Economic classification and regression problems and neural networks

Ekonomické klasifikační a regresní úlohy a neuronové sítě

Arnošt VESELÝ

Department of Information Engineering, Faculty of Economics and Management, Czech University of Life Sciences, Prague, Czech Republic

Abstract: Artificial neural networks provide powerful models for solving many economic classifications, as well as regression problems. For example, they were successfully used for the discrimination between healthy economic agents and those prone to bankruptcy, for the inflation-deflation forecasting, for the currency exchange rates prediction, or for the prediction of share prices. At present, the neural models are part of the majority of standard statistical software packages. This paper discusses the basic principles, which the neural network models are based on, and sum up the important principles that must be respected in order that their utilization in practice is efficient.

Key words: multilayer neural networks, classification, bankruptcy prediction, time series prediction, neural network training, error functions

Abstrakt: Umělé neuronové sítě poskytují efektivní metody pro řešení mnoha ekonomických klasifikačních a regresních problémů. S úspěchem byly například použity pro klasifikaci ekonomických objektů z hlediska jejich náchylnosti k bankrotu nebo pro předpovědi vývoje inflačně-deflačních křivek, vývoje devizových kursů nebo vývoje ceny akcií. Neuronové modely jsou nyní součástí většiny standardních statistických softwarových produktů. Tento článek osvětluje základní principy, na kterých jsou neuronové modely založeny, a shrnuje důležitá upozornění, která je třeba respektovat, aby jejich použití v praxi bylo efektivní.

Klíčová slova: vícevrstvé neuronové sítě, klasifikace, predikce bankrotu, predikce časových řad, učení neuronových sítí, chybové funkce

Neural networks provide powerful models for the statistical data analysis. Their most prominent feature is their ability to learn the dependencies based on a finite number of observations. After learning, the knowledge acquired from the learning samples can be generalized to the yet unseen observations. In the last two decades, the neural networks have been successfully used in many applications in various fields of science and technology. Problems solved have been mostly the classification or regression problems. Many problems in the field of economics can also be regarded in the terms of classification or regression and neural networks can provide effective means for their solution (Herbrich 1999; McNelis 2005).

The majority of papers that use neural networks for the classification tasks in economic applications can be found in the area of the bankruptcy prediction of economic agents. The approach is to use the agents' financial and asset characteristics as an input into the neural network in order to obtain the estimate of the probability of bankruptcy on the output. Neural network based methods often outperform the traditional classification methods (Atia 2001; Fernandez and Olmeda 2006).

Probably the largest amount of economic applications of the neural networks can be found in the field of the prediction of time series. The usual linear models used in statistics perform poorly in the nonlinear cases. However, neural networks are able to approximate any continuous function and therefore they can be expected to provide effective nonlinear models for different time series and thus allow for better predictions. Many authors tried with success to predict the currency exchange rates, as Verkooijen (1996), who linked them to the fundamental variables like GDP and the trade balance, or Emam and Min (2009) and

Supported by the Ministry of Education, Youth and Sports of the Czech Republic (Project No. MSMT 6046070904).

many others. Neural networks were also used for the inflation forecasting. For example Nakamura (2005) evaluated the usefulness of neural networks in an inflation forecasting experiment using the recent U.S data. In his experiment, the neural networks outperformed the autoregressive models. McNelis (2005) studied the forecasting of inflation and deflation on the data from Hong Kong and Japan.

Neural network models are part of the statistical packages, which are on the market today, such as the Statistica, the SPSS or the SAS. All these packages have a user-friendly interface and can be easily used in a similar way as the standard statistical or data mining methods are. The input data can be in a standard table processor format, for example in an Excel table, and can be easily imported into the used software. Users may utilize their own data or may find data on the Internet. Thus they can easily make use of these sophisticated methods to improve their manager decisions.

However, using these methods effectively and interpreting properly the results means to know the underlying fundamental principles, which they are based on. Of course, one may choose a certain method in an haphazard manner and accept those default values of its parameters, which the software supplies. Acting this way means to get worse results or sometimes even misleading results leading to the totally wrong conclusions or predictions. The objective of this paper is to present the basic principles the neural networks methods are based on paying special attention to the features that one must know to choose an appropriate neural model architecture and proper values of its parameters.

METHODOLOGY AND METHODS

Artificial neuron

For building up neural networks, a simple model of physiological neurons called simply neuron is used. The neuron is a simple information-processing unit. It takes a vector of the real-valued inputs, calculates a linear combination of these inputs, adds a constant value called bias or threshold and then applies on the resulting sum an output (activation) function. More precisely, given the inputs x_1, ..., x_n, the output y is computed according to the formula

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{1}$$

where each w_i is a real-valued constant called weight that determines the contribution of the input x_i to the neuron output y, b is the bias (threshold) and $\varphi(\cdot)$ is the output function. The computation of the neuron is illustrated in Figure 1.

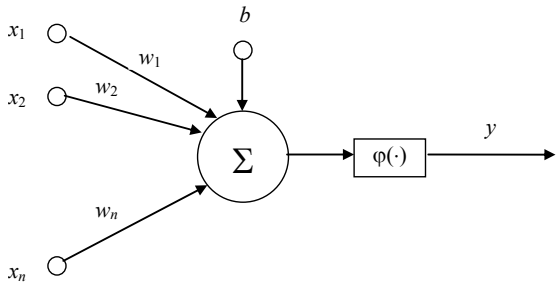

Figure 1. Formal model of a physiological neuron

To simplify the notation, we imagine an additional constant input x_0=1 with weight w_0=b and we write

$$\sum_{i=1}^{n} w_i x_i + b = \sum_{i=0}^{n} w_i x_i \quad \text{and} \quad y = \varphi\left(\sum_{i=0}^{n} w_i x_i\right) \tag{2}$$

The mostly used output functions are the threshold function $\sigma(x)$

$$\sigma(x) = \begin{cases} 1 & \text{if } y > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

the sigmoid function $f(x)$

$$f(x) = \frac{1}{1 + e^{-kx}}, \quad k > 0 \tag{4}$$

the logistic function, which is the sigmoid function (4) with k = 1 or the identity function $\varphi(x) = x$. Neuron with the identity output function is usually called the linear neuron.

Multilayer feedforward networks

Multilayer feedforward networks form an important class of neural networks (see for example Haykin 1999). The network consists of a set of sensory units (receptors) that constitute the input layer, one or more hidden layers of the computation nodes and an output layer that consists also of computation nodes. Computing units in the hidden and output layers are the logistic or sigmoid neurons. In the output layer, also linear neurons might be used. Only neurons of the neighbor layers are connected and the input signal propagates through the network only in the forward direction. The strength of the connection going from i-th neuron of the certain layer to the j-th neuron of the next layer is denoted w_{ji}. When we use the term L-layer network, we refer to a network with L layers of computing units. Thus we shall call a network with one hidden layer a two-layer network, a network with

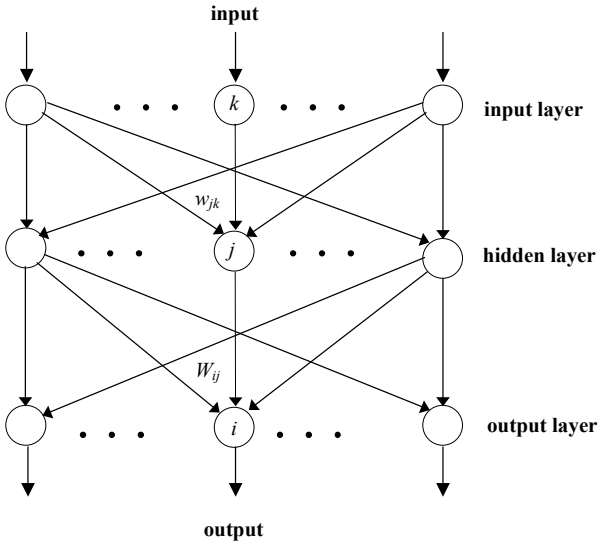

Figure 2. Two-layer feedforward neural network

two hidden layers a three-layer network and so on. A layer network with n inputs and m outputs represents m functions with n arguments. These functions can be easily expressed explicitly. For example, a two-layer network (see Figure 2) with n receptors, one output and m hidden units represents the function

$$y = \varphi\left(\sum_{j=1}^{m} W_j \psi\left(\sum_{i=0}^{n} w_{ji} x_i\right) + W_0\right) \qquad (5)$$

Here W_j is the weight of the connection between j-th neuron at the hidden layer and the output neuron and w_{ji} is the weight of the connection between the i-th input neuron and the j-th neuron at the hidden layer. All neurons in the hidden layer have the same output function $\psi(\cdot)$ and the output neuron has the output function $\varphi(\cdot)$.

Neural network learning

Multilayer feedforward networks can be used for solving different classification or regression problems. The behavior of a neural network is determined by the set of its weights. Therefore, the crucial problem is how to determine the weights in the given problem to get the neural network with the desired behavior.

The neural network approach involves the parameter learning from examples. Neural network learning or training means to have an adaptive procedure in which the weights of the network are incrementally modified. The learning process consists in updating the weights, so that an error function dependent on the network output and the known target value is reduced. Algorithms used for the learning purposes are based on the following methods of adaptation: *gradient learning*, *simulated annealing* and *genetic evolution*.

Gradient learning

Gradient learning is based on a traditional mathematical method of searching the local minimum of a function. Assume that all weights of the neural network are ordered and constitute a vector **w**. The gradient algorithm can be formulated as follows:

Algorithm of the gradient descent search:

1. At the beginning, the weights are set equal to small random values.
2. At the n-th step, the value of gradient grad$E(\mathbf{w}^{n-1})$ is estimated. For the estimation, one or more elements of the learning set are used. Then the weight vector $\mathbf{w}^{n-1}$ is modified according to

$$\mathbf{w}^{n} = \mathbf{w}^{n-1} - \varepsilon \,\mathrm{grad}E(\mathbf{w}^{n-1}) + \mu\,\Delta\mathbf{w}^{n-1} - \gamma\,\mathbf{w}^{n-1} \qquad (6)$$

where $\varepsilon > 0$ determines the rate of the weight vector change, $\Delta\mathbf{w}^{n-1} = \mathbf{w}^{n-1} - \mathbf{w}^{n-2}$ is the weight vector change in the previous $n-1$-th step of the algorithm, $0 \le \mu \le 1$ is a constant called the momentum and $\gamma > 0$ is a regularization (weight decay) constant.

If search algorithm starts with small weights, the output function of the network represents in the weight space roughly the linear surface. The reason is that the neuron output functions are either linear or sigmoid functions, and that the sigmoid functions are near zero approximately linear. Subtraction of $\gamma\mathbf{w}^{n-1}$ helps to keep the weights small which means to have a smoother error surface and this helps the algorithm to escape the local minima.

The use of momentum in the algorithm represents a minor modification of the classical gradient descent algorithm, yet it may improve the learning behavior of the algorithm. It has the effect of gradually increasing the step size of the search in regions where the gradient is unchanging and thus to accelerate the learning process. It also helps to prevent the learning process from terminating in some shallow local minimum of the error surface.

The gradient grad$E(\mathbf{w})$ is usually calculated on the learning set by the means of the algorithm named back propagation of error. Sometimes also the matrix **H,** known as the Hesian matrix, is on the learning set evaluated. Knowing **H** makes it possible to estimate the optimal size of the weights change and thus to accelerate the process of learning. Therefore, the algorithms known as the Quasi-Newton algorithm, the conjugate gradients algorithm and the Levenberg-Marquart algorithm, which all make the use of the

matrix **H** estimate, are quicker than the standard back propagation algorithm.

Simulating annealing

The method is based on the analogy between annealing of solids and solving optimalization. It is a modified Monte Carlo method based on the Metropolis algorithm. In the n-th step of the algorithm, the weight vector **w** of the neural network is modified by a small randomly generated step. If the new error value is smaller than the old one, the step is accepted. Otherwise, the new value of weight vector is accepted only with the probability

$$P = \exp^{-\frac{\Delta E}{T}}$$

Here ΔE is the difference between the new and old value of the error function and T is parameter called temperature. Temperature controls the probability of accepting a step in the wrong direction and during learning process, it is continuously decreasing.

Genetic evolution (Genetic algorithm)

This method starts with a population of neural networks represented by its weight vectors. All weight vectors are unequivocally encoded into a string of real numbers called genome or chromosome. The ability of a neural network, represented by its genome, to solve the underlying regression or classification problem is measured with the fitness function, which might be, for example, a negatively taken error function. From the current population, there are selected couples for "breeding". The greater value of the fitness function means the greater probability of being chosen into the parents' couple. Each parent couple breeds two children that come into being by the means of the procedure called crossover. During the crossover, parts of parents' genomes are shuffled. The incurred children are then put into the new generation. After the new generation is created, its genomes undergo a small probability process called mutation. In the mutation process, one randomly chosen gene of the genome is taken and its value randomly changed. As new generations come into being, their ability to solve the underlying problem ameliorate.

Error functions

The neural network performance criterion is based on the error function $E(\mathbf{w})$ evaluated on the training set. The mostly used error functions are the **sum-of-square error function** for regression tasks and the **cross-entropy error function** for classification tasks.

In a regression problem, the neural network should learn s n-argument functions

$$f_\alpha(x_1, ..., x_n) = f_\alpha(\mathbf{x}), \quad \alpha = 1, ..., s \tag{7}$$

and the training set T consists of N samples

$$T = \{(\mathbf{x}^i, \mathbf{t}^i), i = 1, ..., N\}, \ \mathbf{x}^i = (x_1^i, ..., x_n^i), \ \mathbf{t}^i = (t_1^i, ..., t_s^i) \tag{8}$$

The commonly used error function is the sum-of-square error function

$$E_S(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{\alpha=1}^{s}(y_\alpha^i(\mathbf{x}^i, \mathbf{w}) - t_\alpha^i)^2 \tag{9}$$

where **w** is the vector of the neural network weights and y_α^i is the output of the α-th output neuron if the sample vector $\mathbf{x}^i$ is on its input.

In a classification problem, the input vectors are classified into s classes C_α. The classes are coded using the so-called 1-of-s coding scheme. According to this scheme, the membership of input **x** is coded by the binary vector **t** of length s. Each element t_α of **t** takes the value 1 if the input vector belongs to the class $C_{\alpha,}$ and it takes value 0 otherwise. The obvious advantage of this coding scheme is that the conditional expectation values of variables t_α equal to the conditional probabilities of classes. For two mutually exclusive classes C^+ and C^-, a more simple way of coding can be adopted. Instead of the vector **t,** a single scalar target variable t is used, the value of which is 1 if the vector **x** belongs to the class C^+ and 0 if it belongs to the class C^-.

In the role of the error function, we can use the above described sum-of-square error function or the so- called cross-entropy error function

$$E_{CE}(\mathbf{w}) = \sum_{i=1}^{N}\sum_{\alpha=1}^{s}(t_\alpha^i \log y_\alpha^i(\mathbf{x}^i, \mathbf{w})) \tag{10}$$

From the detailed statistical analysis by the maximum likelihood method, it follows that using the cross-entropy function is more appropriate in classification problems (Bishop 1996).

Preparing data

At the beginning, the data must be attentively scrutinized and the input vectors with missing component values must be either discarded or the missing values

must be substituted with their probable values. For this purpose, some standard statistical method can be used. For the non-categorical data, the simplest substitution utilizes the average value of the missing component, computed over the whole data set. For categorical data, we may estimate the probabilities of the particular values and substitute the missing values according to this probability distribution estimate. A more elaborate approach is to express the variable with missing values in terms of other variables and then to fill the missing values according to the regression function.

After getting rid of the missing values, the data should be normalized. If we suspect the data to be very redundant, we ought to consider their transformation into a feature space with a reduced dimension.

The data should be randomly separated into training, validation and test sets. During the training period, the training and validation sets are used. Vectors from the training set are used for computing the weight vector change, and the vectors from the validation set are used for checking the network performance on new, yet unseen data. Usually if a network is trained, its error on the training set is decreasing. However, from certain moment, the network starts to behave badly, if the previously unseen vectors are put on its input. This phenomenon is called overfitting. To prevent overfitting, the training process is stopped as soon as the phenomenon of overfitting is observed on the validation set. Eventually, the final error of the trained network is estimated on the test set. Of course, a greater training set leads to better trained networks. On the other hand, smaller validation and testing sets mean that the gained results will be less reliable. Therefore, certain compromise must be adopted. If we have too little data at our disposal, we should think of the use of the cross-validation technique known from the mathematical statistics.

RESULTS AND DISCUSSION

Classification

The majority of applications that use the neural network for the classification tasks in economics can be found in the area of the bankruptcy prediction of economics agents. Bankruptcy prediction has long been an important and widely studied topic. Banks need to predict the probability of bankruptcy of a potential client, whether it is a company or a consumer, since the accuracy of this prediction has significant impact on their lending decisions and consequently on their profitability.

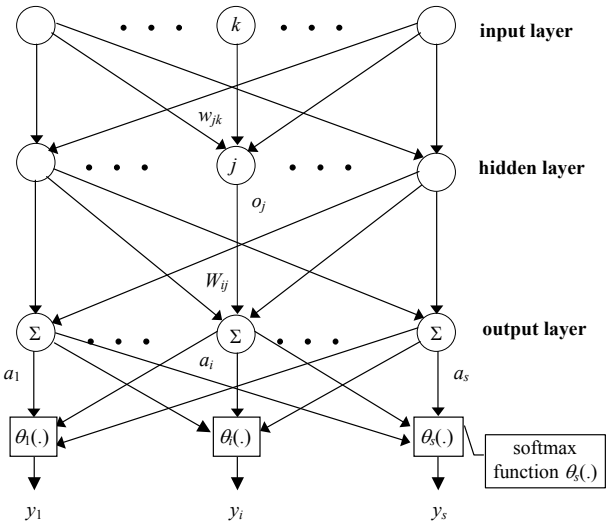

Figure 3. Neural network with softmax output neurons

The trickiest problem in the classification is to choose the proper neural network architecture. As mentioned above, the theoretical analysis recommends using 1-of-*s* coding scheme for the class membership coding and using the cross-entropy function as the error function. Then the minimization of the cross-entropy function provides a network that approximates the conditional probabilities of classes $p(C_\alpha|\mathbf{x})$. As the network outputs are interpreted as probabilities, the logistic neurons, which have their output values from interval <0, 1>, are commonly used in the output layer. If $s > 2$ and the classes are mutually exclusive, then all network outputs should sum to 1. To fulfill this requirement, the so-called softmax neurons are usually used in the network output layer (see Figure 3). Softmax neurons have output functions

$$y_i = \theta_i(a_1,...,a_s) = \frac{\exp(a_i)}{\sum_{j=1}^{s}\exp(a_j)} \tag{11}$$

which obviously sum to 1. If $s = 2$ and input vectors are classified into two mutually excluded classes, the class membership may be coded with the scalar variable and the neural network may have only one output logistic neuron.

The numbers of neurons in the input and output layer are given by the nature of the solved classification problem. However, it is difficult to estimate the size of the hidden layer. Thus the number of neurons in the hidden layer results from experiments. Neurons in hidden layer cannot be linear, because in this case the behavior of the network would degenerate to that of one layer network. Therefore, logistic neurons are usually chosen for the hidden layer.

Training

During training, the input vectors are taken from the training set and the neural network weight vector is adopted in a stepwise manner according to the before chosen learning algorithm. Till the network architecture is definitively settled, some of the fast learning algorithms that are based on the Hessian matrix are used (e.g. conjugate gradients algorithm). Eventually, the classical backpropagation algorithm with the momentum and weight decay parameters is used. We can start learning experiments with the default values of the momentum and the weight decay and then we may try to improve the acquired results by the gradual little change of their value. The overfitting is usually avoided automatically, due to the used software package that is continuously testing the neural network generalization efficiency on the validation set.

Classification error

Very often the input vectors are classified into two disjunctive categories C^+ and C^-. Then besides the *overall error e,* the quality of classification also depends on two parameters named *sensitivity* and *false alarm*. The network classifies the input vectors with sensitivity s if $s \times 100\%$ of the input vectors from category C^+ are correctly determined as being from the category C^+. The network classifies the input vectors with the false alarm f, if $f \times 100\%$ of inputs from category C^- are incorrectly determined as being from the category C^+. The functional dependence of sensitivity on the false alarm is the so-called *ROC curve* (see Figure 4). The overall error e depends on the sensitivity s and the false alarm f

$$e = (1-s)\pi^+ + f\pi^- \tag{12}$$

where π^+ and π^- are incidences of the categories C^+ and C^-. Clearly, in applications as the bankruptcy prediction having a network with big sensitivity s is important.

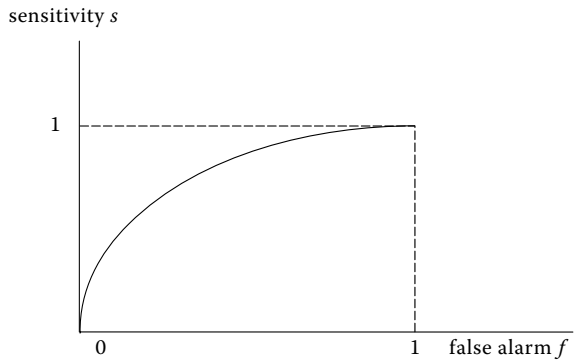

Figure 4. ROC curve

However, according to the ROC curve, the increase of the sensitivity s is followed by the increase of the false alarm f and usually also with the increase of the overall error e. So some compromise must be adopted.

The values of e, s, and f are eventually estimated on the testing set. How good these estimates are can be answered using the standard statistical technique of confidence intervals (see for example Michell 1997). Assume N be the size of the testing set. Then the confidence interval of e is

$$e \pm z_c \sqrt{\frac{e(1-e)}{N}} \tag{13}$$

where z_c is the constant that defines the width of the smallest interval about the mean that includes $100c\%$ of the total probability mass under the bell-shaped Normal distribution (e.g. for $c = 0.95$ the value is $z_c = 1.96$). The confidence intervals for s and f can be calculated in the same way.

Time series prediction

Probably the largest amount of economic applications of neural networks can be found in the field of prediction of time series. Neural network models were successfully used for the inflation-deflation forecasting, for the currency exchange rates prediction or for the prediction of share prices. They often outperformed the standard statistical methods commonly used for time series prediction. However, the time series prediction using neural network is far from straightforward.

Assume that the value $y(t)$ at time t be predicted using the knowledge of its m previous values $y(t-1)$, ..., $y(t-m)$ and the knowledge of $m+1$ values of some parameter $x(t)$, $x(t-1)$, ..., $x(t-m)$. The method the neural network uses for predicting the values of the variable y in time instants t, $t+1$, $t+2$, ... can be described as follows.

1. Values $x(t)$, $x(t-1)$, ..., $x(t-m)$ of x are put into some shift register X and values $y(t-1)$, ..., $y(t-m)$ are put into another shift register Y. Registers X and Y are connected to the receptors of the neural network (see Figure 5). The neural network computes $z(t)$, which is considered to be a prediction of $y(t)$.
2. In the next instant, the values in the registers X and Y are shifted one place to the right. Into the first position of register X, the new value of x is put. Into the first position of the register Y, the value of y, which was in the preceding time instant predicted and which meanwhile materialized, is put. The output of the neural network determines the new value of y predicted for the current instant.

3. To obtain predictions of y for the further time instants, the step 2 is repeated.

However, before the network is used for prediction, it must be trained on the training set in order that its predictions could be reasonable. Assume the training set

$$y(N), \ldots, y(1),\ x(N), \ldots, x(1)\ ,\ N >> m$$

Then the training procedure can be described as follows.

1. The first $m + 1$ values $x(m + 1), x(m), \ldots, x(1)$ of variable x are put into the shift register X and the first m values $y(m), \ldots, y(1)$ of variable y are put into the shift register Y.

2. The output of the network is considered to be a prediction of y. The predicted value of y is compared with the true value of y in the training set and the error of prediction is evaluated. Then according to the before chosen learning algorithm, the weight vector of the network is modified.

3. The values in the registers X and Y are shifted one position to the right and into the first positions of the registers X and Y, the next values of x and y from training set are inserted. Then the step 2 is carried out.

4. The step 3 is repeated until the end of the training set is reached. The last computation is made for the values $y(N - 1), \ldots, y(N - m)$ and $x(N), \ldots, x(N - m)$ and the current learning epoch here ends.

5. If the ending condition of the learning algorithm is not fulfilled, the training continues with step 1 and a further learning epoch is carried out.

Until now, we supposed that only one parameter $x(t)$ supports prediction. The generalization to more such parameters is straightforward. The described model can be also easily modified if the prediction has to be made not one time step ahead, but several time steps ahead. Clearly, if the task is to predict $y(t)$ for n consecutive time steps $t, t + 1, \ldots, t + n - 1$, then the network must have n output neurons. As the time series prediction falls within regression problems, the output neurons are chosen to be linear and as the error function, the sum-of-square error function is chosen. Usually only one hidden layer with logistic neurons is used. The way the network is trained on the training set is similar to that used for classification.

CONCLUSIONS

Neural networks provide powerful means for solving the classification and regression problems. As many economic applications can be formulated in the terms of classification or regression, neural networks might play an important role in the effort to build up more exact and accurate quantitative models in economics and management. Often the neural network models were found to provide generally better results than the classical statistical models, although the computational effort is usually several orders of magnitude higher.

Decades ago, neural networks applications required a big effort as no commercial software was at hand. At present, neural networks are part of the majority of statistical packages with user-friendly graphical interface and easy data input compatible with the standard table processor formats. However, for its successful utilization in practical applications, the knowledge of at least the basic principles of the neural network theory is essential. This paper discusses the

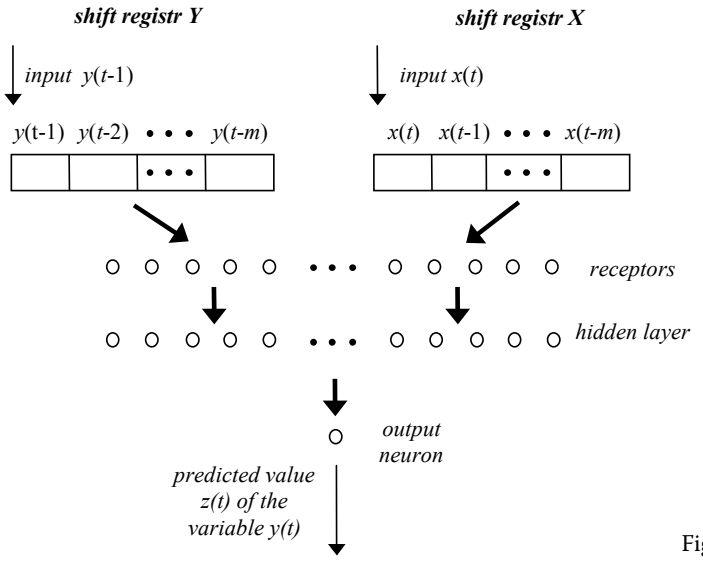

Figure 5. Time series prediction with neural network

basic principles, which the neural network models are based on, and sum up the important principles that must be respected in order that their utilization in practice could be efficient.

REFERENCES

Atia A.F. (2001): Bankruptcy prediction for credit risk using neural networks: A survey and new results. IEEE Transactions on Neural Networks, *12*: 929–935.

Bishop C.M. (1996): Neural Networks for Pattern Recognition. Oxford University Press, Oxford.

Emam A., Min H. (2009): The artificial neural network for forecasting foreign exchange rates. International Journal of Services and Operational Management, *5*: 740–757.

Fernandez E., Olmeda I. (2006): Bankruptcy Prediction with Artificial Neural Networks. In: From Natural to Artificial Neural Computation, Springer, Berlin; ISBN 978-3-540-59497-0.

Haykin S. (1999): Neural Networks. Prentice Hall, London; ISBN 0-13-273350-1.

Herbrich R., Keilbach M., Graepel T., Bollmann-Sdorra P., Obermayer K. (1999): Neural networks in economics: Background, applications, and new developments. Advances in Computational Economics, 11: 169–196.

McNelis P.D. (2005): Neural Networks in Finance. Elsevier Academic Press, Oxford; ISBN 978-0-12-485967-8.

Mitchell T.M. (1997): Machine Learning. McGraw-Hill, Boston; ISBN 0070428077.

Nakamura E. (2005): Inflation forecasting using a neural network. Economics Letters, *86:* 373–378.

Verkooijen W. (1996): A neural network approach to long-run exchange rate prediction. Computational Economics, *9*: 51–65.

Arrived on 11th May 2010

Contact address:

Arnošt Veselý, Czech University of Life Sciences Prague, Kamýcká 129, 165 21 Prague 6, Czech Republic
e-mail: vesely@pef.czu.cz