MODIFIED DYNAMIC NEURON MODEL

Dubravko Majetić, Danko Brezak, Josip Kasać, Branko Novaković

Prof.dr.sc. D. Majetić, University of Zagreb, FSB, I. Lucica 5, 10000 Zagreb
Mr.sc. D. Brezak, University of Zagreb, FSB, I. Lucica 5, 10000 Zagreb
Dr.sc. J. Kasać, University of Zagreb, FSB, I. Lucica 5, 10000 Zagreb
Prof.dr.sc. B. Novaković, University of Zagreb, FSB, I. Lucica 5, 10000 Zagreb

Keywords: dynamic neural network, adaptive activation function, bias neuron, error back–propagation, RPROP

Abstract

In this paper a modification of the nonlinear dynamic discrete-time neuron model, the so-called Dynamic Elementary Processor (DEP), is proposed. DEP disposes of local memory, in that it has dynamic states. Instead of the most popular unipolar and bipolar Sigmoidal neuron activation functions, the Gauss activation function with adaptive parameters is applied. Based on the DEP neurons in hidden layer a modified dynamic neural network (MDNN) without any Bias neurons is proposed. For such neural network, the Error Back-Propagation and RPROP learning strategies are compared in solving of two benchmarks, the Glass-Mackey time series prediction and XOR classification problem.

1. INTRODUCTION

The first version of DEP neuron was presented in [1]. Dynamic neural network based on DEP neurons in hidden layer was trained according the Error Back–Propagation (EBP) learning procedure. It is well known that such learning algorithm is quite slow. Over the last decade, many improvement strategies have been developed to speed up the EBP algorithm and improve neural network learning and generalization features. All of them can be separated in three basic categories.

The first category deals with the improvement of the EBP learning algorithm [2]. The second category deals with the neurons weights initial values [3, 4] and the third category deals with neural network topology optimization [5]. Therefore in this paper two directions or strategies for MDNN learning speed up are discussed. The first one deals with neural network topology optimization, and the second one deals with different type of the learning algorithm. According to the first strategy, the DEP neuron structure modification and activation function (AF) with adaptive parameters are proposed for modified dynamic neural network (MDNN). With such adaptive Gauss AF we wish to eliminate the Bias neuron from all network layers. That leads to minimization of neural network topology. But before doing that, we must answer to some fundamental questions. The first one is, why is the Bias neuron so important, and the second one is, can we eliminate Bias neuron for hidden layer?

It is well known that any nonlinear, smooth, differentiable, and preferably non-decreasing function can be used as AF in hidden layer. The two most popular activation functions are the unipolar Logistic and the bipolar Sigmoidal functions. For those types of activation functions, Bias neuron is very important, and the error-back propagation neural network without Bias neuron for hidden layer does not learn [6]. Shortly, the Bias weights have control of the shape, orientation and steepness of all types of Sigmoidal functions through data mapping space. However, if one uses Gauss AF with adaptive parameters, than the Bias neuron with related weights can be excluded. Gauss function parameters controls the AF shapes and orientation, and the position of activation functions in data mapping space. According to the second strategy the RPROP algorithm [7] is proposed and compared with EBP algorithm. Although RPROP algorithm was originally designed for feed–forward neural networks, it shows the great learning potentials for recurrent neural networks as well.

2. MODIFIED DYNAMIC NEURAL NETWORK

The basic idea of the dynamic neuron concept is to introduce some dynamics to the neuron activation function (AF), such that the neuron activity depends on the internal neuron states. In this study, an ARMA (Auto Regressive Moving Average) filter is integrated within the well-known static neuron model. This filter allows the neuron to act like an infinite impulse response filter, and the neuron processes past values of its own activity and input signals. Therefore some good network properties in signal filtering can be expected [8]. The structure of a proposed dynamic neuron model is plotted in figure 1. The filter input and output at time instant (n) are given in (1) and (2) and respectively [1]:

$$net(n) = \sum_{j=1}^{J-1} w_j u_j \ . \qquad (1)$$

$$\tilde{y}(n) = b_0 net(n) + b_1 net(n-1) + b_2 net(n-2) - \\ -a_1\tilde{y}(n-1) - a_2\tilde{y}(n-2). \quad (2)$$

The hidden layer neuron output is given with (3),

$$y(n) = \gamma(\tilde{y}(n)), \quad (3)$$

where $\gamma(n)$ represents the adaptive neuron activation function defined in (4).

$$y(n) = \gamma(\tilde{y}(n)) = e^{-\frac{1}{2}\left(\frac{\tilde{y}-c}{\sigma}\right)^2} \quad (4)$$

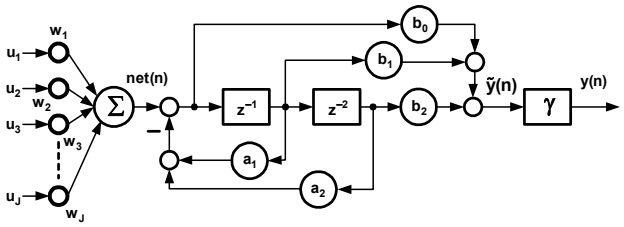

Figure 1. Discrete-time DEP neuron model

Adaptive means that AF parameters σ and C can be adapted by neural network learning algorithm. MDNN shown in figure 2 has three layers. Each i-th neuron in the first, input layer has single input that represents the external input to the neural network.

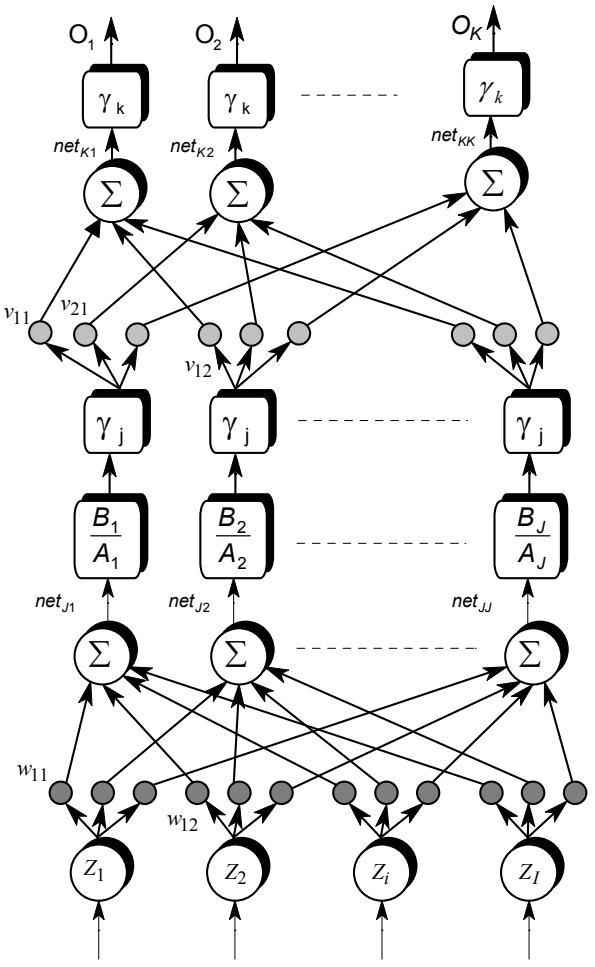

Figure 2. Modified dynamic neural network

The second layer consists of dynamic neurons, which are presented by figure 1. Each j-th dynamic neuron in hidden layer has an input from every neuron in the first layer. And finally, each k-th neuron in the third, output layer has an input from every neuron in the second layer. The adaptive neuron activation function given by (4) is chosen activation function for static neurons in output layer.

$$O_k(n) = \gamma(net_{Kk}(n)) = e^{-\frac{1}{2}\left(\frac{net_{Kk}-c_k}{\sigma_k}\right)^2}, \quad (5)$$

where k=1,2,…,K is the number of neural network outputs.

All error measures are reported using non-dimensional Normalized Root Mean Square error index NRMS [1, 9]. Normalized means that the root mean square is divided by the standard deviation of the target or output desired data.

In all hereafter performed experiments the training started with random weights values between -1 and +1. All σ parameters were set to 0.5 and all C parameters were set to zeros. The filter coefficients $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ were initialized to zeros, while the coefficients $\boldsymbol{b}_0$ were initialized to ones to support a stable learning procedure. For equal networks topologies the same initial learning parameters were used in all learning procedures.

3. EBP LEARNING ALGORITHM

The goal of the supervised learning algorithm is to adjust the neural network parameters (the weights, filter coefficients and activation function parameters) based on a given set of input and desired output pairs. Therefore for the neural network plotted in figure 2, the EBP and RPROP learning strategies are compared.

For the EBP learning algorithm the index performance E is defined as follows:

$$E = \frac{1}{2}\sum_{p=1}^{N}(O_d(p) - O(p))^2, \quad (6)$$

where N is the training set size, and the error is the signal defined as difference between the desired response $O_d(p)$ and the actual output neuron response $O(p)$.

To determine the optimal network learning parameter ϑ ($\boldsymbol{V}$, $\boldsymbol{W}$, $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, $\boldsymbol{b}_0$, $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, $\boldsymbol{C}$, $\boldsymbol{\sigma}$) that minimizes the index performance E a gradient method can be applied. Iteratively, the optimal learning parameters are approximated by moving in the direction of steepest descent:

$$\vartheta_{new} = \vartheta_{old} + \Delta\vartheta, \quad (7)$$

$$\Delta\vartheta = -\eta\nabla E = -\eta\frac{\partial E}{\partial\vartheta}, \quad (8)$$

where η is a user-selected positive learning constant (learning rate).

The choice of the learning constant depends strongly on the class of the learning problem and on the network architecture. The learning rate values ranging from 10^{-4} to 10^{2} have been reported throughout the literature as successful for many computational back-propagation experiments. For large constants, the learning speed can be drastically increased. However, the learning may not be exact, with tendencies to overshoot, or it may be never stabilized at any minimum. To accelerate the convergence of the EBP learning algorithm given in (7), the momentum method is usually applied. The method [1, 10] is given in (9) and involves supplementing the current learning parameter adjustment (8) with a fraction of the most recent parameter adjustment. This is usually done according to the formula:

$$\Delta\vartheta(n) = -\eta\nabla E(n) = -\eta\frac{\partial E(n)}{\partial\vartheta(n)} + \alpha\Delta\vartheta(n-1)\,, \qquad (9)$$

where the arguments *n* and *n-1* indicates the current and the most recent training step (instant time), respectively, and α is a user-selected positive momentum constant.

Typically, α is chosen between 0.1 and 0.8. To simplify the derivation of the learning algorithm, a linear time shifting operator can be defined by equation (10).

$$\begin{aligned}
&\left[\tilde{y}(n)\right] = \frac{B(z)}{A(z)}\left[net(n)\right]\\
&z^{-i}\left[net(n)\right] = net(n-i)\\
&A(z)\left[\tilde{y}(z)\right] = \tilde{y}(n) + a_1\tilde{y}(n-1) + a_2\tilde{y}(n-2)\\
&B(z)\left[net(n)\right] = b_0 net(n) + b_1 net(n-1) +\\
&\qquad\qquad + b_2 net(n-2)
\end{aligned} \qquad (10)$$

Using the time shifting operator defined in (10), three cases can be distinguished:
1) ϑ is a filter coefficient of the numerator $B(z)$:

$$\left.\frac{\partial\left[\tilde{y}(n)\right]}{\partial\vartheta}\right|_{\vartheta = b_i} = \left[D_\vartheta(n)\right] = \frac{z^{-1}}{A(z)}\left[net(n)\right]. \qquad (11)$$

2) ϑ is a filter coefficient of the denominator $A(z)$:

$$\left.\frac{\partial\left[\tilde{y}(n)\right]}{\partial\vartheta}\right|_{\vartheta = a_i} = \left[D_\vartheta(n)\right] = \frac{-z^{-1}}{A(z)}\left[\tilde{y}(n)\right]. \qquad (12)$$

3) ϑ is a neuron input weight :

$$\left.\frac{\partial\left[\tilde{y}(n)\right]}{\partial\vartheta}\right|_{\vartheta = w_j} = \left[D_\vartheta(n)\right] = \frac{B(z)}{A(z)}\left[u_j(n)\right]. \qquad (13)$$

$D_\vartheta(n)$ is a current parameter state within the dynamic filters described on the right side of equations (11), (12) and (13). Thus, to determine the change of the dynamic neuron activity depending on a filter, activation function and weight parameters, the gradient has to be calculated through time by the memory of the used filter.

EBP algorithm was used both, in batch and pattern learning procedure. Pattern means that learning parameters adaptation occurs for each input-output data pair or pattern from learning data set. Thus the one learning step implies the pattern number of changes of the learning parameters. On the other side the batch procedure implies only one change of the learning parameters per one learning step. Therefore one can expect that the batch learning procedure requires less processing time. In all experiments, we used the same constant learning parameters as follows:

$$\eta = 0.02\,, \quad \alpha = 0.8\,. \qquad (14)$$

4. RPROP LEARNING ALGORITHM

RPROP (**R**esilient back-**PROP**agation) learning algorithm is typical representative of batch learning procedures. Many reports points to its fast convergence in learning of feed forward neural networks. Our goal was to determine the possibility of usage the RPROP learning algorithm for recurrent neural networks. In this paper the New Weight-Backtracking RPROP scheme was used [7]. All learning parameters ϑ can be adapted according to the pseudo-code as follows:

for each ϑ **do**
  if $\frac{\partial E}{\partial\vartheta}(n-1)\bullet\frac{\partial E}{\partial\vartheta}(n)>0$ **then**
    $\Delta(n) = \min(\Delta(n-1)\bullet\eta^{+}, \Delta_{\max})$
    $\Delta\vartheta^{(n)} = -\text{sign}\left(\frac{\partial E}{\partial\vartheta}(n)\right)\bullet\Delta^{(n)}$
    $\vartheta(n+1) = \vartheta(n) + \Delta\vartheta(n)$
  elseif $\frac{\partial E}{\partial\vartheta}(n-1)\bullet\frac{\partial E}{\partial\vartheta}(n)<0$ **then**
    $\Delta(n) = \max(\Delta(n-1)\bullet\eta^{-}, \Delta_{\min})$
    if $E(n)>E(n-1)$ **then**
      $\vartheta(n+1) = \vartheta(n) - \Delta\vartheta(n-1)$
      $\frac{\partial E}{\partial\vartheta}(n) = 0$
  elseif $\frac{\partial E}{\partial\vartheta}(n-1)\bullet\frac{\partial E}{\partial\vartheta}(n)=0$ **then**
    $\Delta\vartheta(n) = -\text{sign}\left(\frac{\partial E}{\partial\vartheta}(n)\right)\bullet\Delta(n)$
    $\vartheta(n+1) = \vartheta(n) + \Delta\vartheta(n)$
  fi
od (15)

In all experiments, we used the same constant learning parameters as follows:

$$\eta^+ = 1.2\ ,\quad \eta^- = 0.8,\ \Delta^{(0)} = 0.02\ ,$$
$$\Delta_{\min} = 0.0001\ ,\ \Delta_{\max} = 5.0\ . \tag{16}$$

5. XOR CLASSIFICATION PROBLEM

One of the famous neural networks testbed is the logical XOR problem given in table 1. XOR is a typical representative of the class of linear non-separable classification problems. According to the table 1, the input layer consists of 2 neurons, and output layer consists of one static neuron. For the hidden layer we used 2 dynamic neurons.

Table 1. XOR problem

Input Z1	Input Z2	Desired output D
0	0	0
0	1	1
1	0	1
1	1	0

The networks were trained until the error index NRMS dropped to 0.01. The comparison of the learning algorithms is given in table 2.

Table 2. The learning results

Learning			
MDNN	EBP - Batch	EBP - Pattern	RPROP
NLS*	8571	196	59
CPU**	100	2.52	1.10
NRMS	0.01	0.01	0.01

* - the Number of Learning Steps
** - required CPU time (%)

It is obvious that EBP algorithm with batch learning procedure is bad choice. Such algorithm has very slowly convergence. It requires a lot of learning steps (epochs) i.e. CPU time. On the other hand the EBP learning algorithm with pattern learning procedure needs less learning steps. It is almost 40 times faster than batch procedure, although it's one learning step requires more CPU time than one batch learning step. But if one chose the RPORP algorithm, the required learning steps and CPU time can be additionally reduced. In comparison with EBP pattern learning procedure, the RPROP learning algorithm in this experiment is more than twice faster.

6. GLASS-MACKEY TIME SERIES PREDICTION

Many conventional signal processing tests, such as correlation function analysis, cannot distinguish deterministic chaotic behaviour from stochastic noise. Particularly difficult systems to predict are those that are nonlinear and chaotic. It is known that chaos has a technical definition based on nonlinear, dynamic systems theory [9]. Examples of chaotic systems in nature include chemical reactions, plasma physics, turbulence in fluids, lasers, to name a few. When parameters are varied, chaotic systems also display the full range of nonlinear behaviour (limit cycles, fixed points, etc.). Therefore chaotic systems provide a good testbed in which to investigate techniques of nonlinear signal processing, such as neural networks. Lapedes and Farber [9] suggested the Glass-Mackey time series as a good benchmark for learning algorithms, because it has a simple definition, yet its elements are hard to predict (the series is chaotic). Glass-Mackey equation given in (15) is a nonlinear differential delay equation with an initial condition specified by an initial function defined over a strip with τ.

$$\dot{x} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \tag{17}$$

Choosing the initial function to be constant function, with a = 0.2, b = 0.1 and τ = 17 yields a time series $x(t)$ obtained by equation (17), that is chaotic with a fractal attractor of dimension 2.1. Increasing τ to 30 yields more complicated evolution and fractal dimension (d_A) of 3.5. The time series for 1000 time steps for τ = 30 (time in units of τ) is plotted in figure 3.

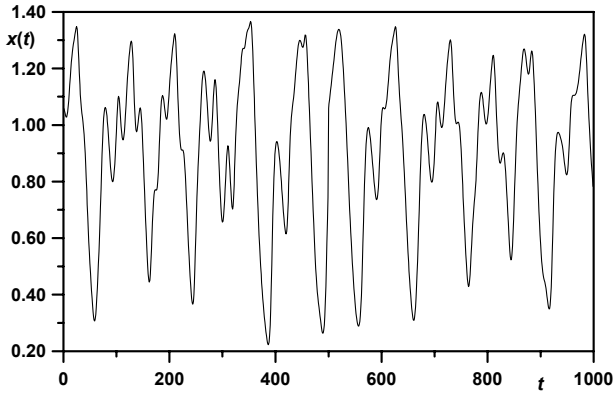

Figure 3. The Glass-Mackey Time Series with a = 0.1, b = 0.2, τ = 30

The goal of the task is to use known values of the time series up to the point $x(t)$, to predict the value $x(t+P)$ at some point P in the future . The standard method for this type of prediction is to create a mapping $f()$ as follows :

$$x(t+P) = f\left(x(t), x(t-\underline{\Delta}), x(t-2\underline{\Delta}), \ldots, x(t-m\underline{\Delta})\right), \tag{18}$$

where P is a prediction time into the future, $\underline{\Delta}$ is a time delay , and m is an integer.

According to the (18) an attractor can be reconstructed from a time series by using a set of time delayed samples of a series. By choosing P=$\underline{\Delta}$ [9] it is possible to predict the value of time series at any multiple of $\underline{\Delta}$ time steps in the future, by feeding the output back into the input and iterating the solution. In this study we choose to use P=Δ=6, since results can be compared with previous experiments where P=6. Takens theorem [11] states the range for dimension of the attractor (d_A):

$$d_A < m+1 < 2d_A+1 \tag{19}$$

For $\tau = 30$ we choose *m*=4. It is obvious that for *P*=∆=6 and *m*=4 the expansion (18) has the following form:

$$x(t+6) = f(x(t), x(t-6), x(t-12), x(t-18), x(t-24)) \quad (20)$$

Takens theorem unfortunately gives no information on the form of the *f*() in equation (20). Therefore, it is necessary to point out that the neural networks provide a robust approximating procedure for continuous *f*().

According to the equation (20) the input layer consists of 5 neurons (input buffer), and output layer consists of one static neuron. In the hidden layer we suggested 10 dynamic neurons. For training the neural network, we used first 500 values plotted in figure 3. The networks were trained until the error index NRMS dropped to 0.05. The comparison of the learning algorithms is given in table 3.

Table 3. Comparison of the learning algorithms

	Learning		
MDNN	EBP - Batch	EBP - Pattern	RPROP
NLS*	16805	2127	1987
CPU**	100	15.9	11.6
NRMS	0.05	0.05	0.05

* - the Number of Learning Steps
** - required CPU time (%)

The obtained learning results are quite similar with the results in previous benchmark problem. Again the RPROP learning algorithm shows the best learning performances. He requires the least number of learning steps i.e. CPU time.

After learning the networks generalization features were tested through many tests. The trained network were used to predict new sets of values *x*(*t*) in the future. Some of the tests results are given in table 4.

Table 4. MDNN testing procedure

	Testing (NRMS)		
Test	EBP - Batch	EBP- Pattern	RPROP
1	0.052	0.084	0.058
2	0.046	0.068	0.059
3	0.047	0.081	0.061
4	0.048	0.077	0.058
5	0.047	0.076	0.061

According to the table 4., all networks generalizes very well and in many practical applications such differences can be neglected.

In this experiment the EBP algorithm with batch learning procedure shows the best generalization features. However, such generalization advantages are not significant toward the fact that EBP batch learning algorithm requires the huge number of learning steps i.e. CPU processing time. Also, some of that one can perceive throughout the plots with test results given in figures 4-9. The figures 4, 5 and 6 reveal that all neural networks solved the problem.

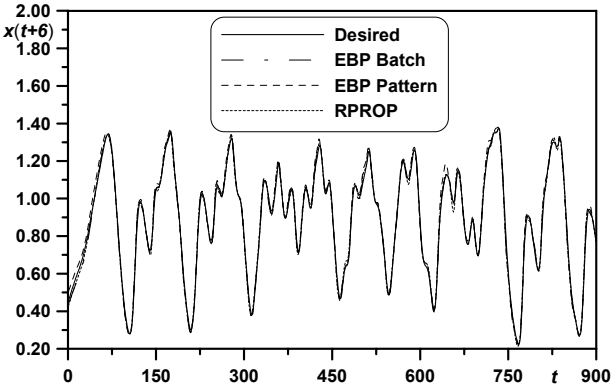

Figure 4. DNN output for Test 1.

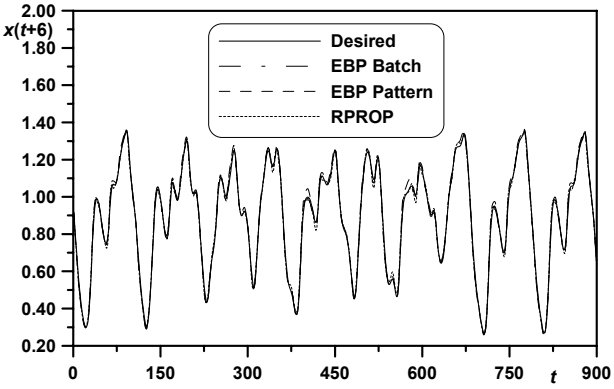

Figure 5. DNN output for Test 2.

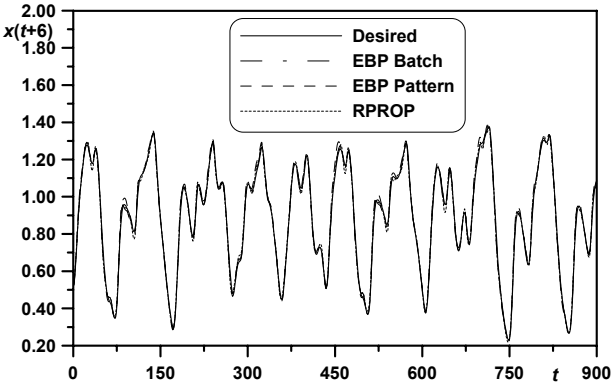

Figure 6. DNN output for Test 5.

Real differences between generalization features of the used learning algorithms can be better visible in figures 7, 8 and 9. These figures present some characteristic zooms of figures 4, 5 and 6. It is obvious that differences in generalization for all learning algorithms are negligible.

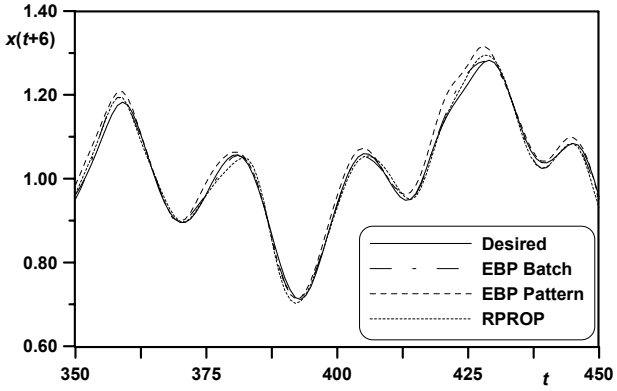

Figure 7. The set of 100 data points from test 1.

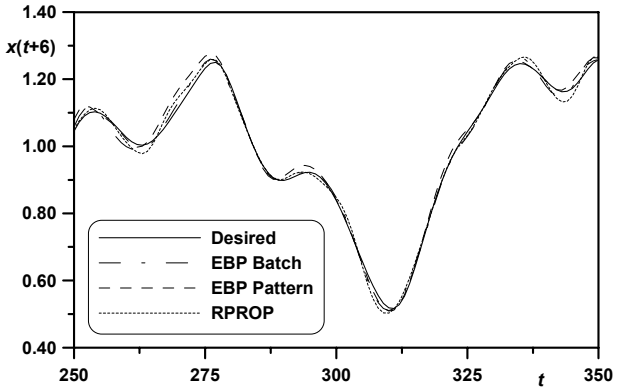

Figure 8. The set of 100 data points from test 2.

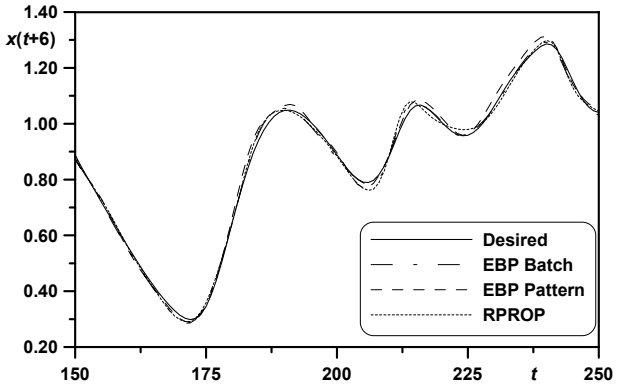

Figure 9. The set of 100 data points from test 5.

7. CONCLUSION

We established a dynamic neuron model, which processes multi inputs and does not require past values of the process measurements or prior information about its activity functions.

The main advantage of the proposed dynamic neuron model is that it reduces the network input space. Additionally, because of elimination of the Bias neuron, the neural network with adaptive Gauss activation function has the less number of neurons and learning parameters. Trained neural network with smaller topology has much faster response. With reduced CPU time and memory needed, such neural network is more promising in real-time domain applications. With RPROP learning algorithm MDNN learns much faster and at the same time has very good generalization property. The proposed neural network offers a great potential in signal filtering and in solving many problems that occurs in system modelling with a special emphasis on the systems with characteristics such as nonlinearity, time delays, saturation or time-varying parameters.

8. REFERENCES

[1] Novakovic, B., Majetic, D., Siroki, M., 1998, *Artificial Neural Networks*, Faculty of Mechanical Engineering and Naval Architecture, Zagreb, Croatia.

[2] Smagt, P., 1994, Minimization methods for training feed-forward networks, Neural Networks 7, pp. 1-11.

[3] Nguyen, D., Widrow, B., 1990, Improving the Learning Speed of Two-Layer Networks by Choosing Initial Values of the Adaptive Weights, Proceedings of International Joint Conference on Neural Networks, Sand Diego, Vol. 3, pp. 21-26.

[4] Darken, C., Moody, J., 1991, Note of Learning Rate Schedules for Stochastic Optimization, Neural Information Processing Systems, pp. 832-838.

[5] Lawrence, S., Giles, C.L., Tsoi, A.C., 1996, What Size of Neural Network Gives Optimal Generalization, Convergence Properties of Back-Propagation, Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute of Advanced Computer Studies, Maryland.

[6] Kecman, V., 2001, Learning and Soft Computing, MIT Press, England.

[7] Igel, C., Husken, M., 2000, Improving the RPROP Learning Algorithm, Proceedings of the Second International Symposium on Neural Computation, NC'2000, ICSC Academic Press, pp. 115–121.

[8] Brezak, D., Majetic,D., Udiljak, T., Novakovic, B., Kasac, J., 2006, Adaptive Control Model for Maintaining Tool Wear Rate in the Predefined Cutting Time, 17.th DAAAM International Symposium, Vienna, pp. 63-64.

[9] Lapedes, A.S., Farber, R., 1987, Nonlinear Signal Processing Using Neural Networks: Prediction And System Modeling, Technical Report, Los Alamos National Laboratory, Los Alamos, New Mexico.

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

[11] Takens, T., 1981, Detecting Strange Attractor In Turbulence, Lecture Notes in Mathematics, D.Rand, L.Young (editors), Springer Berlin, pp. 366.