Journal of Applied Sciences

ISSN 1812-5654

Journal of Applied Sciences 8 (8): 1534-1540, 2008
ISSN 1812-5654
© 2008 Asian Network for Scientific Information

Short-Term Prediction of Traffic Rate Interval Router Using Hybrid Training of Dynamic Synapse Neural Network Structure

[1]M. Shakiba, [1]M. Teshnehlab, [1]S. Zokaie and [2,3]M. Zakermoshfegh
[1]Department of Electrical Engineering, KN Toosi University of Technology, Tehran, Iran
[2]Department of Civil Engineering, Tarbiat Modarres University, Tehran, Iran
[3]Department of Civil Engineering, Jundi-Shapur University of Technology, Dezful, Iran

Abstract: In this study, a hybrid learning algorithm for training the Dynamic Synapse Neural Network (DSNN) to high accurate prediction of congestion in TCP computer networks is introduced. The idea behind this technique is to inform the TCP transmitters of congestion before it happens and to make transmitters decrease their data sending rate to a level which does not overflow the routers buffer. Traffic rate data are available in the format of time series and these real data are used to train and predict the future traffic rate condition. Hybrid learning algorithm aims to solve main problems of the Gradient Descent (GD) based method for the optimization of the DSNN, which are instability, local minima and the problem of generalization of trained network to the test data. In this method, Adaptable Weighted Particle Swarm Optimization (AWPSO) as a global optimizer is used to optimize the parameters of synaptic plasticity and the GD algorithm is used to optimize the weighted parameters of DSNN. As AWPSO is a derivative free optimization technique, a simpler method for the train of DSNN is achieved. Also the results are compared to GD algorithm.

Key words: Synaptic plasticity, prediction of time series, gradient descent, AWPSO learning algorithm, hybrid learning algorithm

INTRODUCTION

Due to traffic problems in router between Iran Telecommunication Research Center (ITRC) and Data Company, the aim of this study is prediction of interval traffic rate of that router (Box and Jenkins, 1976; Haltiner and Williams, 1980; Tanenbaum, 1996). One of the reasons of high rate of data loss in computer networks is the inability of the network to predict the congestion and inform the transmitters on time. The main objective of the study is introducing a powerful learning algorithm for training dynamic synapse neural network to high accurate prediction of chaotic time series. We monitored the traffic rate interval router in sampling of every thirty five minuets in December, January, February and March 2007 in kilo byte per second unit (Fig. 1). This time series has 5000 samples and we used 3000 samples for training and 2000 samples for testing neural network. We considered two learning algorithms for DSNN. These learning algorithms include the GD learning algorithm which uses GD for the training of both synaptic plasticity and weight matrix parameters and the hybrid learning algorithm which benefits both the PSO global search ability and the computational power of the GD. Finally some simulation results are provided for a comparison between these two methods.

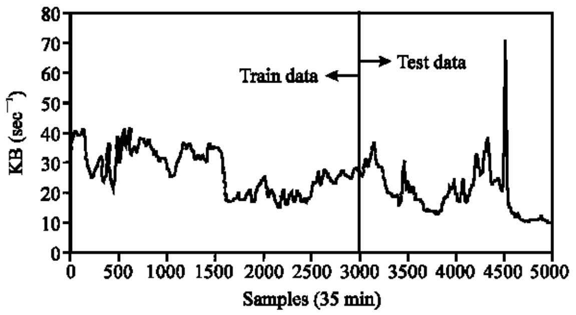

Fig. 1: The real data for traffic rate during Dec., Jan., Feb. and Mar. 2007, with interval sample every 35 min

MATERIALS AND METHODS

The structure of dynamic synapse neural network: Synaptic strength depends on three quantities: pre-synaptic activity $x_{ij}(t)$; a use-dependent term $P_{ij}(t)$ which may loosely related to pre-synaptic release probability and postsynaptic efficacy W_{ij} (Natschlager *et al*., 2001; Kim *et al*., 2003). Synaptic dynamics arise from the dependence of pre-synaptic release probability on the history of pre-synaptic activity. Specifically, the effect of activity $x_{ij}(t)$ in the j-th pre-synaptic unit on the i-th postsynaptic unit is given by the product of the synaptic

Corresponding Author: M. Shakiba, Department of Electrical Engineering, KN Toosi University of Technology, P.O. Box 14115/146, Tehran, Iran Tel: +98 21 77000513 Fax: +98 21 77000910

coupling between the two units and the instantaneous pre-synaptic activity as: $x_{ij}(t).P_{ij}(t).W_{ij}(t)$. The pre-synaptic activity $x_{ij}(t)$ is a continuous value (constrained to fall in the range [0, 1]). The coupling is in turn the product of a history-dependent release probability $P_{ij}(t)$ and a static scale factor $W_{ij}(t)$ corresponding to the postsynaptic response or potency at the synapse connecting j and i. Note that $W_{ij} \geq 0$ for excitatory synapses and $W_{ij} \leq 0$ for inhibitory synapses. The history-dependent component $P_{ij}(t)$ is constrained to fall in the range (0, 1). This component in turn depends on two auxiliary history-dependent functions $f_{ij}(t)$ and $d_{ij}(t)$. The quantity $d_{ij}(t)$ can be interpreted as the number of releasable synaptic vesicles; it decreases with activity and thereby instantiates a form of use-dependent depression. The quantity $f_{ij}(t)$ represents the propensity of each vesicle to be released; like (Ca^{2+}) in the pre-synaptic terminal, it increases with pre-synaptic activity $x_j(t)$ and thereby instantiates a form of facilitation. $f_{ij}(t)$ models facilitation (with time constant $F_{ij}>0$ and the initial release probability $U_{ij} \in [0, 1]$), whereas $d_{ij}(t)$ models the combined effects of synaptic depression (with time constant $D_{ij}>0$) and facilitation. Hence, the dynamics of a synaptic connection is characterized by the three parameters $U_{ij} \in [0, 1]$, $D_{ij}>0$, $F_{ij}>0$. For the numerical results presented in this paper we consider a discrete time (t = 1,2,...,T) version of the model defined by Eq. 1. In this setting we consider the dynamics with the initial conditions $\bar{f}_{ij}(1) = 0$ (i.e., $f_{ij}(1) = U_{ij}$) and $d_{ij}(1) = 1$. Note that in this case the time constants F_{ij} and D_{ij} have to be ≥ 1 (Mehrtash *et al.*, 2003; Kasthuri and Lichtman, 2004).

$$P_{ij}(t) = f_{ij}(t).d_{ij}(t) \tag{1}$$

$$\bar{f}_{ij}(t+1) = \bar{f}_{ij}(t) - \frac{\bar{f}_{ij}(t)}{F_{ij}} + U_{ij}.(1-\bar{f}_{ij}(t)).x_j(t) \tag{2}$$

$$d_{ij}(t+1) = d_{ij}(t) + \frac{1-d_{ij}(t)}{D_{ij}} - P_{ij}(t).x_j(t) \tag{3}$$

$$f_{ij}(t) = \bar{f}_{ij}(t).(1-U_{ij}) + U_{ij} \tag{4}$$

The input-output behavior of this model synapse depends on the four synaptic parameters U_{ij}, F_{ij}, D_{ij} and W_{ij}, as described, the same input yields markedly different outputs for different values of these parameters.

The dynamic synapses we have described (Fig. 2) are ideally suited to process signals with temporal structure.

Based on Fig. 2 we can show:

$$X = \left[x^1 x^2 ... x^{n_0}\right]^T \tag{5}$$

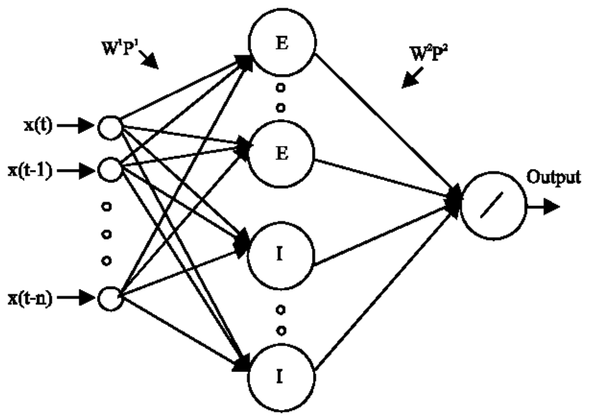

Fig. 2: Network architecture. The activation function at the hidden units was log-sigmoid function and linear function at the output unit

$$W^1 = \begin{bmatrix} w^{11} & w^{12} & \cdots & w^{1n_1} \\ w^{21} & w^{22} & \cdots & w^{2n_1} \\ w^{31} & w^{32} & \cdots & w^{3n_1} \\ \vdots & \vdots & \vdots & \vdots \\ w^{n_0 1} & w^{n_0 2} & \cdots & w^{n_0 n_1} \end{bmatrix}, P^1 = \begin{bmatrix} p^{11} & p^{12} & \cdots & p^{1n_1} \\ p^{21} & p^{22} & \cdots & p^{2n_1} \\ p^{31} & p^{32} & \cdots & p^{3n_1} \\ \vdots & \vdots & \vdots & \vdots \\ p^{n_0 1} & p^{n_0 2} & \cdots & p^{n_0 n_1} \end{bmatrix}, \tag{6}$$

$$B^1 = \left[b^1 \quad b^2 \quad \cdots \quad b^{n_1}\right]^T$$

$$Net^1 = (W^{1T}.*P^1).X + B^1 \tag{7}$$

$$O_e^1 = \log sig(Net^1(1:n_e)) \quad , \quad O_i^1 = \log sig(Net^1(1:n_i)) \tag{8}$$

$$O^1 = [o_e^1 o_e^2 \ldots o_e^{n_e} o_i^1 o_i^2 \ldots o_i^{n_i}] \tag{9}$$

$$W^2 = \begin{bmatrix} w^{11} & w^{12} & \cdots & w^{1n_1} \\ w^{21} & w^{22} & \cdots & w^{2n_1} \\ w^{31} & w^{32} & \cdots & w^{3n_1} \\ \vdots & \vdots & \vdots & \vdots \\ w^{n_1 1} & w^{n_1 2} & \cdots & w^{n_1 n_2} \end{bmatrix}, P^2 = \begin{bmatrix} p^{11} & p^{12} & \cdots & p^{1n_1} \\ p^{21} & p^{22} & \cdots & p^{2n_1} \\ p^{31} & p^{32} & \cdots & p^{3n_1} \\ \vdots & \vdots & \vdots & \vdots \\ p^{n_1 1} & p^{n_1 2} & \cdots & p^{n_1 n_2} \end{bmatrix}, \tag{10}$$

$$B^2 = \left[b^1 \; b^2 \; \ldots \; b^{n_2}\right]^T$$

$$Net^2 = \sum_{k \in E} W_k^2.P_k^2.O_K^2 + \sum_{k \in I} W_k^2.P_k^2.O_k^2 \tag{11}$$

$$E: Exhibitory \quad , \quad I: Inhibitory$$

$$Output = Net^2 \tag{12}$$

Particle swarm optimization: Particle Swarm Optimization (PSO) was originally designed and introduced by Eberhart and Kennedy (1995) and Kennedy and Eberhart (1995, 2001). The PSO is a population based search algorithm based on the simulation of the social behavior of birds, bees or a school of fishes. This algorithm originally intendeds to graphically simulate the graceful and unpredictable choreography of a bird folk. A vector in multidimensional search space represents each individual within the swarm. This vector has also one assigned vector, which determines the next movement of the particle and is called the velocity vector. The PSO algorithm also determines how to update the velocity of a particle. Each particle updates its velocity based on current velocity and the best position it has explored so far and based on the global best position explored by swarm (Engelbrecht, 2005, 2002; Sadri and Suen, 2006). The PSO process then is iterated a fixed number of times or until a minimum error based on desired performance index is achieved. It has been shown that this simple model can deal with difficult optimization problems efficiently.

A detailed description of PSO algorithm is presented in (Eberhart and Kennedy, 1995; Kennedy and Eberhart, 1995, 2001). Here we will give a short description of the PSO algorithm proposed by Kennedy and Eberhart. Assume that our search space is d-dimensional and the i-th particle of the swarm can be represented by a d-dimensional position vector $X_i = (x^1_i, x^2_i,...,x^d_i)$. The velocity of the particle is denoted by $V_i = (v^1_i, v^2_i,...,v^d_i)$. Also consider best visited position for the particle is $P_{ibest} = (p^1_i, p^2_i,...,p^d_i)$ and also the best position explored so far is $P_{gbest} = (p^1_g, p^2_g,...,p^d_g)$. So the position of the particle and its velocity is being updated using following equations:

$$v_i(t+1) = w.v_i(t) + c_1\varphi_1(p_i(t) - x_i(t)) + c_2\varphi_2(p_g(t) - x_i(t)) \quad (13)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (14)$$

where, c_1 and c_2 are positive constants and φ_1 and φ_2 are two uniformly distributed number between 0 and 1. In this equation, W is the inertia weight, which shows the effect of previous velocity vector on the new vector. The inertia weight W plays the role of balancing the global and local searches and its value may vary during the optimization process. A large inertia weight encourages a global search, while a small value pursues a local search. In (Mahfouf *et al.*, 2004) authors have proposed an Adaptive Weighted PSO (AWPSO) algorithm in which the velocity formula of PSO is modified as follows:

$$v_i(t+1) = w.v_i(t) + \alpha.[r_1(p_i - x_i(t)) + r_2(p_g - x_i(t))] \quad (15)$$

The second term in Eq. 15 can be viewed as an acceleration term, which depends on the distances between the current position x_i the personal best p_i and the global best p_g. The acceleration factor α is defined as follows:

$$\alpha = \alpha_0 + t/N_t \quad (16)$$

where, N_t denotes the number of iterations, t represents the current generation and the suggested range for α is (0.5, 1). As can be seen from Eq. 16, the acceleration term will increase as the number of iterations increases, which will enhance the global search ability at the end of run and help the algorithm to jump out of the local optimum, especially in the case of multi-modal problems. Furthermore, instead of using a linearly decreasing inertia weight, they used a random number, which was proved by Zhang and Hu (2003) to improve the performance of the PSO in some benchmark functions as follows:

$$w = w_0 + r\,(1 - w_0) \quad (17)$$

where, is $w_0 \in (0, 1)$ a positive constant and r is a random number uniformly distributed in (0, 1). The suggested range for w_0 is (0, 0.5), which makes the weight w randomly varying between 0 and 1. An upper bound is placed on the velocity in all dimensions. This limitation prevents the particle from moving too rapidly from one region in search space to another. This value is usually initialized as a function of the range of the problem. For example if the range of all x_{ij} is (-1, 1) then V_{max} is proportional to 1.

p_{ibest} For each particle is updated in each iteration when a better position for the particle or for the whole swarm is obtained. The feature that drives PSO is social interaction. Individuals (particles) within the swarm learn from each other and based on the knowledge obtained then move to become similar to their better previously obtained position and also to their better neighbors. Individual within a neighborhood communicate with one other. Based on the communication of a particle within the swarm different neighborhood topologies are defined. One of these topologies which is considered here, is the star topology. In this topology each particle can communicate with every other individual, forming a fully connected social network. In this case each particle is attracted toward the best particle (best problem solution) found by any member of the entire swarm. Each particle therefore imitates the overall best particle. So, the p_{gbest} is updated when a new best position within the whole swarm is found.

Learning algorithms for dynamic synapse neural network model: Here, two learning algorithms for the training of DS model are discussed.

GD based training of DSNN model: A minimizing of the sum of the square errors is used for training as the supervise rule:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} (y(n) - d(n))^2} \qquad (18)$$

Here, we describe the basics of the conjugate gradient learning algorithm- a generalized version of simple gradient descent-which we used to minimize the MSE between the network Output y(n) in response to the input time series x(t) and the target output is d(n). In order to apply a conjugate gradient algorithm ones has to calculate the partial derivatives $\frac{\partial E[y,d]}{\partial U_{ij}}, \frac{\partial E[y,d]}{\partial F_{ij}}, \frac{\partial E[y,d]}{\partial D_{ij}}$ and $\frac{\partial E[y,d]}{\partial W_{ij}}$ for all synapses < ij > in the network.

We state the equations for all of the partial derivatives for the network architecture shown in Fig. 2 (i.e., four dimensional network input and one-dimensional output and a single synapse between a pair of neurons). The calculations and determining the updates of W_{ij}, F_{ij}, U_{ij} and D_{ij} are started from outer layer. For calculating the partial derivatives $\frac{\partial E[y,d]}{\partial U_{ij}}$ we can show:

$$\frac{\partial E}{\partial U^2} = \frac{\partial E}{\partial e}\frac{\partial e}{\partial y}\frac{\partial y}{\partial net^2}\frac{\partial net^2}{\partial P^2}\frac{\partial P^2}{\partial U^2} = -e(t)\, W^2 O^1(t)\frac{\partial P^2}{\partial U^2} \qquad (19)$$

$$\frac{\partial P^2}{\partial U^2} = d^2(t)\frac{\partial f^2(t)}{\partial U^2} + f^2(t)\frac{\partial d^2(t)}{\partial U^2} \qquad (20)$$

$$\frac{\partial f^2}{\partial U^2} = -\bar{f}^2(t) + 1 + (1 - U^2)\frac{\partial \bar{f}^2}{\partial U^2} \qquad (21)$$

$$\frac{\partial \bar{f}^2(t)}{\partial U^2} = (1 - \frac{1}{F^2}).\frac{\partial \bar{f}^2(t-1)}{\partial U^2} + O^1(t-1).(1 - \bar{f}^2(t-1) - U^2.\frac{\partial \bar{f}^2(t-1)}{\partial U^2}) \qquad (22)$$

$$\frac{\partial d^2(t)}{\partial U^2} = (1 - \frac{1}{D^2}).\frac{\partial d^2(t-1)}{\partial U^2} - O^1(t-1).(d^2(t-1).\frac{\partial f^2(t-1)}{\partial U^2} + f^2(t-1).\frac{\partial d^2(t-1)}{\partial U^2}) \qquad (23)$$

Similar calculations are needed for the optimization of other synaptic plasticity parameters. But the optimization rule for the weighted are quite simple and as follows:

$$\frac{\partial E}{\partial W^2} = \frac{\partial E}{\partial e}\frac{\partial e}{\partial y}\frac{\partial y}{\partial net^2}\frac{\partial net^2}{\partial W^2} = -eP^2O^1 \qquad (24)$$

To ensure that the parameters $0 \le U \le 1$, $D \ge 1$, $F \ge 1$ and $W \ge 1$ (indices skipped for clarity) stay within their allowed range we introduce the unbounded parameters $\tilde{U}, \tilde{D}, \tilde{F}, \tilde{W} \in R$ with the following relationships to the original parameters: $U = 1/1 + \exp(-\tilde{U})$, $D = 1 + \exp(\tilde{D})$ $F = 1 + \exp(\tilde{F})$, $W = \exp(\tilde{W})$. The conjugate gradient algorithm was then used to adjust these unbounded parameters which are allowed to have any value in R. The partial derivatives of E[y,d] with respect to the unbounded parameters can easily be obtained by applying the chain rule, e.g.,

$$\frac{\partial E[y,d]}{\partial \tilde{U}} = \frac{\partial E[y,d]}{\partial U}.\frac{\partial U}{\partial \tilde{U}} = \frac{\partial E[y,d]}{\partial U}.\frac{\exp(\tilde{U})}{(1 + \exp(-\tilde{U}))^2} \qquad (25)$$

As it was shown before, the learning rules for the parameters of the antecedent part are complex and cannot be easily calculated. Also, as it was mentioned before GD learning algorithm, suffers from some shortcomings that are mainly local minima problem, selection of the learning rate, stability problems and insensitivity to long-term dependencies.

Hybrid training of DSNN: The hybrid-learning algorithm proposed here uses Adaptable Weighted Particle Swarm Optimization (AWPSO) based method to train the parameters of synaptic plasticity (U, F, D) and GD based methods for tuning the weights. These methods provide derivative free exploration for solution in input space. In addition using hybrid algorithms less local minimum solutions may be obtained.

The components of each particle in PSO population are (U_{ij}, F_{ij}, D_{ij}) parameters. The update rules of each population are as Eq. 14-17, which is the update rule for AWPSO. In this algorithm in each step the PSO will update the U_{ij}, F_{ij}, D_{ij} parameters between all layers and then the GD optimizer will run once to update the weights parameters using train data and the update rule for the weights parameters. After update of the parameters of both W_{ij} and U_{ij}, F_{ij}, D_{ij} one epoch has completely performed and mean squared error of the train data is calculated. This value would be the cost function of each particle which must be minimized. For a better exploration of the search space a mutation operator is defined. The previous vector of velocity in this way is reset to a random vector if for some iteration the global best value doesn't change.

RESULTS AND DISCUSSION

The simulations are presented here to compare the hybrid learning algorithm and the gradient descent learning algorithm. The simulations include time series prediction and identification of interval traffic rate router.

One hundred epochs for each algorithm was chosen and the best epoch for them was chosen within 500 epoch of run of algorithms. The best epoch chosen here, is the epoch after which the error for test data becomes larger. In addition to have a better comparison both algorithms run for 10 times.

Table 1: Comparing methods for prediction of interval traffic rate router

Traffic rate prediction		Dynamic synapse neural network: GD learning parameters: W, U, F, D	W, U	Hybrid training
NMSE (%)	Training	60.05090	51.06230	8.44880
	Testing	77.68730	71.96350	11.69580
Fitness (%)	Training	39.94910	48.93770	91.55120
	Testing	22.31270	28.03650	88.30420
RMSE	Training	0.16234	0.13804	0.94775
	Testing	0.10387	0.09622	0.03273
MAE	Training	0.11403	0.09883	0.69888
	Testing	0.09139	0.08380	0.01503
MBE	Training	0.00051	-0.01823	-0.34830
	Testing	-0.07710	-0.06940	-0.00080

As it can be seen the hybrid algorithm returns much better results in terms of generalization in testing and training (Table 1). In addition the results obtained by hybrid training algorithm are very near together. This result shows that independent of the initial population the results of hybrid algorithm will always converge to almost same point and the results are much more reliable than the GD algorithm. Also as it is shown the results obtained by hybrid learning algorithm proposed here are better.

The results of using GD learning algorithm: In Fig. 3, we use four-dimensional network input, one-dimensional network output and two hidden layers with towel neurons in first layer and six neurons in second layer. With this structure the network has 504 learning parameters. The standard deviation of Fitness of GD learning algorithm for test data is 22.3127%.

We compared network performance when different parameter subsets were optimized using the conjugate gradient algorithm, while the other parameters were held fixed. In all experiments, the fixed parameters were chosen to ensure heterogeneity in pre-synaptic dynamics. To achieve better performance one has to change at least two different types of parameters such as {W, U} or {W, D} (all other pairs yield worse performance) (Fig. 4). The standard deviation of Fitness of GD learning algorithm for test data is 28.0365%. The results of changing {W, U} are as follows:

The results of using hybrid learning algorithm: To improve exploration ability of the PSO small value for maximum velocity is selected. This value in all of experiments is 0.2. Also the values of w_0 is 0.5 and the value of α_0 is 1. The population size of the PSO for all of the experiments is 20. In simulations we use two-dimensional network input, one-dimensional network output and one hidden layer with towel neurons. With this structure the network has 144 learning parameters. In Fig. 5 the standard deviation of Fitness of hybrid learning algorithm for test data is 88.3042%.

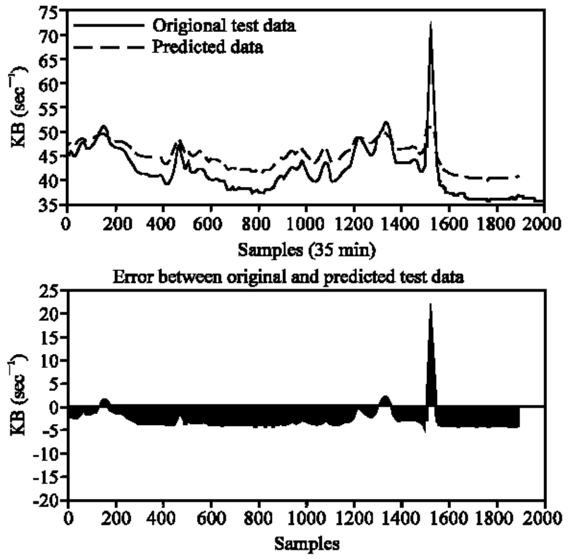

Fig. 3: One step ahead prediction of traffic rate time series with all of the learning parameters (W_{ij}, F_{ij}, U_{ij}, D_{ij})

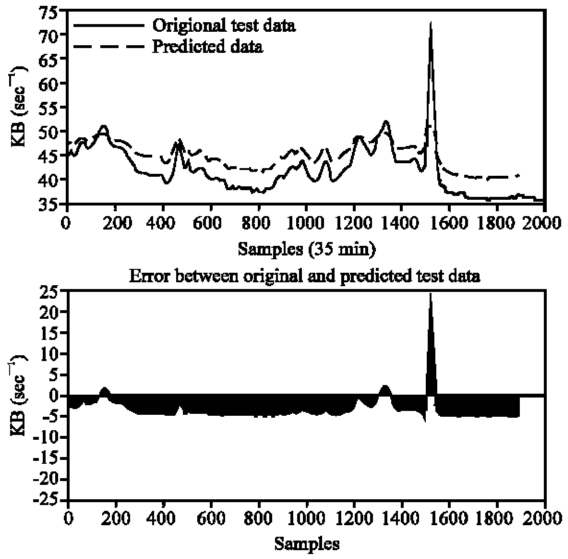

Fig. 4: One step ahead prediction of traffic rate time series with two learning parameters (W_{ij}, U_{ij})

Know the results are presented here to compare the hybrid learning algorithm and the gradient descent learning algorithm. For evaluating and comparing two different modeling and prediction methods which have been examined, five criteria are considered.

- Root Mean Squared Errors (RMSE)
- Normalized Mean Squared Errors (NMSE)
- Mean Absolute Error (MAE)
- Mean Bias Error (MBE)
- Fitness (100-NMSE)

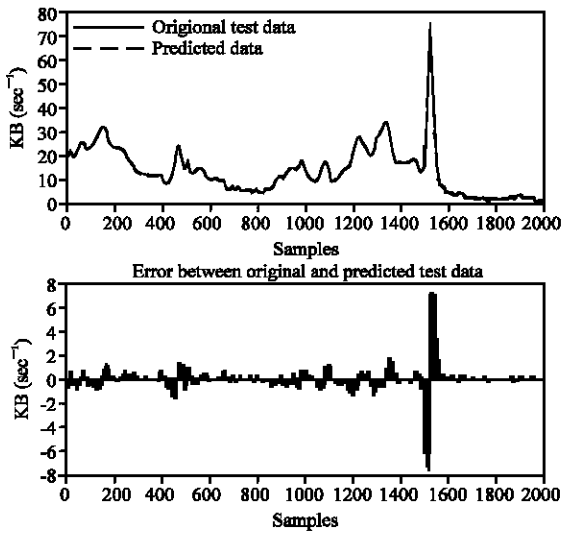

Fig. 5: One step ahead prediction of traffic rate by DSNN (two dimensional network input, one-dimensional network output and one hidden layer with towel neurons)

The criteria are defined as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{x}(i)-x(i))^2} \tag{26}$$

where, x is the measured and $\hat{x}$ is the Predicted value.

$$\mathrm{NMSE} = \frac{\sum_{i=1}^{N}(\hat{x}(i)-x(i))^2}{\sum_{i=1}^{N}(x(i)-\bar{x}(i))^2} \tag{27}$$

where, the value of x is the mean of the measured data.

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{x}(i)-x(i)| \tag{28}$$

$$\mathrm{MBE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{x}(i)-x(i)) \tag{29}$$

CONCLUSION

In this study, a hybrid learning algorithm for DSNN model has been introduced. The hybrid method introduced, uses AWPSO for the training of the parameters of synapses and the GD algorithm to optimize the weighted parameters of the DSNN. This method was compared with GD learning algorithm which uses GD algorithm to optimize both the parameters of synapses and weights. The GD algorithm suffers from some shortcomings which are mainly stability problems, complexity of learning algorithm, sensitivity to initial conditions of the values, local minima problems and the problem of overtraining to the train data. As it was shown before using the hybrid algorithm different initial conditions for parameters will converge to almost the same point and less local minima problem may occur. Also generalization results were quite better and less stability problems may happen. The simulation results support these complain. In addition, by using hybrid learning algorithm, the learning rules of the parameters simple.

ACKNOWLEDGMENTS

The authors would like to express their gratitude to Iran Telecommunication Research Center for their support and consent.

REFERENCES

Box, G.E.P. and G.M. Jenkins, 1976. Time Series Analysis Forecasting and Control. San Francisco. Holden-Day.

Eberhart, R. and J. Kennedy, 1995. A new optimizer using particles swarm theory. In: Proceeding 6th International Symposium on Micro Machine and Human Science, NJ., pp: 39-43.

Engelbrecht, A.P., 2002. Computational Intelligence. John Wiley and Sons.

Engelbrecht, A.P., 2005. Fundamentals of Computational Swarm Intelligence. Wiley.

Haltiner, G.J. and R.T. Williams, 1980. Numerical Prediction and Dynamic Metrology. John Wiley and Sons.

Kasthuri, N. and J.W. Lichtman, 2004. Structural dynamics of synapses in living animals. Curr. Opin. Neurobiol., 14: 105-111.

Kennedy, J. and R. Eberhart, 1995. Particle swarm optimization. IEEE International Conference on Neural Networks, pp: 1942-1948.

Kennedy, J. and R. Eberhart, 2001. Swarm Intelligence. Morgan Kaufmann Publishers Inc., San Francisco, CA.

Kim, J.H., H. Udo, H.L. Li, T.Y. Youn, M. Chen, E.R. Kandel and C.H. Bailey, 2003. Pre-synaptic activation of silent synapses and growth of new synapses contribute to intermediate and long-term facilitation in Aplysia. Neuron, 40: 151-165.

Mahfouf, M., M.Y. Chen and D.A. Linkens, 2004. Adaptive weighted particle swarm optimization for multi-objective optimal design of alloy steels. Lecture Notes Comput. Sci. Eng., 3242: 762-771.

Mehrtash, N., D. Jung and H. Klar, 2003. Image preprocessing with dynamic synapses. Neural. Comput. Appl., 12: 33-41.

Natschlager, T., W. Maass and A. Zador, 2001. Efficient temporal processing with biologically realistic dynamic synapses. Network: Comput. Neural. Syst., 12: 75-87.

Sadri, J. and Y.C. Suen, 2006. A genetic binary particle swarm optimization model. IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada.

Tanenbaum, A.S., 1996. Computer Networks. 3rd Edn. Prentice Hall.

Zhang, L. and S. Hu, 2003. A new approach to improve particle swarm optimization. Lecture Notes Comput. Sci. Eng., 2723: 134-139.