A Comparison of Feed-forward and Recurrent Neural Networks in Time Series Forecasting

Danko Brezak, Tomislav Bacek, Dubravko Majetic, Josip Kasac and Branko Novakovic, *Member, IEEE*

***Abstract*—Forecasting performances of feed-forward and recurrent neural networks (NN) trained with different learning algorithms are analyzed and compared using the Mackey–Glass nonlinear chaotic time series. This system is a known benchmark test whose elements are hard to predict. Multi–layer Perceptron NN was chosen as a feed-forward neural network because it is still the most commonly used network in financial forecasting models. It is compared with the modified version of the so-called Dynamic Multi–layer Perceptron NN characterized with a dynamic neuron model, i.e., Auto Regressive Moving Average filter built into the hidden layer neurons. Thus, every hidden layer neuron has the ability to process previous values of its own activity together with new input signals. The obtained results indicate satisfactory forecasting characteristics of both networks. However, recurrent NN was more accurate in practically all tests using less number of hidden layer neurons than the feed-forward NN. This study once again confirmed a great effectiveness and potential of dynamic neural networks in modeling and predicting highly nonlinear processes. Their application in the design of financial forecasting models is therefore most recommended.**

I. Introduction

THE aim of time series forecasting is to estimate future data by analyzing past data values. This process is often nonlinear and very demanding for modeling. The significance of prediction models is particularly important in the field of financial engineering where they have been a subject of research for several decades. Many financial models contain some sort of time series forecasting in order to adequately predict future market movements. First models have been based on traditional statistical linear or nonlinear techniques. However, none of these techniques have continuously shown satisfactory prediction success rates owing to the presence of noise and nonlinearity in forecasting financial markets, together with the fact that much is assumed and little is known about the nature of the financial engineering processes [1].

As a nonparametric and nonlinear artificial intelligence method capable of modeling complex functions, neural networks have become widely used in the field of financial time series prediction in the recent years. Because of their abilities to be trained without the restriction of a model to derive parameters and discover relationships, driven and shaped solely by the nature of the data, their importance in this field is significant and growing [2].

Neural networks have been used in experimental and commercial systems for stock market prediction, tracking commodity markets and futures, foreign exchange trading, financial planning, company stability prediction, to scan credit and loan applications, estimate bankruptcy probabilities, etc. [1, 3]. The most popular model is a feed-forward Multi–layer Perceptron neural network (MLP NN) which is predominantly trained using Error Back-propagation algorithm [1, 3–6]. In order to speed up the learning process, avoid local minima and overfitting problem, and to determine optimal network structure, other learning algorithms and neural network configurations, such as conjugate gradient algorithm, genetic algorithms, recurrent, self-organizing and hybrid neural networks, are also occasionally analyzed [3, 7–8]. There is no single network configuration or learning method which can be adequately used for all domains because each has its own advantages and disadvantages. According to [3], the major research thrust in this area should be determining better network architectures, because the commonly used feed–forward back–propagation network offers good performance, but this performance could be improved by using recurrence or reusing past inputs and outputs. The motivation behind using recurrence is that pricing patterns may repeat in time. A network which remembers previous inputs or feedbacks previous outputs may have greater success in determining these time dependent patterns.

In that sense, a comparison of feed-forward and recurrent neural networks has been performed in this study. Both networks were trained with four types of algorithms: Error Back–propagation (EBP), Resilient Back–propagation (RPROP), Conjugate gradient (CG) and Levenberg–Marquardt (LM). Performances of chosen network and learning algorithms were compared using the Mackey–Glass nonlinear chaotic system, which is a good benchmark test because its elements are hard to predict.

Manuscript received November 1, 2011. This work was supported by the Ministry of Science, Education and Sport of the Republic of Croatia through fund for national scientific projects.

D. Brezak is with the Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia (phone: +385 1 6168357, fax: +385 1 6168351; e-mail: danko.brezak@fsb.hr).

T. Bacek is with the Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb (e-mail: tomislav.bacek@fsb.hr).

D. Majetic is with the Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb (e-mail: dubravko.majetic@fsb.hr).

J. Kasac is with the Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb (e-mail: josip.kasac@fsb.hr).

B. Novakovic is with the Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb (e-mail: branko.novakovic@fsb.hr).

II. Neural Network Algorithms

A. Feed-forward Neural Network

Feed–forward neural network analyzed in this paper is the most commonly used MLP NN with three layers. The input of the *j*th hidden layer neuron (except Bias which has no input) for the *n*th learning sample is defined as

$$net_j(n) = \sum_{i=1}^{I} v_{ji} x_i(n), \quad j = 1,..., J-1, \tag{1}$$

where I is the number of input layer neurons, v are their weight factors, and x are network inputs. This sum is the argument of the bipolar sigmoid activation function

$$y_j(n) = \frac{2}{1+e^{-c_j net_j(n)}} - 1, \quad j = 1,..., J-1, \tag{2}$$

where c_j is adaptive gain parameter which defines function slope. Bias output is always one ($y_J = 1$).

By summing the outputs of all hidden layer neurons (including Bias) and belonging weight factors w, the final *m*th network output value is defined as

$$O_m(n) = \sum_{j=1}^{J} y_j(n) w_{mj}, \quad m = 1,...,M. \tag{3}$$

Three types of parameters have to be adapted in the learning process: v, w, and c.

B. Recurrent Neural Network

The Dynamic Multi–layer Perceptron Network (DMLP), proposed in [9], was modified and used as the second, recurrent type of network in this study. Unlike the feed-forward MLP NN, this type of network is characterized by a dynamic neuron model, the so-called Dynamic Elementary Processor (DEP) which is structured as an Auto Regressive Moving Average (ARMA) filter, and is built into the network hidden layer. Thus, every hidden layer neuron has the ability to process previous values of its own activity together with new input signals. Unlike the original DMLP structure, this network uses Gauss activation functions whose parameters are adapted during the learning process. Because Gaussian parameters control shape and position of the activation functions Bias neuron and its weights can be omitted. The structure is additionally simplified by omitting the output layer activation function.

The input value of the *j*th hidden layer neuron for the *n*th learning sample is calculated from the sum of the products of all I network input vector elements x and their weight factors v:

$$net_j(n) = \sum_{i=1}^{I} v_{ji} x_i(n), \quad j = 1,..., J. \tag{4}$$

The obtained sum is then processed in the DEP unit (ARMA filter), which can be written in the form of an impulse transfer function:

$$G_j(z) = \frac{B_j(z)}{A_j(z)} = \frac{\overline{y}_j(z)}{net_j(z)} = \frac{b_{0j} + b_{1j} z^{-1} + b_{2j} z^{-2}}{1 + a_{1j} z^{-1} + a_{2j} z^{-2}}, \tag{5}$$

that is, in the form of a difference equation

$$\begin{aligned} \overline{y}_j(n) = {} & b_{0j} net_j(n) + b_{1j} net_j(n-1) + b_{2j} net_j(n-2) - \\ & - a_{1j} \overline{y}_j(n-1) - a_{2j} \overline{y}_j(n-2), \end{aligned} \tag{6}$$

where a and b are the filter coefficients, and $\overline{y}_j$ is the filter output.

The output of the *j*th hidden layer neuron (DEP unit) is

$$y_j(n) = e^{-\frac{1}{2}\left(\frac{\overline{y}_j(n) - t_j}{\sigma_j}\right)^2}, \tag{7}$$

where t_j is the centre and σ_j is the width of the activation function. Network outputs are computed as in (3). Nine types of network parameters have to be adapted in the learning process: v, w, b_0, b_1, b_2, a_1, a_2, t, and σ.

III. Learning Algorithms

Learning process of both types of neural networks is divided into two phases: feed–forward and feedback. The result of the feed–forward phase is network outputs, which are then compared to desired values. The difference between desired and actual network outputs is a learning error which is used to change network learning parameters in a feedback phase. Parameter change, in its general form, is defined as

$$\vartheta(k+1) = \vartheta(k) + \Delta\vartheta(k), \tag{8}$$

where $\vartheta(k)$ is a learning parameter value in the kth learning step (epoch), $\Delta\vartheta(k)$ denotes parameter change, while $\vartheta(k+1)$ is a new parameter value in the next learning step. Learning algorithms, which are briefly depicted hereafter, differ in the way $\Delta\vartheta(k)$ is changed. In all four utilized methods learning parameters were updated after all learning samples were presented to the network (batch learning procedure). Additionally, in the case of EBP method, both networks were also trained with a pattern or stochastic procedure. It implies adaptation of learning parameters after each sample from the learning data set is presented to the network.

A. Error Back-propagation Algorithm

The EBP is the most widely used learning algorithm for the feed–forward neural networks. It is actually a simple gradient method, which means that parameter change is scaled error gradient

$$\Delta\vartheta(k) = -\eta \nabla E_\vartheta(k). \tag{9}$$

Scale factor η is known as learning coefficient. Parameter change, as given in (9), has several drawbacks related to the problem of choosing adequate learning coefficient in order to speed up the learning and, at the same time, avoid oscillations of the error function. Also, there is no guarantee that the algorithm will find global minimum of the error function, and the influence of the partial derivative on the rate of change of learning parameters is "contra intuitive", as it is emphasized and explained in [10]. In order to prevail above mentioned drawbacks, Zurada [11] and Pearlmutter [12] suggested using the first (α) and the second order momentum (β), which act as a low-pass filter and can significantly decrease oscillations

$$\Delta\vartheta(k) = -\eta \nabla E_\vartheta(k) + \alpha\vartheta(k-1) + \beta\vartheta(k-2). \tag{10}$$

Although proposed modifications can lead to significant improvements in convergence rates, there is still no guarantee for that to happen. Furthermore, there is no such thing as an optimal learning coefficient or momentum, but only suggestions of intervals from which these values should be chosen [10, 13].

Beside above–mentioned improvements of the EBP algorithm, Almeida [14] suggested using adaptive learning coefficient η. In this paper, SuperSAB method [15] is used for learning coefficient adaptation:

$$\eta(k) = \begin{cases} \eta^+ \cdot \eta(k-1), & \nabla E_\vartheta(k-1) \cdot \nabla E_\vartheta(k) > 0 \\ \eta^- \cdot \eta(k-1), & \nabla E_\vartheta(k-1) \cdot \nabla E_\vartheta(k) < 0 \\ \eta(k-1), & \nabla E_\vartheta(k-1) \cdot \nabla E_\vartheta(k) = 0 \end{cases}, \tag{11}$$

where $\nabla E_\vartheta(k)$ and $\nabla E_\vartheta(k-1)$ denote partial derivative of the error function with respect to learning parameter in a current and previous learning step, respectively, whereas $\eta(k–1)$ denotes individual learning coefficient in the previous step. Also, the values of η^+ and η^- should be in the interval $0 < \eta^- < 1 < \eta^+$. Adaptation of η coefficient was used only in the batch learning approach, while in the pattern learning η remained constant ($\eta = 0.01$).

Error function E is for all analyzed methods defined as the sum of squared errors of all network outputs (M)

$$E = \frac{1}{2}\sum_{m=1}^{M}\sum_{n=1}^{N}\left(d_{mn} - O_{mn}\right)^2, \tag{12}$$

where d_{mn} and O_{mn} denote desired and actual network response value of the mth output for the nth learning sample.

B. Resilient Back-propagation Algorithm

The RPROP algorithm is a heuristic method developed in order to overcome the main drawback of the EBP algorithm – dependence on the size of the partial derivative. Instead of dependence on the size of the partial derivative, parameter change in the RPROP algorithm depends only on the sign of derivative in two succeeding learning steps, which leads to the same convergence speed of the algorithm regardless of the error surface. Other advantages of such approach are discussed in [10, 16].

A variant of RPROP method proposed in [16] and named as iRPROP+ has been used in this study. Modification of the network learning parameter ϑ depends on parameter Δ which has to be adapted in every kth learning step according to the following expression:

$$\Delta(k) = \begin{cases} min\left(\eta^+ \cdot \Delta(k-1), \Delta_{max}\right), & \nabla E_\vartheta(k-1) \cdot \nabla E_\vartheta(k) > 0 \\ max\left(\eta^- \cdot \Delta(k-1), \Delta_{min}\right), & \nabla E_\vartheta(k-1) \cdot \nabla E_\vartheta(k) < 0 \\ \Delta(k-1), & \nabla E_\vartheta(k-1) \cdot \nabla E_\vartheta(k) = 0 \end{cases}, \tag{13}$$

where $0 < \eta^- < 1 < \eta^+$. Suggested values of η^+ and η^-, as well as of the minimal and the maximal individual step (Δ_{min}, Δ_{max}) are discussed in [10]. Parameter change rule is defined as

$$\Delta\vartheta(k) = -sign\,\nabla E_\vartheta(k) \cdot \Delta(k). \tag{14}$$

If the sign of the partial derivative in two sequential steps is changed ($\nabla E_\vartheta(k–1)\nabla E_\vartheta(k) < 0$), the gradient of error function with respect to the learning parameter ϑ is set to zero ($\nabla E_\vartheta(k) = 0$), thus reducing the amount of ϑ in the next step. If, at the same time, condition $E_\vartheta(k) > E_\vartheta(k–1)$ is fulfilled, then $\Delta\vartheta(k) = -\Delta\vartheta(k–1)$, and in the case $E_\vartheta(k) \leq E_\vartheta(k–1)$, the amount of learning parameter in the actual step does not change ($\vartheta(k+1) = \vartheta(k)$).

C. Conjugate Gradient Algorithm

The CG algorithm is the only global optimization method used in this study. Global methods beside local information (partial derivative) take into account global information as well (direction of parameter change vector in a previous epoch). Parameter change is therefore defined as

$$\Delta\vartheta(k) = \eta(k)\,d(k), \tag{15}$$

where η denotes learning coefficient and d minimum search direction.

Learning coefficient is usually determined using line search method, but this method can lead to a multiple error calculations in a single epoch, so in this paper modified SuperSAB method is used [17] where

$$\eta(k) = \begin{cases} d^+ \cdot \eta(k-1), & \nabla E_\vartheta^T(k) \cdot \nabla E_\vartheta(k-1) \geq 0 \\ d^- \cdot \eta(k-1), & \nabla E_\vartheta^T(k) \cdot \nabla E_\vartheta(k-1) < 0 \end{cases}, \tag{16}$$

and $0 < d^- < 1 < d^+$. Search direction is always conjugate to all previous directions

$$d(k+1) = -\nabla E_\vartheta(k+1) + \beta(k)\, d(k), \tag{17}$$

where β is a positive number bounded above in order to avoid algorithm instability (usually, β_{max} = 1.2 guarantees stability). This coefficient can be determined in several different ways. In this study, Fletcher-Reeves expression was used [17],

$$\beta(k) = min\left\{ \frac{\nabla E_\vartheta^T(k+1)\nabla E_\vartheta(k+1)}{\nabla E_\vartheta^T(k)\nabla E_\vartheta(k)}, \beta_{max} \right\}. \tag{18}$$

D. Levenberg–Marquardt Algorithm

The LM algorithm is the most widely used optimization tool in the field of nonlinear optimization. It is a pseudo-second order method which means that, beside gradient information, it also uses information of the Hessian approximation. The idea of the LM algorithm is quite simple. When a mapping function is linear, the sum of squared error is quadratic and minimization problem becomes trivial (parabola). On the other hand, when a mapping function is nonlinear, which is a case with neural networks, sum of squared error is of higher order and minimization becomes a problem. Newton–Gauss method, which is a base for the LM algorithm, solves this problem by linearizing mapping function near minimum.

Quadratic error function approximation near its minimum results in the following learning parameter change [18]:

$$\Delta\vartheta(k) = -\left[\nabla^2 E_\vartheta(k)\right]^{-1} \nabla E_\vartheta(k), \tag{19}$$

where $\nabla^2 E_\vartheta(k)$ denotes Hessian and error function gradient is

$$\nabla E_\vartheta(k) = J_\vartheta^T(k)\, e(k), \tag{20}$$

where J denotes Jacobian, and e is the error vector given as

$$e = \sum_{m=1}^{M}\sum_{n=1}^{N}(d_{mn} - O_{mn}). \tag{21}$$

Because Hessian is often hard to find, the following approximation is used:

$$\nabla^2 E_\vartheta(k) = J_\vartheta^T(k)\, J_\vartheta(k). \tag{22}$$

For every network learning parameter different Jacobians need to be calculated.

Parameter change rule of the Levenberg algorithm is a combination of simple gradient method and the Newton–Gauss method,

$$\Delta\vartheta(k) = -\left(\nabla^2 E_\vartheta(k) + \lambda I\right)^{-1} \nabla E_\vartheta(k), \tag{23}$$

where λ denotes gradient method learning coefficient. If error function increases, quadratic approximation is unsatisfactory and λ is increased by the factor β_{inc} – gradient method becomes dominant in the adaptation of network learning parameters. If error decreases, quadratic approximation is good and λ is decreased by factor β_{dec} – adaptation of learning parameters is now mainly under the influence of the Newton–Gauss method. Although the same factors is usually used in both cases, that was not the case in this study (β_{inc}= 100; β_{dec}= 0.01), because separate parameters result in better convergence.

In the case of large λ, information on Hessian is neglected. Since Hessian can provide useful information, Marquardt suggested modification of the rule given in (23) in the form known as Levenberg–Marquardt algorithm:

$$\Delta\vartheta(k) = -\left(\nabla^2 E_\vartheta(k) + \lambda\, diag\left(\nabla^2 E_\vartheta(k)\right)\right)^{-1} \nabla E_\vartheta(k). \tag{24}$$

IV. Experimental Work

A. The Mackey–Glass Chaotic Time Series

Lapedes and Farber [19] suggested the Mackey–Glass chaotic time series as a good benchmark for testing different forecasting algorithms, because it has a simple definition, and yet its elements are hard to predict. Discrete Mackey–Glass dynamic system is defined as

$$x(n-1) = \frac{1}{1+b}\left[x(n-1) + \frac{a\,x(n-\tau)}{1+x^{10}(n-\tau)} \right], \tag{25}$$

where a and b are constants, and τ is time delay. Sampling time is T_0 = 1s. In this study, a = 0.2, b = 0.1 and τ = 30 were chosen.

The aim of the analyzed forecasting algorithm is to predict future value of the time series at instant P in the future using m past values and the actual one. The standard method for this type of prediction is to create a mapping function $f(\bullet)$ as follows:

$$x(t+P) = f\left(x(t),\, x(t-\Delta),\, x(t-2\Delta),\ldots,\, x(t-m\Delta)\right), \tag{26}$$

where Δ is a time delay. We choose $P = \Delta = 6$, and m = 4, for which

$$x(t+6) = f\left(x(t),\, x(t-6),\, x(t-12),\ldots,\, x(t-24)\right). \tag{27}$$

B. Results

Since neural network task in this study was to predict one future value by using values from the four past and a present point, input layer has $i = 5$ neurons (plus bias in the case of MLP NN), while output layer has only one neuron ($m = 1$). Hidden layer can have arbitrary number of neurons (j), so different tests were performed using 2, 5, 7, 10, 12 and 15 neurons. The MLP NN with 12 (+Bias) hidden layer neurons and recurrent NN with 7 and 10 neurons have predominantly shown the best results (Table I and II).

In order to improve learning, a modification of random weight initialization is proposed [20]:

$$W = 0.7H^{\frac{1}{N}}\left(-1+2\cdot rand\right), \tag{28}$$

where H and N denote the number of neurons in layers connected with the weight vector W – H refers to succeeding and N to preceding layer. Weights are initialized in the interval [0.001, 0.1], and the outputs from the Mackey–Glass model are normalized in the interval [0, 1].

Adaptation of the network parameters has been performed using 500 learning samples accompanied by additional 500 samples for the validation. Every learning process was carried out in 30000 steps, during which current network algorithm was validated after every 10 steps in order for the network to achieve good generalization characteristics. Network learning parameters were memorized only in the case when new validation error was lower than the previous one. The results are quantified and compared with the normalized root mean square error (NRMSE):

$$NRMSE = \frac{1}{\sigma_d}\sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(d_n - O_n\right)^2}, \tag{29}$$

where

$$\sigma_d = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(d_n - \overline{d}\right)^2}, \quad \overline{d} = \frac{1}{N}\sum_{n=1}^{N} d_n . \tag{30}$$

Parameters d_n and O_n denote desired and actual network response for the nth sample, and σ_d is a standard deviation. After completing the learning phase all network structures were tested with 3 tests each composed from 500 new samples.

The best results of both networks for every learning method are presented in Table I and II. In the case of MLP NN (Table I), test results have shown good prediction characteristics of the network regardless the utilized learning method. Although test results are very similar, it can be concluded that the best network performance was achieved with LM algorithm in view of all criteria – the fastest error function convergence and the lowest NRMSE values obtained in the learning, validation and test phases. As expected, network trained with both variants of the EBP method has shown the slowest convergence of error function.

TABLE I
PREDICTION OF THE MACKEY–GLASS TIME SERIES USING MLP NN

Learning algorithm	Learning steps	NRMSE				
		Learning	Validation	Test 1	Test 2	Test 3
EBP (patt.)	30000	0.066	0.094	0.075	0.07	0.08
EBP	29000	0.062	0.083	0.082	0.0652	0.071
iRPROP+	18000	0.07	0.088	0.085	0.071	0.077
CG	12000	0.062	0.083	0.082	0.067	0.071
LM	1000	0.036	0.078	0.076	0.063	0.06

TABLE II
PREDICTION OF THE MACKEY–GLASS TIME SERIES USING RECURRENT NN

Learning algorithm	Learning steps	NRMSE				
		Learning	Validation	Test 1	Test 2	Test 3
EBP (patt.)	30000	0.017	0.047	0.063	0.021	0.044
EBP	14770	0.035	0.045	0.087	0.037	0.043
iRPROP+	19450	0.027	0.061	0.074	0.039	0.058
CG	12240	0.043	0.071	0.076	0.05	0.066
LM	8280	0.036	0.062	0.079	0.049	0.058

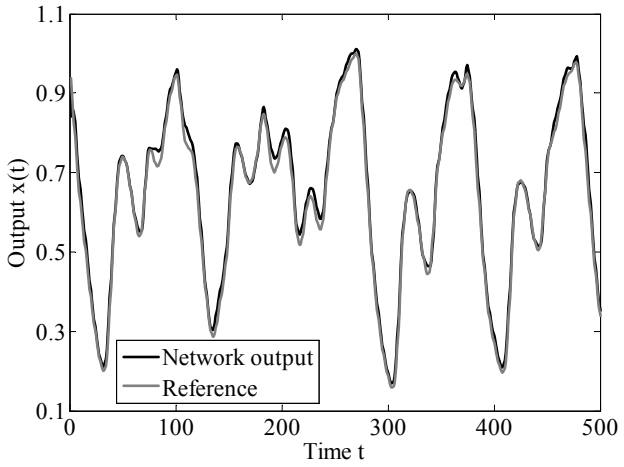

Fig. 1. Prediction of the Mackey–Glass time series using recurrent NN trained with EBP algorithm (pattern learning) – Test 1.

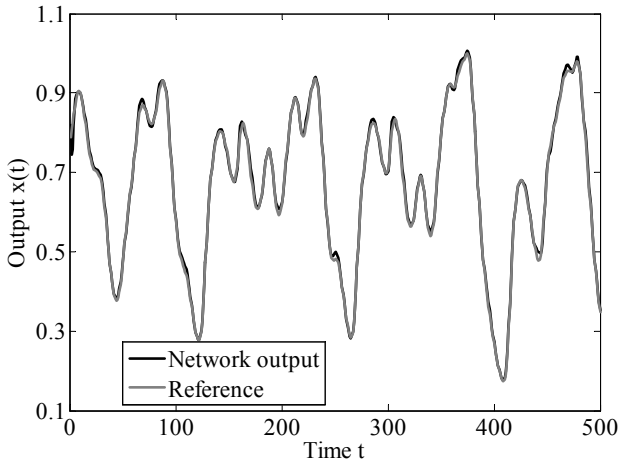

Fig. 2. Prediction of the Mackey–Glass time series using recurrent NN trained with EBP algorithm (pattern learning) – Test 2.

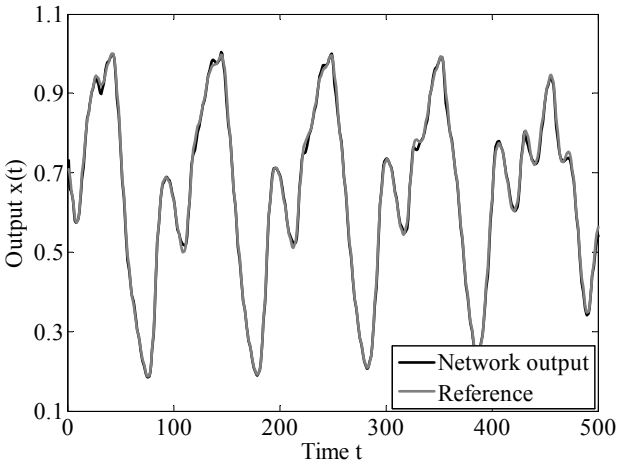

Fig. 3. Prediction of the Mackey–Glass time series using recurrent NN trained with EBP algorithm (pattern learning) – Test 3.

On the other hand, recurrent NN expectedly achieved better results in practically all tests (Table II). However, it is interesting to note that the best overall results were achieved by the EBP method when using pattern learning procedure, except in the case of error convergence rate which was the lowest one (30000 steps). Network outputs of all tests are presented in Fig. 1-3. The other variant of EBP algorithm (batch learning) provided the second best network performance with the error convergence rate (14770 steps) smaller than in the case of the CG algorithm (12240 steps) or LM method (8280 steps), but unexpectedly higher comparing to the iRPROP+ (19450 steps).

REFERENCES

[1] M. Thenmozhi, "Forecasting Stock Index Returns Using Neural Networks," *Delhi Business Review,* vol. 7, pp. 59–69, July – December 2006.

[2] M. P. Wallace, "Neural Networks and Their Application to Finance," *Business Intelligence Journal*, vol. 1, pp. 67–77, July 2008.

[3] R. Lawrence, "Using Neural Networks to Forecast Stock Market Prices," Department of Computer Science, University of Manitoba, Winnipeg, Canada, Tech. Rep., Dec. 1997, Available: https: //people. ok.ubc.ca/rlawrenc/research/Papers/nn.pdf.

[4] M. Adya and F. Collopy, "How Effective are Neural Networks at Forecasting and Prediction? A Review and Evaluation," *Journal of Forecasting*, vol. 17, pp. 481–495, September – November 1998.

[5] D. Zhang, Q. Jiang and X. Li, "Application of Neural Networks in Financial Data Mining," *International Journal of Information and Mathematical Sciences*, vol. 1, pp. 106–109, Spring 2005.

[6] J. Yao and C. L. Tan, "Guidelines for Financial Forecasting with Neural Networks," in *Proc. of International Conference on Neural Information Processing,* Shanghai, China, 2001, pp. 772–777.

[7] A. Skabar and I. Cloete, "Neural Networks, Financial Trading and the Efficient Markets Hypothesis," *Australian Computer Science Communications*, vol. 24, pp. 241–249, January-February, 2002.

[8] M.-C. Chan, C.-C. Wong and C.-C. Lam, "Financial Time Series Forecasting by Neural Network Using Conjugate Gradient Learning Algorithm and Multiple Linear Regression Weight Initialization," in *Proc. of Computing in Economics and Finance*, Barcelona, Spain, 2000.

[9] M. Ayoubi, M. Schefer, and S. Sinsel, "Dynamic neural units for nonlinear dynamic systems identification", Lecture Notes in Computer Science, vol. 930, pp. 1045-1051, 1995.

[10] M. Riedmiller, "Advanced Supervised Learning in Multi-layer Perceptrons – From Backpropagation to Adaptive Learning algorithms", Computer Standards & Interfaces, vol. 16, pp. 265-278, 1994.

[11] J. M. Zurada, *Artificial Neural Systems*, W.P. Company, USA, 1992.

[12] B. Pearlmutter. (1991). Gradient descent: Second order momentum and saturating error. *Advances in Neural Information Processing Systems*. [Online]. Available: http://books.nips.cc/nips04.html

[13] R. M. Hristev. (1998). The ANN book [Online]. Available: ftp://ftp.informatik.uni-freiburg.de/papers/neuro/ANN.ps.gz

[14] L. B. Almeida, *Handbook of Neural Computation*, Bristol, Philadelphia, Oxford University Press, 1997 ch. C

[15] T. Tollenaere, "SuperSAB: Fast adaptive backpropagation with good scaling properties", *Neural Networks*, vol. 3, pp. 561-573, 1990.

[16] C. Igel and M. Husken, 'Improving The RPROP Learning Algorithm', in *Proc. of the Second International Symposium on Neural Computation*, Berlin, Germany, 2000, pp. 115-121.

[17] J. Kasac, J. Deur, B. Novakovic, I.V. Kolmanovsky, "A Conjugate Gradient-based BPTT-like Optimal Control Algorithm", *in Conf. Rec. 2009 IEEE Multiconference on Systems and Control,* pp.861–866.

[18] S. Roweis, Levenberg-Marquardt Optimization [Online]. Available: http://cs. nyu.edu/~roweis/notes/lm.pdf, 2011.

[19] A.S. Lapedes and R. Farber, "Nonlinear Signal Processing Using Neural Networks: Prediction And System Modeling", Los Alamos National Laboratory, Los Alamos, New Mexico, Tech. Rep., 1987.

[20] D. Nguyen and B. Widrow, "Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights", in *Proc. of the International Joint Conference on Neural Networks*, San Diego, CA , USA, vol.3, pp. 21–26, 1990.