Dynamic neural network with adaptive Gauss neuron activation function

Dubravko Majetic, Danko Brezak, Branko Novakovic & Josip Kasac

Abstract: *An attempt has been made to establish a nonlinear dynamic discrete-time neuron model, the so called Dynamic Elementary Processor (DEP). This dynamic neuron disposes of local memory, in that it has dynamic states. To accelerate the convergence of proposed extended dynamic error-back propagation learning algorithm, the adaptive neuron activation function and momentum method are applied. Instead of most popular bipolar and unipolar Sigmoid neuron activation functions, the Gauss activation function with adaptive parameters is proposed. Based on the DEP neuron with adaptive activation function in hidden layer, a Dynamic Multi Layer Neural Network is proposed and tested in prediction of a Glass-Mackey time series. The learning results are presented in terms that are insensitive to the learning data range and allow easy comparison with other learning algorithms, independent of machine architecture or simulator implementation.*

Keywords: *dynamic neural network, adaptive neuron activation function, momentum, prediction, Glass-Mackey time series*

1. Introduction

Since artificial neural networks can effectively represent complex nonlinear functions, they proved to be a very useful tool in identifying of highly nonlinear systems. The neuron models most commonly applied are the Feed Forward Perceptron used in multi layer networks, and the Radial Basis Function neuron (RBF) used in RBF neural networks with one step learning algorithm. Both networks are proved to be universal approximators of any static nonlinear mapping. They are capable of identifying any nonlinear unique state function to arbitrary desired accuracy.
Recently, interests have been increasing towards the usage of neural networks for modeling and identification of dynamic systems. These networks, naturally, involve dynamics in the form of feedback connections and are known as Recurrent Neural Networks. Several learning methods for recurrent networks have been proposed in literature. Most of these methods rely on the gradient methodology and involve the computation of partial derivatives, or sensitivity functions. In this sense, they are extension of the well-known error back propagation algorithm for feedforward neural networks (Zurada, 1992). Examples of such learning algorithms (Narendra, 1990), (Kosmatopoulos, 1992) include the error-back propagation through-time algorithms, the real-time recurrent learning algorithm, the recurrent back-propagation, and dynamic back-propagation. Most of these methods, demanding certain knowledge of dynamic system behavior and in their application one must estimate the order of identified system in advance. When using dynamic neural network proposed in this paper, one does not have to estimate the order of the identified system in advance.
Unfortunately error-back propagation learning algorithm can be very slow for practical applications. Over the last years many improvement strategies have been developed to speed up error-back propagation and improve neural network learning and generalization features. All of these strategies can be separated in three basic categories. The first category deals with the improvement of the error back-propagation learning algorithm (Smagt, 1994). The second category deals with the neurons weights initial values (Nguyen & Widrow, 1990; Darken & Moody, 1991; Kecman, 2001) and the third category deals with neural network topology optimization (Lawrence at al., 1996).
In this paper the hidden layer neuron structure modification and activation function, with adaptive parameters are proposed. With applying only momentum method for speeding up the learning algorithm and proposed neuron activation function, neural network training procedure can be much efficient and faster. More over, the neural network with proposed activation function has the less number of neurons. And finally, trained neural network with smaller topology has much faster response, which is more promising in real-time domain applications.

2. Dynamic neuron model

The basic idea of proposed dynamic neuron concept is to introduce some dynamics to the neuron transfer function, 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. Such a 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. The structure of a proposed dynamic neuron model is plotted in Fig. 1. The filter input and output at time instant (n) are given in (1) and (2) respectively (Novakovic at al., 1998):

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

$$\begin{aligned} \tilde{y}(n) &= b_0 net(n) + b_1 net(n-1) + b_2 net(n-2) - \\ &- a_1 \tilde{y}(n-1) - a_2 \tilde{y}(n-2). \end{aligned} \tag{2}$$

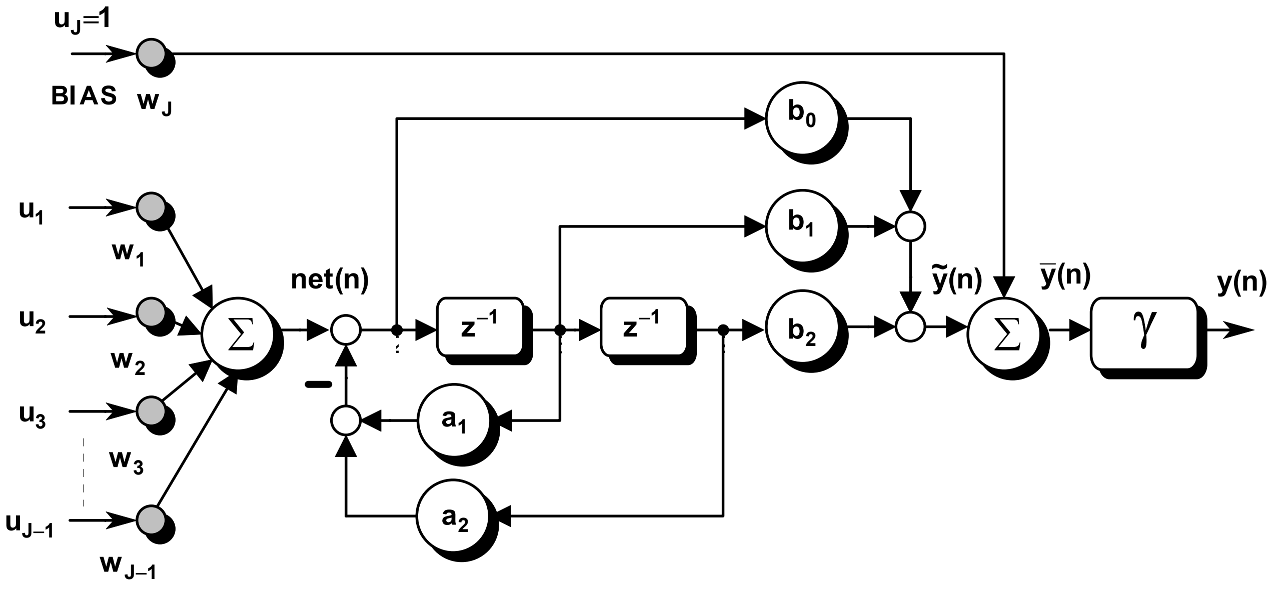

Fig. 1. Dynamic neuron model.

The input of the neuron activation function (AF) is given in (3), and widely used nonlinear Sigmoid unipolar activation function and Gauss activation function with adaptive parameters are described in (4) and (5) respectively.

$$\bar{y}(n) = \tilde{y}(n) + w_J u_J \,, \tag{3}$$

where $\boldsymbol{u_J = 1}$ represents a threshold unit, also called Bias.

$$y(n) = \gamma(\bar{y}(n)) = \frac{1}{1 + e^{-\bar{y}(n)}} \,, \tag{4}$$

$$y(n) = \gamma(\bar{y}(n)) = e^{-\frac{1}{2}\left(\frac{\bar{y}-c}{\sigma}\right)^2} \,. \tag{5}$$

3. Dynamic neural network

The neural network (Fig. 2) proposed in this study has three layers. Each i-th neuron in the first, input layer has a single input which represents the external input to the neural network. The second layer is consisting of dynamic neurons, which are presented by Fig. 1. Each j-th dynamic neuron in hidden layer has an input from every neuron in the first layer, and one additional input with a fixed value of unity usually named as Bias.

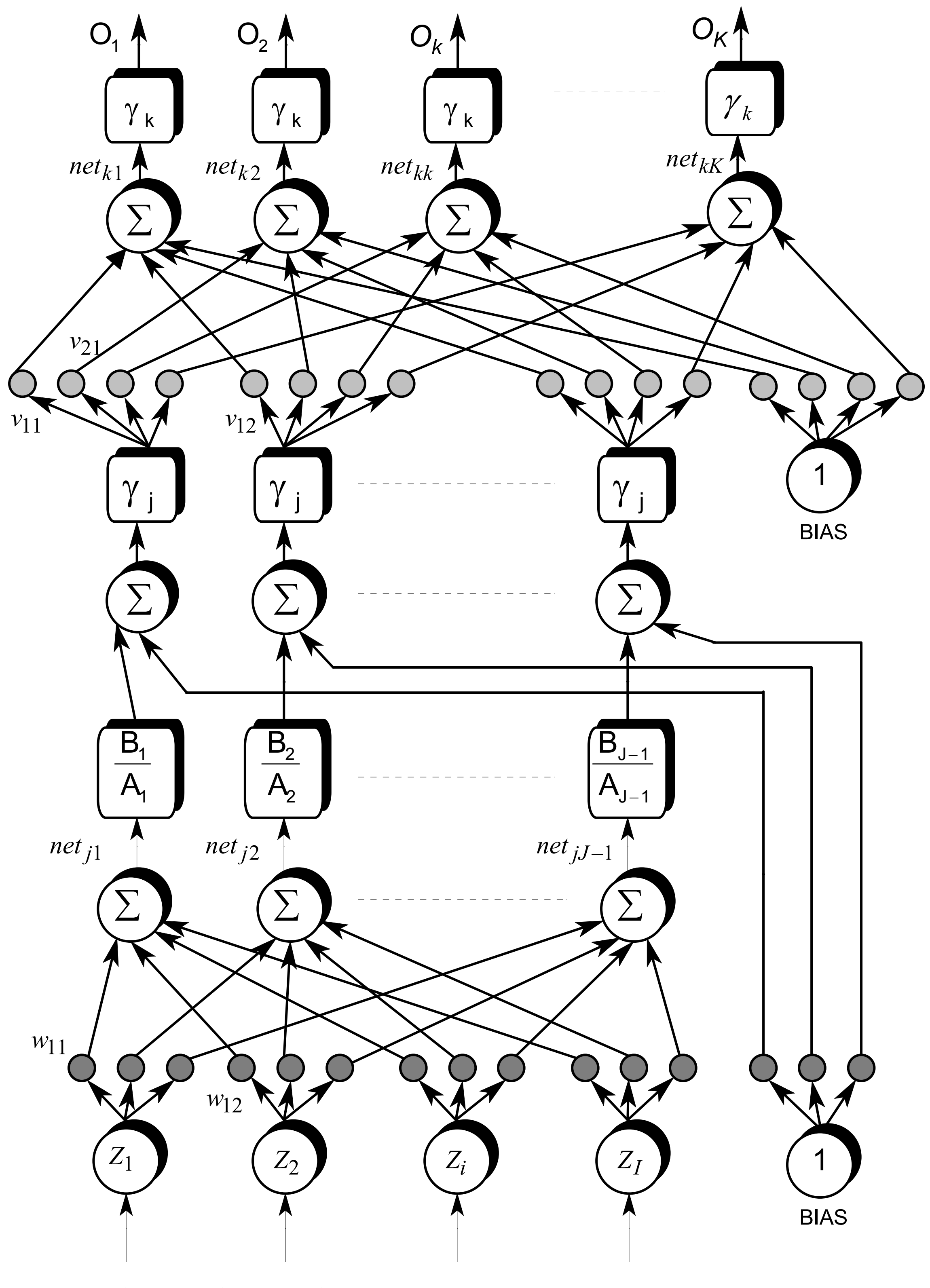

Fig. 2. Dynamic neural network.

In hidden layer adaptive Gauss activation function is suggested. Gauss activation function is well known as basic activation function for Radial Basis Function (RBG) neural networks. RBF is a real specialist for classification problems, and learns in one step learning algorithm.

Each k-th neuron in the third, output layer of proposed dynamic neural network has an input from every neuron in the second layer and, like the second layer one additional input with fixed value of unity (Bias). The linear activation function given in (6) is a chosen activation function for all static neurons (Fig. 3) in output layer. Such static neuron topology is widely used in feed-forward error-back propagation neural networks.

$$O_k(n) = \gamma_k(net_k(n)) = net_k(n) \tag{6}$$

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

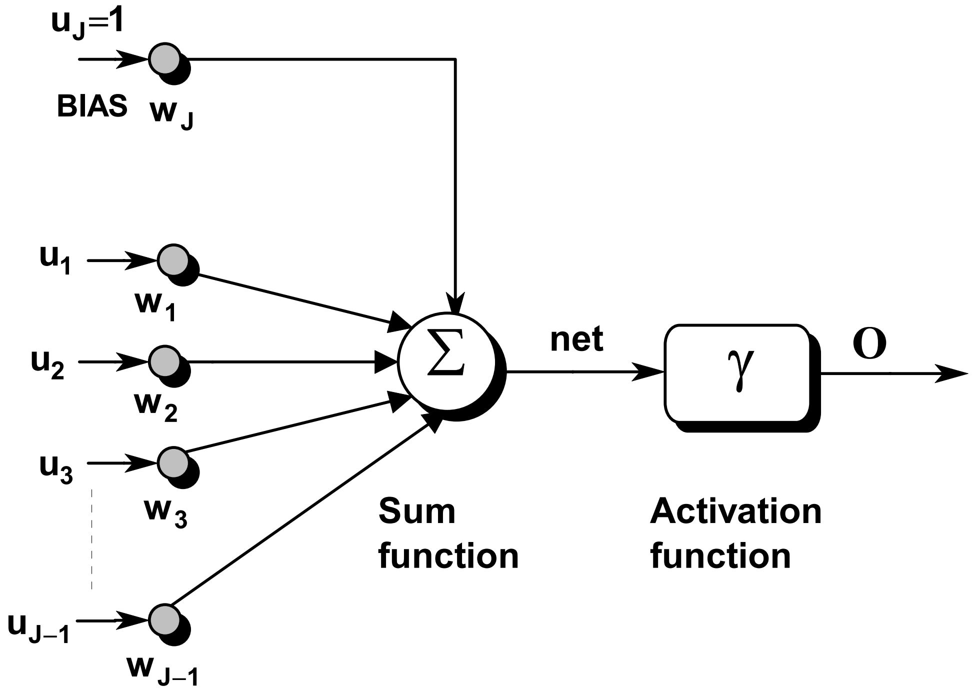

Fig. 3. Static neuron model in output layer with linear activation function.

4. Learning algorithm

The goal of the learning algorithm is to adjust the neural network learning parameters ϑ in order to determine the optimal parameter set that minimizes a performance index E (Zurada, 1992) as follows :

$$E = \frac{1}{2}\sum_{n=1}^{N}(O_d(n) - O(n))^2 , \tag{7}$$

where N is the training set size, and the error is the signal defined as difference between the desired response $O_d(n)$ and the actual neuron response $O(n)$. This error is propagated back to the input layer through the dynamic filters of dynamic neurons in hidden layer. Iteratively, the optimal parameters output layer weights (***V***), hidden layer weights (***W***), filter coefficients (*a*1, *a*2, *b*0, *b*1, *b*2) and DEP activation function parameters (c and σ, (5)) for all processing elements in hidden layer are approximated by moving in the direction of steepest descent (10):

$$\vartheta = \{ \boldsymbol{V}, \boldsymbol{W}, a_{1j}, a_{2j}, b_{oj}, b_{1j}, b_{2j}, c_j, \sigma_j \} \qquad j = 1,2,...,J-1\,, \tag{8}$$

where J-1 is number of hidden nodes. J stands for threshold neuron, also known as Bias neuron with fixed unity output (Fig. 2).

$$\vartheta_{new} = \vartheta_{\ old} + \Delta\vartheta\,, \tag{9}$$

$$\Delta\vartheta = -\eta \nabla E = -\eta \frac{\partial E}{\partial \vartheta}, \tag{10}$$

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^{-3} to 10 have been reported throughout the technical 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 learning algorithm given in (9), momentum method is applied. The momentum method is given in (11) and involves supplementing the current learning parameter adjustment (10) with a fraction of the most recent parameter adjustment. This is usually done according to the formula

$$\Delta\vartheta(n) = -\eta \frac{\partial E(n)}{\partial \vartheta(n)} + \alpha \Delta\vartheta(n-1), \tag{11}$$

where α is a user-selected positive learning constant.

Typically, α is chosen between 0.1 and 0.8. The arguments n and $n\text{-}1$ are used to indicate the current and the most recent training step (instant time), respectively. To simplify the derivation of the learning algorithm, a linear time shifting operator can be defined by expression (12) as follows,

$$\begin{gathered}
[\tilde{y}(n)] = \frac{B(z)}{A(z)}[net(n)], \\
z^{-i}[net(n)] = net(n-i)\,, \\
A(z)[\tilde{y}(z)] = \tilde{y}(n) + a_1\tilde{y}(n-1) + a_2\tilde{y}(n-2)\,, \\
B(z)[net(n)] = b_0 net(n) + b_1 net(n-1) + b_2 net(n-2).
\end{gathered} \tag{12}$$

According to the Fig. 1 it is obvious that :

$$\frac{\partial y(n)}{\partial \vartheta(n)} = \frac{\partial y(n)}{\partial \tilde{y}(n)} \frac{\partial \tilde{y}(n)}{\partial \tilde{y}(n)} \frac{\partial \tilde{y}(n)}{\partial \vartheta(n)} = \gamma' \frac{\partial \tilde{y}(n)}{\partial \tilde{y}(n)} \frac{\partial \tilde{y}(n)}{\partial \vartheta(n)}. \tag{13}$$

Therefore, the used activation function in output and hidden layer has to be differentiable. Using the time shifting operator defined in (12), four cases can be distinguished:

1) ϑ is a filter coefficient of the numerator $B(z)$:

$$\left.\frac{\partial[\tilde{y}(n)]}{\partial\vartheta}\right|_{\vartheta=b_i} = [D_\vartheta(n)] = \frac{z^{-1}}{A(z)}[net(n)], \tag{14}$$

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

$$\left.\frac{\partial[\tilde{y}(n)]}{\partial\vartheta}\right|_{\vartheta=a_i} = [D_\vartheta(n)] = \frac{-z^{-1}}{A(z)}[\tilde{y}(n)], \tag{15}$$

3) ϑ is a neuron input weight :

$$\left.\frac{\partial[\tilde{y}(n)]}{\partial\vartheta}\right|_{\vartheta=w_j} = [D_\vartheta(n)] = \frac{B(z)}{A(z)}[u_j(n)], \tag{16}$$

4) ϑ is a neuron threshold :

$$\left.\frac{\partial y(n)}{\partial\vartheta}\right|_{\vartheta=w_J} = \frac{\partial\gamma}{\partial w_J}. \tag{17}$$

$D_\vartheta(n)$ is a current parameter state within the dynamic filters described on the right side of equations (14), (15), and (16). Thus, to determine the change of the dynamic neuron activity depending on a filter and weight parameters, the gradient has to be calculated through time by the memory of the used filter. The weight adjustment in output layer (Fig. 2) can be obtained by expansion (19).

$$\left.\frac{\partial E(n)}{\partial\vartheta(n)}\right|_{\vartheta=v_{kj}} = \frac{\partial E(n)}{\partial O_k(n)}\frac{\partial O_k(n)}{\partial net_k(n)}\frac{\partial net_k(n)}{\partial\vartheta(n)}, \tag{18}$$

$$\left.\frac{\partial E(n)}{\partial\vartheta(n)}\right|_{\vartheta=v_{kj}} = -(d_k(n) - O_k(n))y_j(n). \tag{19}$$

Finally, a measure of performance must be specified**.** All learning and test error measures will be reported using non-dimensional error index NRMS, Normalized Root Mean Square error. “Normalized” means that the root mean square is divided by the standard deviation of the target data (Lapedes & Farber, 1987). Thus the resulting error index, or index of accuracy is insensitive to the dynamic range of the learning data, and allows easy comparison with other learning algorithms, independent of machine architecture or simulator implementation.

5. Experimental results

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 (Lapedes & Farber, 1987). 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 test bed in which to investigate techniques of nonlinear signal processing, such as neural networks.

Lapedes and Farber (Lapedes & Farber, 1987) 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 (20) is a nonlinear differential delay equation with an initial condition specified by an initial function defined over a strip with τ.

$$\dot{x} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \qquad (20)$$

Choosing the initial function to be constant function, with $a = 0.2$, $b = 0.1$ and $\tau = 17$ yields a time series $x(t)$ obtained by equation (20), that is chaotic with a fractal attractor of dimension 2.1 . Increasing τ to 30 yield more complicated evolution and fractal dimension (d_A) of 3.5. The time series for 1000 time steps for $\tau = 30$ (time in units of τ) is plotted in Fig. 4.

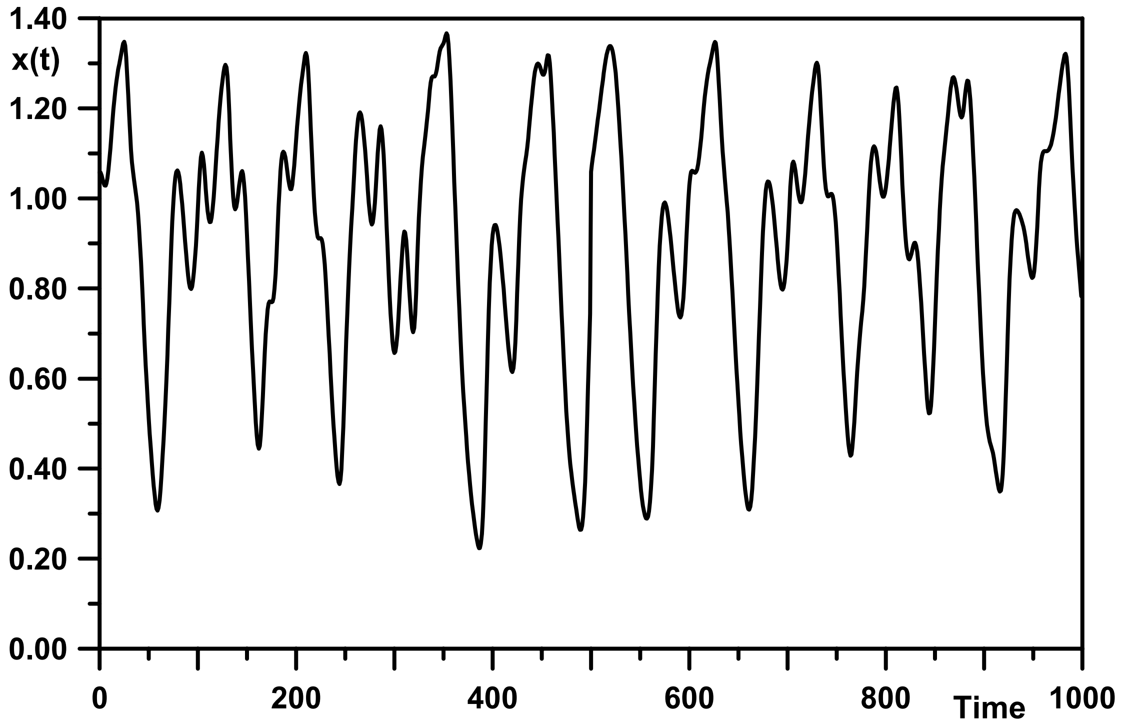

Fig. 4. Glass-Mackey time series.

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 :

$$x(t+P) = f\big(x(t), x(t-\Delta), x(t-2\Delta), \ldots, x(t-m\Delta)\big). \qquad (21)$$

where P is a prediction time into the future, Δ is a time delay , and m is an integer. According to the equation (21) an attractor can be reconstructed from a time series by using a set of time delayed samples of a series. By choosing $P = \Delta$ (Lapedes & Farber, 1987) it is possible to predict the value of time series at any multiple of Δ time steps in the future, by feeding the output back into the input and iterating the solution. In this study we choose to use $P = \Delta = 6$, since results can be compared with previous experiments where $P = 6$. Takens theorem (Takens, 1981) states the range for dimension of the attractor (d_A) :

$$d_A < m + 1 < 2d_A + 1. \tag{22}$$

For $\tau = 30$ we choose m=4. It is obvious that for $P = \Delta = 6$ and $m = 4$ the expansion (21) has the following form :

$$x(t+6) = f\big(x(t), x(t-6), x(t-12), x(t-18), x(t-24)\big). \tag{23}$$

Takens theorem unfortunately gives no information on the form of the *f*() in (23). Therefore, it is necessary to point out that the neural networks provide a robust approximating procedure for continuos *f*().

The network which will be used to predict the chaotic system (23) is given in Fig. 2. According to the equation (23) the input layer consists of 5 neurons (input buffer), and output layer consists of one static neuron with linear activation function. For hidden layer we suggested 5 dynamic neurons. Lapedes and Farber (Lapedes & Farber, 1987) for the same task used 20 hidden static neurons arranged in two hidden layer architecture.

For training the neural network described above, we used first 500 values plotted in Fig. 4. Training started with random weights values between -1 and +1, while the filter coefficients a_1 and a_2 were initialized to zeros to support a stable learning procedure. The network was trained with $\eta = 0.01$ and $\alpha = 0.8$.

The trained network were used to predict new sets of values $x(t)$ in the future. Learning and testing results are given in Table 1.

Neuron AF	Unipolar Sigmoid		Adaptive Gauss	
Network Topology	5-10-1	5-5-1	5-10-1	5-5-1
Learning Epoch's	70000	80000	35000	50000
Learning (NRMS)	0,069	0,053	0,027	0,057
Test 1. (NRMS)	0,069	0,071	0,048	0,043
Test 2. (NRMS)	0,071	0,067	0,052	0,058
Test 3. (NRMS)	0,073	0,078	0,050	0,052

Table 1. Learning and test results.

It is obvious that proposed neuron structure modification concerning integrated ARMA filter and adaptive Gauss activation function gives very promising results. The goal was achieved with only 5 hidden nodes. Neural network with adaptive activation function learns faster and have smaller topology. The table 1 shows that neural network with adaptive Gauss activation function for almost the same learning error rate, needs almost twice less learning steps. More over, neural network with adaptive activation function perform mapping in all tests with much smaller error rate (NRMS). To illustrate the both networks generalization capability, the 300 data points of test 2 are given in Fig. 5.

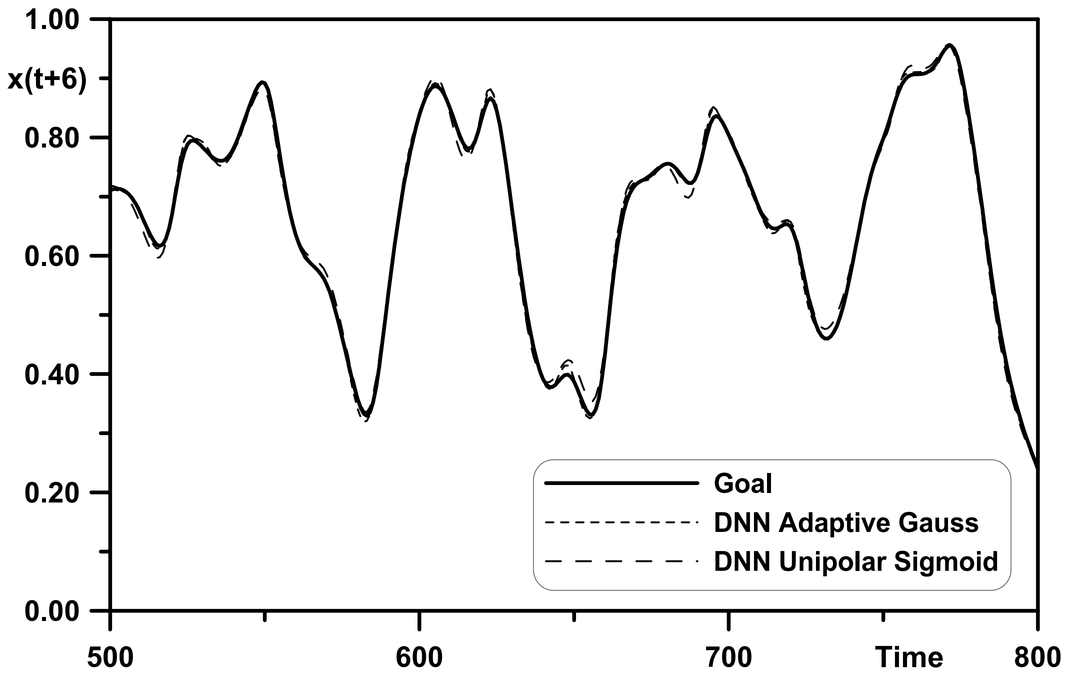

Fig. 5. Test 2. for the 5-5-1 neural network topology.

According to the Fig. 5 it is obvious that both neural networks solved the problem. Still, neural network with adaptive activation function perform better mapping. This is clearly presented with Fig. 6 where we made a zoom of one part of Fig. 5.

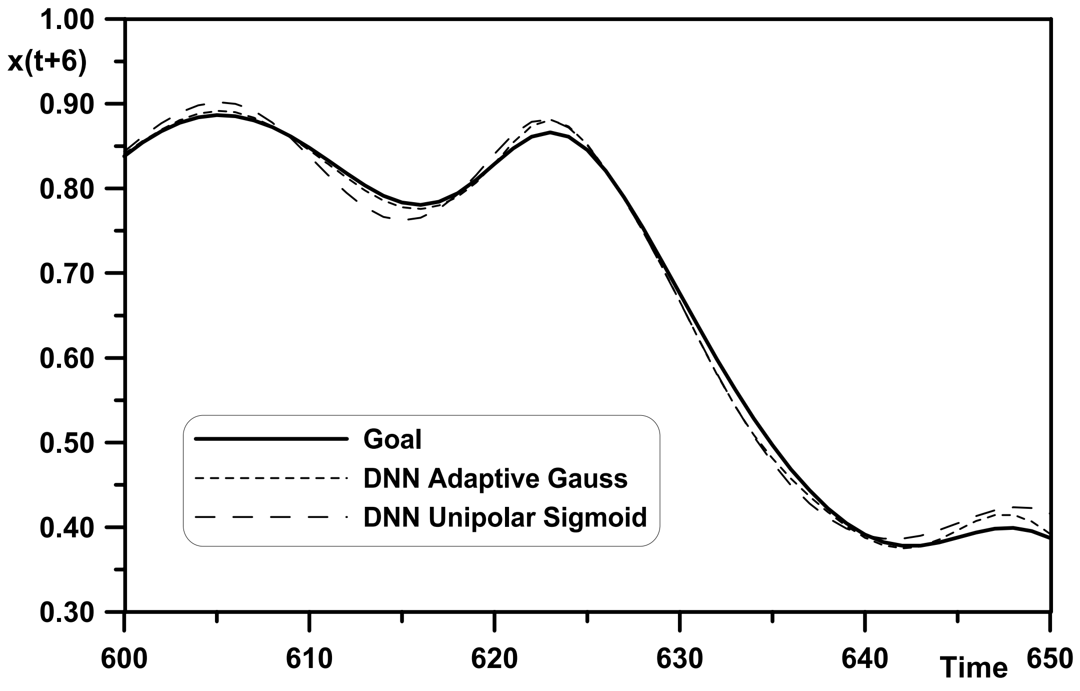

Fig 6. The set of 50 data points from test 2.

All other experimental results shows the same results. Another data set of 500 data points of test 3 is given in Fig. 7. As in previous test, both neural networks generalized well and made a good prediction of chaotic system dynamic behavior. Again, the neural network with adaptive activation function perform better mapping.

In Fig. 8 we zoom one part of Fig. 7. It is easy to see the real domination of neural network with adaptive Gauss activation function.

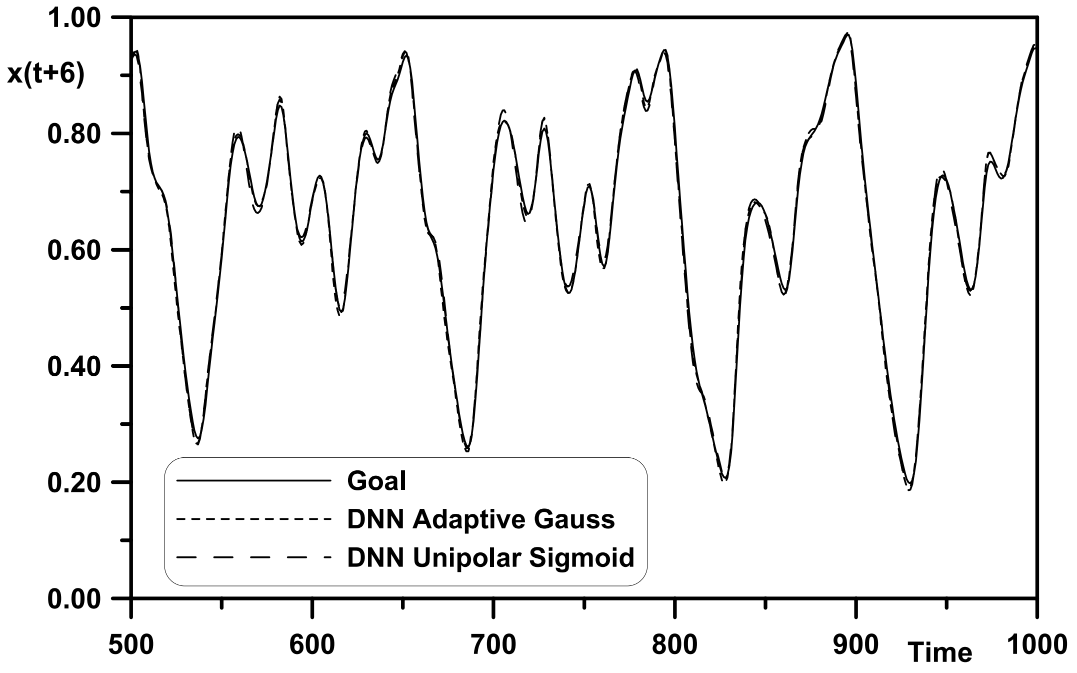

Fig. 7. Test 3. for the 5-5-1 neural network topology.

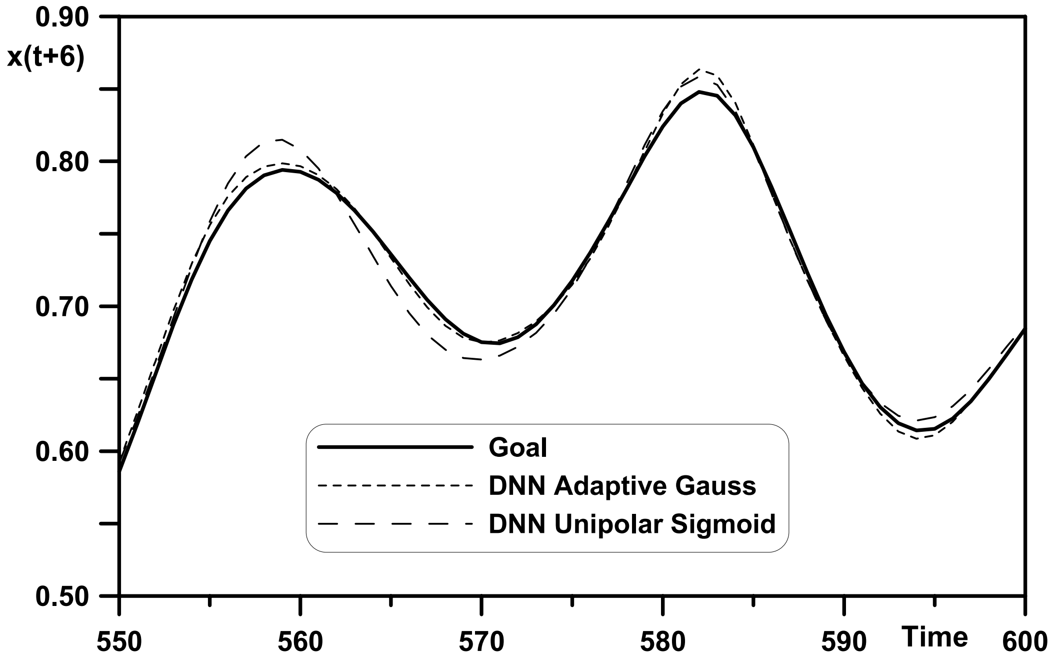

Fig. 8. The set of 50 data points from test 3.

6. Conclusion

Within this approach a Multi Layer Perceptron with distributed dynamics based on the DEP neuron model and adaptive activation function was proposed to predict a time series of nonlinear chaotic system. An attempt was made within this approach to establish a basic 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 proposed dynamic neuron model is that it reduces the network input space. The advantage of adaptive activation function is speeding up the learning algorithm. For the same learning error, neural network with adaptive Gauss activation function need almost twice less learning steps and in the same time obtain better generalization then the neural network with unipolar Sigmoidal activation function. Adaptive Gauss activation function shows the great possibility in solving the local minima's problems.

The proposed dynamic neural network offers a great potential in solving many problems that occurs in system modeling with a special emphasis on the systems with characteristics such as nonlinearity, time delays, saturation or time-varying parameters.

7. References

Darken, C. & Moody, J. (1991). Note of Learning Rate Schedules for Stochastic Optimization, *Neural Information Processing Systems*, pp. 832-838.

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.

Lawrence, S.; Giles, C.L. & Tsoi, A.C. (1996). What Size of Neural Network Gives Optimal Generalization, Convergence Properties of Backpropagation, *Technical Report UMIACS-TR-96-22 and CS-TR-3617*, Institute of Advanced Computer Studies, Maryland.

Kecman, V. (2001). *Learning and Soft Computing*, MIT Press, ISBN: 0-262-11255-8, Cambridge, Massachusetts, England.

Kosmatopoulos, E.B. (1992), Ioannou, P.A. & Christodoulou M.A. Identification of Nonlinear Systems Using New Dynamic Neural Network Structure, *Proceedings of the 31st Conference on Decision and Control*, pp. 20-25, 1992, Tucson, Arizona.

Narendra ,K.S. & Parthasarathy, K., (1990). Identification and Control of Dynamical Systems Using Neural Networks. *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, pp. 4-27.

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*, Vol. 3, pp. 21-26, Sand Diego, CA, USA.

Novakovic, B.; Majetic, D. & Siroki, M. (1998). *Artificial Neural Networks*, Faculty of Mechanical Engineering and Naval Architecture, ISBN: 953-6313-17-0, Zagreb, Croatia.

Takens, F. (1981). *Detecting Strange Attractor In Turbulence*, Lecture Notes in Mathematics, D.Rand, L.Young (editors), Springer Berlin, pp. 366-380.

Smagt, P. (1994). Minimization methods for training feed-forward networks, *Neural Networks 7*, pp. 1-11.

Zurada, J.M. (1992). *Artificial Neural Systems*, W.P. Company, ISBN: 0-314-93391-3, USA.