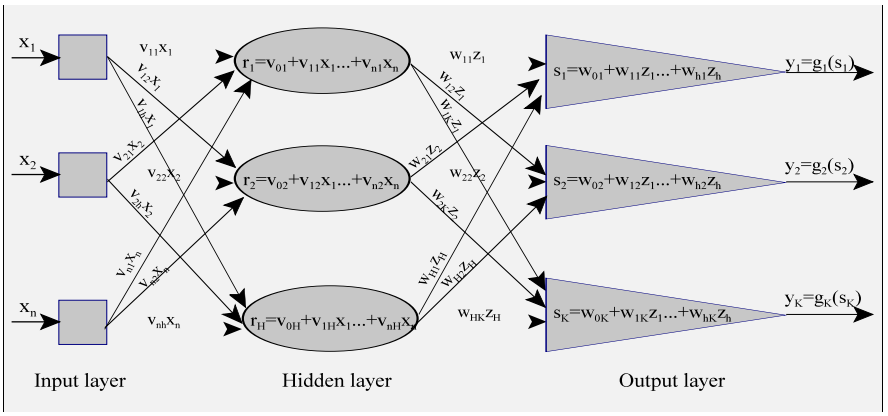

Figure 3: **The structure of "Perceptron" neural network with a hidden layer**

We shall suppose that the sample representing the learning or training set of a *simple perceptron* neural network includes a number of T patterns, represented by n-dimensional vectors set:

$$X = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(T)}\},$$

as well as a number of T reference vectors, K-dimensional, respectively:

$$\tilde{Y} = \{\tilde{\mathbf{y}}^{(1)}, \tilde{\mathbf{y}}^{(2)}, ..., \tilde{\mathbf{y}}^{(T)}\},$$

whose elements are the target value of neural network, playing the part of "*professor*" for the learning process of neural network.

The neural network training supposes some sequence of operations, sequence known under the name of *learning algorithm*, whose aim is to set the network with a view to perform certain specific operations. Currently, learning a neural network is equivalent to an iterative process of *updating its synaptic weights*, such that the

neural network to achieve a certain function, starting from a set of values known aprioristically and called *examples*, set known under the name of *training set or learning set.*

An element of the training set, denoted generically by **x** is called *pattern* or *configuration* and is represented by means of a *n*-dimensional vector, whose elements are the *characteristics* or the *attributes* of the respective pattern. Thus, any pattern **x** appears as being a point in an *n*-dimensional $\mathbb{R}^n$ vector space, called *space of patterns*. From a strictly formal point of view, a neural network can be regarded as being a transformation of an input space $\mathbb{R}^n$, into an output space $\mathbb{R}^m$.

If we shall take the case when the input is represented by pattern $\mathbf{x}^{(t)}$ and the ideal outputs of final neurons are represented by values $\tilde{y}_1^{(t)},\tilde{y}_2^{(t)},...,\tilde{y}_K^{(t)}$, then the *error function*, constructed on the basis of the differences between the real outputs $y_1^{(t)},y_2^{(t)},...,y_K^{(t)}$ and the ideal outputs $\tilde{y}_1^{(t)},\tilde{y}_2^{(t)},...,\tilde{y}_K^{(t)}$ can be defined as follows:

$$E_t = \frac{1}{2}\sum_{k=1}^{K}\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)^2.$$

In this context, the neurons activations in the hidden layer, when the output is represented by pattern $\mathbf{x}^{(t)}$, are defined by relations:

$$r_h^{(t)} = v_{0h}^{(t)} + \sum_{i=1}^{n} v_{ih}^{(t)} x_i^{(t)}, \quad h=1,2,...,H.$$

By considering that the input layer is extended by one more neuron, whose output $x_0^{(t)}$ is always equal to unit, the activations of neurons in the hidden layer can be written under the following condensed form:

$$r_h^{(t)} = \sum_{i=0}^{n} v_{ih}^{(t)} x_i^{(t)}, \quad h=1,2,...,H.$$

At the same time, the outputs of the neurons in the hidden layer are defined by the following relation:

$$z_h^{(t)} = f_h\left(r_h^{(t)}\right) = f_h\left(\sum_{i=0}^{n} v_{ih}^{(t)} x_i^{(t)}\right), \quad h=1,2,...,H,$$

where $f_h(\cdot)$ is the activation function of the *h*-th neuron in the hidden space. Similarly, the activity of the *k*-th neuron in the output layer is defined by relation:

$$s_k^{(t)} = w_{0k}^{(t)} + \sum_{h=1}^{H} w_{hk}^{(t)} z_h^{(t)}, \quad k=1,2,...,K,$$

and the output of the respective neuron is defined by relation:

$$y_k^{(t)} = g_k\left(s_k^{(t)}\right) = g_k\left(w_{0k}^{(t)} + \sum_{h=1}^{H} w_{hk}^{(t)} z_h^{(t)}\right), \quad k=1,2,...,K,$$

where $g_k(\cdot)$ is the activation function of the *k*-th neuron in the output layer.

In order to simplify, we shall consider that the hidden layer contains an additional neuron, whose output $z_0^{(t)}$ is always equal to unit. Under these conditions, the activation of the *k*-th neuron in the output layer can be written:

$$s_k^{(t)} = \sum_{h=0}^{H} w_{hk}^{(t)} z_h^{(t)}, \quad k=1,2,...,K,$$

and the output of the respective neuron can be written:

$$y_k^{(t)} = g_k\left(s_k^{(t)}\right) = g_k\left(\sum_{h=0}^{H} w_{hk}^{(t)} z_h^{(t)}\right), \quad k=1,2,...,K.$$

Taking into consideration the relation that defines the output of a neuron *k* in the hidden layer, the output of the *k*-th neuron in the output layer can be written:

$$y_k^{(t)} = g_k\left(s_k^{(t)}\right) = g_k\left[\sum_{h=0}^{H} w_{hk}^{(t)} f_h\left(\sum_{i=0}^{n} v_{ih}^{(t)} x_i^{(t)}\right)\right], \quad k=1,2,...,K.$$

On the other hand, taking into account how the error function E_t is defined, it results that for a certain given input pattern $\mathbf{x}^{(t)}$, it can be regarded as a function depending on all the synaptic weights, respectively, as a function of the following form:

$$E_t\left(v_{01}^{(t)}, v_{11}^{(t)}, ..., v_{nH}^{(t)}, w_{01}^{(t)}, w_{11}^{(t)}, ..., w_{HK}^{(t)}\right) = \frac{1}{2}\sum_{k=1}^{K}\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)^2 = \frac{1}{2}\sum_{k=1}^{K}\left(g_k\left(\sum_{h=0}^{H} w_{hk}^{(t)} z_h^{(t)}\right) - \tilde{y}_k^{(t)}\right)^2$$
$$= \frac{1}{2}\sum_{k=1}^{K}\left(g_k\left(\sum_{h=0}^{H} w_{hk}^{(t)} f_h\left(\sum_{i=0}^{n} v_{ih}^{(t)} x_i^{(t)}\right)\right) - \tilde{y}_k^{(t)}\right)^2.$$

The updating algorithm of the synaptic weights of a neural network, known under the name of the *backpropagation algorithm*, is based on the following general relations of modifying the weights:

$$w_{hk}^{(t+1)} = w_{hk}^{(t)} + \Delta w_{hk}^{(t)}, \quad v_{ih}^{(t+1)} = v_{ih}^{(t)} + \Delta v_{ih}^{(t)}, \quad h=0,1,...,H;\ k=1,2,...,K;\ i=0,1,...,n,$$

where:

$$\Delta w_{hk}^{(t)} = -c\frac{\partial E_t\left(v_{01}^{(t)}, v_{11}^{(t)}, ..., v_{nH}^{(t)}, w_{01}^{(t)}, w_{11}^{(t)}, ..., w_{HK}^{(t)}\right)}{\partial w_{hk}^{(t)}}, \quad h=0,1,...,H;\ k=1,2,...,K;$$

$$\Delta v_{ih}^{(t)} = -c\frac{\partial E_t\left(v_{01}^{(t)}, v_{11}^{(t)}, ..., v_{nH}^{(t)}, w_{01}^{(t)}, w_{11}^{(t)}, ..., w_{HK}^{(t)}\right)}{\partial v_{ih}^{(t)}}, \quad i=0,1,...,n;\ h=1,2,...,H,$$

and *c* is the *learning constant* or the *learning rate*.

Bearing in mind the form of the error function E_t, the modifications of the weights corresponding to final layer of neurons, can be written as follows:

$$\Delta w_{hk}^{(t)} = -c\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(\sum_{j=0}^{H} w_{jk}^{(t)} z_j^{(t)}\right)}{\partial w_{hk}^{(t)}} = -c\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}}\frac{\partial s_k^{(t)}}{w_{hk}^{(t)}},$$

pattern that, if we take into consideration that $\frac{\partial s_k^{(t)}}{w_{hk}^{(t)}} = z_h^{(t)}$ becomes:

$$\Delta w_{hk}^{(t)} = -c\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}} z_h^{(t)}.$$

Under these conditions, the recurrence relations that define the modification of the weights corresponding to neurons in the final layer, are under the form:

$$w_{hk}^{(t+1)} = w_{hk}^{(t)} - c\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}} z_h^{(t)}, \qquad h=0,1,...,H;\ k=1,2,...,K.$$

Similarly, the modifications of the weights corresponding to hidden layer of neurons, can be written:

$$\Delta v_{ih}^{(t)} = -c\sum_{k=1}^{K}\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(\sum_{j=0}^{H} w_{jk}^{(t)} f_j\left(\sum_{q=0}^{n} v_{qj}^{(t)} x_q^{(t)}\right)\right)}{\partial v_{ih}^{(t)}} = -c\sum_{k=1}^{K}\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}}\frac{\partial s_k^{(t)}}{\partial v_{ih}^{(t)}}$$

$$= -c\sum_{k=1}^{K}\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}}\frac{\partial}{\partial v_{ih}^{(t)}}\left[\sum_{j=0}^{H} w_{jk}^{(t)} f_j\left(\sum_{q=0}^{n} v_{qj}^{(t)} x_q^{(t)}\right)\right]$$

$$= -c\sum_{k=1}^{K}\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}} w_{hk}^{(t)}\frac{\partial f_h\left(\sum_{q=0}^{n} v_{qh}^{(t)} x_q^{(t)}\right)}{\partial v_{ih}^{(t)}}$$

$$= -c\sum_{k=1}^{K}\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}} w_{hk}^{(t)}\frac{\partial f_h\left(r_h^{(t)}\right)}{\partial r_h^{(t)}}\frac{\partial r_h^{(t)}}{\partial v_{ih}^{(t)}},$$

pattern which, if we take into consideration that $\frac{\partial r_h^{(t)}}{v_{ih}^{(t)}} = x_i^{(t)}$, becomes:

$$\Delta v_{ih}^{(t)} = -c\sum_{k=1}^{K}\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}}\frac{\partial f_h\left(r_h^{(t)}\right)}{\partial r_h^{(t)}} w_{hk}^{(t)} x_i^{(t)} = -c\frac{\partial f_h\left(r_h^{(t)}\right)}{\partial r_h^{(t)}} x_i^{(t)}\sum_{k=1}^{K}\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}} w_{hk}^{(t)}.$$

It results that the recurrence relations that define the modification of the weights corresponding to neurons in the intermediate layer, are of the form:

$$v_{ih}^{(t+1)} = v_{ih}^{(t)} - c\frac{\partial f_h\left(r_h^{(t)}\right)}{\partial r_h^{(t)}} x_i^{(t)}\sum_{k=1}^{K}\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}} w_{hk}^{(t)},\ i=0,1,...,n;\ h=0,1,...,H.$$

An improved variant of the backpropagation algorithm, known under the name of backpropagation algorithm with momentum, is that in which tha synaptic weights updating for the output layer and for the hidden layer is defined by the relations:

$$w_{hk}^{(t+1)} = w_{hk}^{(t)} - c\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}} z_h^{(t)} + m\cdot\left(w_{hk}^{(t)} - w_{hk}^{(t-1)}\right);$$

$$v_{ih}^{(t+1)} = v_{ih}^{(t)} - c\frac{\partial f_h\left(r_h^{(t)}\right)}{\partial r_h^{(t)}} x_i^{(t)} \sum_{k=1}^{K}\left(y_k^{(t)} - \tilde{y}_k^{(t)}\right)\frac{\partial g_k\left(s_k^{(t)}\right)}{\partial s_k^{(t)}} w_{hk}^{(t)} + m\cdot\left(v_{ih}^{(t)} - v_{ih}^{(t-1)}\right),$$

where $h=0,1,...,H;\ k=1,2,...,K$, respectively $i=0,1,...,n;\ h=0,1,...,H$, and *m* is an *inertial constant*, called *momentum*. This modified variant of synaptic weights training determines an increase of the algorithm stability and accelerates the convergence process.

4. SOFTWARE IMPLEMENTATION OF THE BACKPROPAGATION ALGORITHM

Software implementation of the backpropagation learning algorithm was performed in Eviews 6 language and aims at learning a one hidden layer perceptron neural network. The input layer number of neurons in the hidden layer and output layer can be set by user according to neural network configuration to be used. At the same time, the user has the possibility to choose the pattern of the neurons activation function in the hidden and output layers.

In order to increase the efficiency of the backpropagation training algorithm, this was "prefaced" with a random generation model of the initial synaptic weights, whose purpose is to choose a system of the appropriate initial synaptic weights, such as the back propagation algorithm to start from a conveniently chosen start point.

The software implementation of backpropagation algorithm includes a number of 7 program modules, respectively:

- **neural_net –** the main program playing the role to set the values of the parameters necessary to the backpropagation algorithm and to manage the subprogram calls;
- **gen_pond_syn** – subprogram playing the role to generate the initial synaptic weights of neural network;
- **propag** – subprogram which aims at propagating a pattern of the learning set along neural network, from the neural network input to its output, as well as to evaluate the error associated to the propagated pattern;
- **learn_net** – subprogram playing the role to modify the synaptic weights of neural network, in concordance with the relations which define the backpropagation algorithm;
- **valid_net** – subprogram which aims validation of the performance of neural network on training set;
- **fct_act** – subprogram including the class of functions which can be used as activation functions for the neurons in the hidden and output layers; this class includes the following functions: *sigmoid, bipolar sigmoid, step and linear*;
- **der_fct_act** – subprogram including the derivatives class of activation functions of neurons in the hidden and output layers.

We shall present in the following annexes the source codes for the 7 software modules which implement the backpropagation algorithm.

4. NUMERICAL EXAMPLES

In order to illustrate how the backpropagation algorithm for learning the perceptron neural network with a single hidden layer is used, we shall consider the following three example:

• **a.** ***implementation of XOR function***

The neural network designed to calculate the values of ***logic function XOR***, has the following structure:

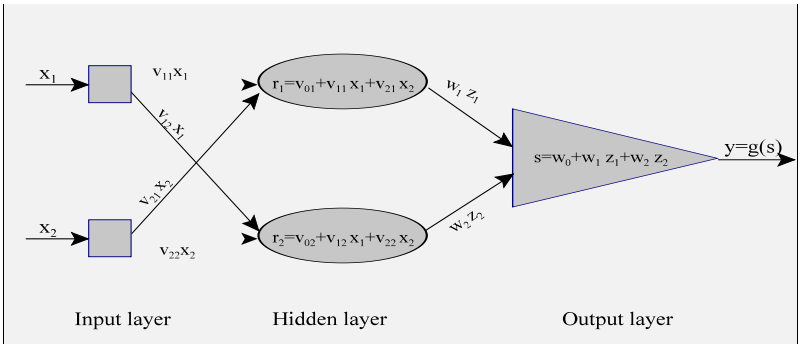

Figure 4: Neural Network Structure Implementing the XOR function

Input context: • *observation number*: **4**; • *number of neurons in the input layer*: **2** neurons; • *number of hidden layer*: **1**; • *number of neurons in the hidden layer*: **2**; • *number of nurons in the output layer*: **1**; • *activation function for the neurons in the hidden layer*: **sigmoid function**; • *activation function for the neuron in the hidden layer*: **sigmoid function**. The weights obtained after running the program are the following ones:

$v_{01} = -10.86300;\quad v_{02} = 9.316429;$
$v_{11} = 14.35190;\quad v_{12} = -6.277778;\quad v_{21} = 14.68031;\quad v_{22} = -8.531974;$
$w_0 = -0.04483;\quad w_1 = 10.940850;\quad w_2 = 1.716078;\quad w_3 = 10.98665.$

Output results: • *number of iteration*: **11**; • *maximum error*: **-0.000299554**; • (*y_pred, y_real*): (**-0.000297389**, **0.000000000**), (**0.999999018**, **1.000000000**), (**0.999999997**, **1.000000000**), (**-0.000299554**, **0.000000000**).

• **b.** ***calculation of the probability density χ^2 with 3 degrees of freedom***

The neural network designed to calculate the values of the probability density χ^2 with 3 degrees of freedom, respectively, the values of function:

$$f(x) = \frac{1}{2^{\frac{k}{2}}\Gamma\left(\frac{k}{2}\right)} x^{\frac{k}{2}-1} e^{-\frac{x}{2}},$$

where *k* is the degrees of freedom number, has the following structure:

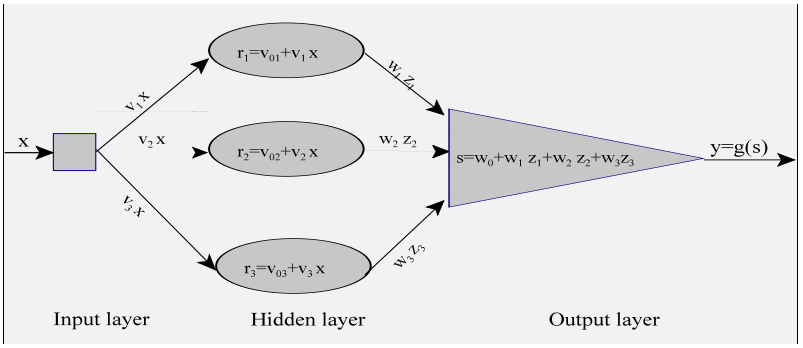

Figure 5: Neural network structure implementing the calculation of probability density χ^2 with 3 degrees of freedom

Input context: • *observation number*: **100** pairs (value x, value of probability density function); • *number of neurons in the input layer*: **1** neuron; • *number of hidden layer*: **1**; • *number of neurons in the hidden layer*: **3**; *number of nurons in the output layer*: **1**; • *activation function for the neurons in the hidden layer*: **bipolar sigmoid function**; • *activation function for the neuron in the hidden layer*: **bipolar sigmoid function**.The weights obtained after running the pro-gram are the following ones:

$v_{01} = 3.060750$; $v_{02} = 2.982753$; $v_{03} = 0.185895$;
$v_1 = 1.906924$; $v_2 = 14.288513$; $v_3 = -0.452750$;
$w_0 = -6.749069$; $w_1 = 5.693702$; $w_2 = 1.716078$; $w_3 = 0.660735$.

Output results: • *number of iteration*: **110**; • *maximum error*: **0.000684951**; • *maximum percent of error*: **0.42240%**; • *mean square error*: **0.00014642**; • *mean absolute error*: **0.000084957**; • *real and predicted values graphics*:

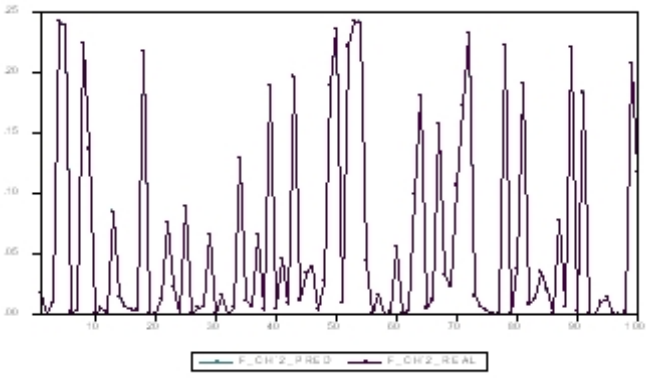

Figure 6: Real χ^2 probability density and Predicted χ^2 probability density

• **c.** ***leu/euro exchange rate prediction***

The neural network designed to predict the leu/euro exchange rate, according to a relation of the form:

$$ex\_rate_t = h\left(ex\_rate_{t-1}, ex\_rate_{t-2}\right) + \varepsilon_t,$$

has the following structure:

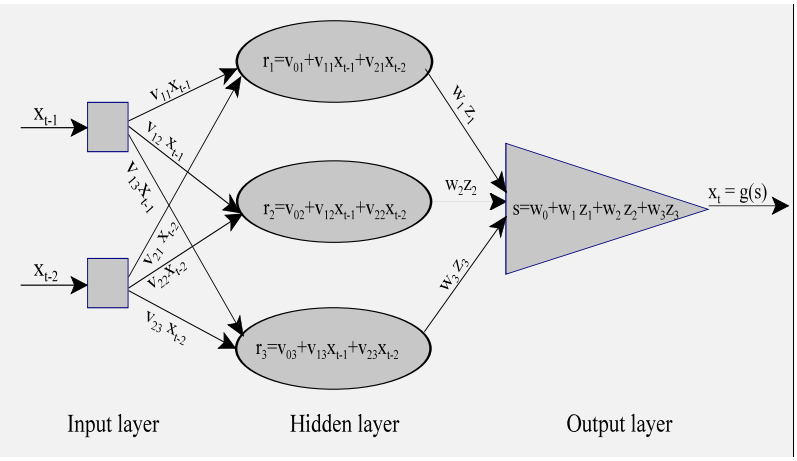

Figure 7: Neural network structure implementing exchange rate prediction

Input context: • *observation number*: **231** (exchange rate leu/euro 01/04/2010 - 11/30/2010); • *number of neurons in the input layer*: **2** neurons; • *number of hidden layer*: **1**; • *number of neurons in the hidden layer*: **3**; *number of nurons in the output layer*: **1**; • *activation function for the neurons in the hidden layer*: **bipolar sigmoid function**; • *activation function for the neuron in the hidden layer*: **identic function.** The weights obtained after running the program are the following ones:

$v_{01} = -35.101629;\quad v_{02} = 11.671472;\quad v_{03} = 24.234089;$

$v_{11} = -2.495037;\quad v_{12} = 3.588576;\quad v_{13} = 6.227018;$

$v_{21} = 10.864618;\quad v_{22} = 3.632890;\quad v_{23} = 6.484894;$

$w_0 = 1.049475;\quad w_1 = 0.247868;\quad w_2 = 1.716078;\quad w_3 = 1.428334.$

Output results: • *number of iteration*: **357**; • *maximum error*: **0.045892136**; • *maximum percent of error*: **1.0720458%**; • *mean square error*: **0.0118426**; • *mean absolute error*: **0.0091453**; • *real and predicted values graphics*:

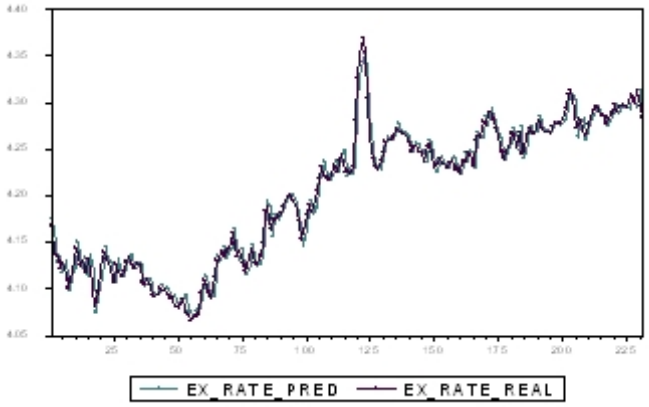

Figure 8: Real exchange rate and Predicted exchange rate

REFERENCES

[1] **Bassi D.** (1995), ***Stock price predictions by recurrent multiplayer neural network architectures***, in Proc. Neural Networks in the Capital Markets Conf., London, U.K., October 1995

[2] **Bishop C.M.** (1995), ***Neural Networks for Pattern Recognition*** (1995), Oxford Univ. Press, 1995

[3] **Fausett L.**, (1994), ***Fundamentals of Neural Networks: Architecture, Algorithms and Applications***, Prentice-Hall, New Jersey, USA, 1994

[4] **Gallant S. I.** (1990), ***Perceptron-based learning algorithms***, IEEE Transactions on Neural Networks, vol. 1, no. 2, 1990

[5] **Minsky M. L.**, **Papert S. A.** (1969), ***Perceptrons***, MIT Press, Cambridge MA, USA, 1969

[6] **Ripley B. D.** (1996), ***Pattern Recognition and Neural Networks***, Cambridge University Press, Cambridge, UK, 1996

[7] **Ruxanda Gh.** (2006), ***Economic and Financial Analysis and Prediction with Neural Network***, in review *Studii şi Cercetări de Calcul Economic şi Cibernetică Economică*, no. 4/2006, Academy of Economic Studies, Bucharest, 2006

[8] **Ruxanda G.** (2007), ***Supervised pattern recognition with iterative methods***, review *Studii şi Cercetări de Calcul Economic şi Cibernetică Economică*, no. 1/2007, Academy of Economic Studies, Bucharest, 2007

[9] **Ruxanda Gh.** (2008), ***Unsupervised Pattern Recognition with Partitioned Algorithms***, review *Studii si Cercetari de Calcul Economic si Cibernetica Economica*, nr. 3/2008, Academy of Economic Studies, Bucharest, 2006

[10] **Ruxanda Gh.** (2009), ***Supervised Pattern Recognition with Potential Functions Methods***, review *Economic Computation and Economic Cybernetics Studies and Research*, no. 2/2009, Academy of Economic Studies, 2009

[12] **Vapnik V. N.** (1999), ***Statistical learning theory***, J. Wiley Interscience, New York, 1999

Annexes

A1. **Neural_net** main program

```
scalar T=@obsrange
scalar n=1
scalar H=3
scalar K=1
scalar c_instr=0.15
scalar c_instr_min=0.01*c_instr
scalar c_instr_max=12*c_instr
scalar c_mom=0.90
scalar n_fct_h=2
scalar n_fct_o=2
matrix (T,n+1) xm
matrix (T,K) y_d
matrix (n+1,H) v
matrix (n+1,H) v_ant
matrix (n+1,H) delt_v_old=0
vector (H) r
vector (H+1) z
matrix (H+1,K) w
matrix (H+1,K) w_ant
matrix (H+1,K) delt_w_old=0
vector (K) s
vector (K) y
vector (n+1) x_f
vector (K) y_d_f
vector (n+1) x_med
vector (K) y_d_med
for !t=1 to T
        xm(!t,1)=1.0
        for !i=1 to n
              xm(!t,!i+1)=x{!i}(!t)
        next
        for !k=1 to K
              !idx=n+!k
              y_d(!t,!k)=x{!idx}(!t)
         next
next
scalar max_epoc=1000
scalar err_ep
scalar min_err_ep=0.000001
vector (max_epoc) err_epoca
vector (max_epoc) c_instr_epoc
scalar ep=1
scalar nr_gen_al_max=10000
call gen_pond_syn(nr_gen_al_max,xm,y_d,v,r,z,w,s,y,err_ep)
v_ant=v
w_ant=w
err_epoca(ep)=err_ep
c_instr_epoc(ep)=c_instr
while (err_ep >= min_err_ep) and (ep <= max_epoc)
        call learn_net(c_instr,c_mom,xm,y_d,v,r,z,w,s,y,err_ep)
        if  err_ep > err_epoca(ep)   then
            while (err_ep >= err_epoca(ep))  and  (c_instr >= c_instr_min)
                  c_instr=0.85*c_instr
                  v=v_ant
                  w=w_ant
                  call learn_net(c_instr,c_mom,xm,y_d,v,r,z,w,s,y,err_ep)
            wend
            c_instr=@mean(c_instr_epoc)*max_epoc/ep
        else
            c_instr=1.15*c_instr
            if  c_instr >= c_instr_max  then
                  c_instr=@mean(c_instr_epoc)*max_epoc/ep
            endif
        endif
        ep=ep+1
        err_epoca(ep)=err_ep
        c_instr_epoc(ep)=c_instr
        v_ant=v
        w_ant=w
wend
matrix (T,4*K) y_pred
scalar err_final=0
call valid_net(T,xm,y_d,v,w,y_pred,err_final)
err_final=@sqrt(err_final/T)
```

A2. ***Learn_net*** subprogram

```
subroutine learn_net(scalar c_instr,scalar c_mom,matrix xm,matrix
y_d,matrix v,vector r,vector z,matrix w,vector s,vector y,scalar err_ep)
scalar g_der
scalar f_der
scalar sum_grad
vector (K) grad_out
vector (H) grad_asc
matrix (n+1,H) delt_v
matrix (H+1,K) delt_w
vector (n+1) x_f
vector (K) y_d_f
scalar err_f
scalar sum_c_instr=0
scalar err_f_old=1.0e+10
for !t=1 to T
    for !i=1 to n+1
         x_f(!i)=xm(!t,!i)
    next
    for !k=1 to K
         y_d_f(!k)=y_d(!t,!k)
    next
    call propag(x_f,y_d_f,v,r,z,w,s,y,err_f)
    for !k=1 to K
        call der_fct_act(n_fct_o,s(!k),g_der)
        grad_out(!k)=(y(!k)-y_d_f(!k))*g_der
        for !h=1 to H+1
          delt_w(!h,!k)=-c_instr*grad_out(!k)*z(!h)+
                        c_mom*delt_w_old(!h,!k)
        next
    next
    for !h=1 to H
        sum_grad=0.0
        for !k=1 to K
            sum_grad=sum_grad+grad_out(!k)*w(!h,!k)
        next
        call der_fct_act(n_fct_h,r(!h),f_der)
        grad_asc(!h)=f_der*sum_grad
        for !i=1 to n+1
           delt_v(!i,!h)=-c_instr*grad_asc(!h)*x_f(!I)+
                         c_mom*delt_v_old(!i,!h)
        next
    next
    v=v+delt_v
    w=w+delt_w
    delt_v_old=delt_v
    delt_w_old=delt_w
    call propag(x_f,y_d_f,v,r,z,w,s,y,err_f)
    if err_f <= err_f_old  then
        if c_instr < c_instr_max  then
            c_instr=1.20*c_instr
        endif
    else
        if c_instr > c_instr_min then
            c_instr=0.80*c_instr
        endif
    endif
    err_f_old=err_f
    sum_c_instr=sum_c_instr+c_instr
next
c_instr=sum_c_instr/T
err_ep=0
for !t=1 to T
    for !i=1 to n+1
        x_f(!i)=xm(!t,!i)
    next
    for !k=1 to K
        y_d_f(!k)=y_d(!t,!k)
    next
    call propag(x_f,y_d_f,v,r,z,w,s,y,err_f)
    err_ep=err_ep+err_f
next
endsub
```

A3. ***Gen_pond_syn*** subprogram

```
subroutine gen_pond_syn(scalar nr_gen_al_max,matrix xm,matrix y_d,matrix
v,vector r,vector z,matrix w,vector s,vector y,scalar err_ep)
scalar nr_gen_al=0
scalar err_min=1.0e+100
scalar err_f
vector (n+1) x_f
vector (K) y_d_f
matrix (n+1,H) v_m
matrix (H+1,K) w_m
scalar nr_subst=0
vector (nr_gen_al_max) err_gen_al
scalar a=-15.0
scalar b= 15.0
while nr_gen_al < nr_gen_al_max
    nr_gen_al=nr_gen_al+1
    for !h=1 to H
        for !i=1 to n+1
            v_m(!i,!h)=@runif(a,b)
        next
    next
    err_ep=0
    for !t=1 to T
        for !i=1 to n+1
            x_f(!i)=xm(!t,!i)
        next
        for !k=1 to K
            !idx=n+!k
            y_d_f(!k)=y_d(!t,!k)
        next
        call propag(x_f,y_d_f,v_m,r,z,w_m,s,y,err_f)
        err_ep=err_ep+err_f
    next
    if err_ep < err_min then
        err_min=err_ep
        v=v_m
        w=w_m
        nr_subst=nr_subst+1
        err_gen_al(nr_subst)=err_min
    endif
wend
```

A4. ***Propag*** and ***Valid*** subprograms

```
subroutine propag(vector x_f,vector y_d_f,matrix v,vector r,vector
z,matrix w,vector s,vector y,scalar err_f)
scalar val_fct
err_f=0
for !h=1 to H
    r(!h)=0.0
    for !i=1 to n+1
        r(!h)=r(!h)+v(!i,!h)*x_f(!i)
    next
    call fct_act(n_fct_h,r(!h),val_fct)
    z(!h+1)=val_fct
next
z(1)=1
for !k=1 to K
    s(!k)=0.0
    for !h=1 to H+1
        s(!k)=s(!k)+w(!h,!k)*z(!h)
    next
    call fct_act(n_fct_o,s(!k),val_fct)
    y(!k)=val_fct
    err_f=err_f+0.5*(y(!k)-y_d_f(!k))^2
next
endsub
```

```
subroutine valid_net(scalar T,matrix xm_pred,matrix y_d_pred,
matrix v,matrix w,matrix y_pred,scalar err_final)
scalar err_f
vector (n+1) x_f
vector (K) y_d_f
for !t=1 to T
    for !i=1 to n+1
        x_f(!i)=xm(!t,!i)
    next
    for !k=1 to K
        y_d_f(!k)=y_d(!t,!k)
    next
    call propag(x_f,y_d_f,v,r,z,w,s,y,err_f)
    err_final=err_final+err_f
    for !k=1 to K
        y_pred(!t,!k)=y(!k)
        y_pred(!t,K+!k)=y_d(!t,!k)
        y_pred(!t,2*K+!k)=y_pred(!t,!k)-y_d(!t,!k)
        if y_d(!t,!k) <> 0.0 then
            y_pred(!t,3*K+!k)=y_pred(!t,2*K+!k)/y_d(!t,!k)*100.0
        endif
    next
next
```

A5. ***fct_act*** and ***der_fct_act*** subprograms

```
subroutine fct_act(scalar n_fct,scalar arg,scalar val)
if n_fct = 1 then
    val=1/(1.0+@exp(-arg))
else
  if n_fct = 2 then
    val=(1.0-@exp(-arg))/(1.0+@exp(-arg))
  else
    if n_fct = 3 then
      if arg >= 0 then
        val=1.0
      else
        val=0.0
      endif
    else
      if n_fct = 4 then
        val=arg
      else
        val=0.0
      endif
    endif
  endif
endif
endsub
```

```
subroutine der_fct_act(scalar n_fct,scalar arg,scalar val)
if n_fct = 1 then
    val=1*@exp(-arg)/(1.0+@exp(-arg))^2
else
  if n_fct = 2 then
    val=2*@exp(arg)/(1.0+@exp(arg))^2
  else
    if n_fct = 3 then
      if arg >= 0 then
        val=1.0
      else
        val=0.0
      endif
    else
      if n_fct = 4 then
        val=1
      else
        val=0.0
      endif
    endif
  endif
endif
endsub
```