Constructive Neural Networks with Regularization

JANI LAHNAJÄRVI, MIKKO LEHTOKANGAS, AND JUKKA SAARINEN
Digital and Computer Systems Laboratory
Tampere University of Technology
P.O.BOX 553, FIN-33101 Tampere
FINLAND

Abstract: In this paper we present a regularization approach to the training of all the network weights in cascade-correlation type constructive neural networks. Especially, the case of regularizing the output neuron of the network is presented. In this case, the output weights are trained by employing a regularized objective function containing a penalty term which is proportional to the weight values of the unit being trained. It is shown that the training can still be done with the pseudo-inverse method of linear regression if the output unit employs linear activation function. The degree of regularization and the smoothness of network mapping can be adjusted by changing the value of the regularization parameter. The investigated algorithms were Cascade-Correlation, Modified Cascade-Correlation, Cascade, and Fixed Cascade Error. The algorithms having the regularization of the hidden and output units were compared with the ones having only the regularization of the hidden units and with those having no regularization at all. The simulation results show that the regularization of the output unit is highly beneficial. It leads to better generalization performance and in many cases to lower computational costs when compared to the partially and non-regulated versions of the same algorithms.

Key-Words: Constructive neural networks, regularization, generalization, cascade-correlation, classification, regression.

1 Introduction

One of the central issues in the application of neural networks is the determination of the appropriate level of complexity for the networks. The complexity determines the generalization properties of the model, since a network which is either too simple or too complex will have poor generalization performance. This means that there is some optimum number of network coefficients for which the network will give the best representation of the function needed for solving the given problem [2].

There are basically three ways to control the network size and structure. Constructive algorithms use a minimal initial network and add weights, hidden neurons or even layers until the network has learnt the given problem at a desired accuracy. Pruning algorithms, on the other hand, use an excessively large initial network, and as the training proceeds the numbers of weights, hidden units or layers are decreased until the network can just represent the problem solution [5]. The third way to control the network structure is regularization [1], [2], in which the cost function of the network is modified to contain a term which favours smaller network parameter values. When the unimportant network parameters have been driven close to zero, their effect on the network output is decreased although they are not actually removed from the network.

Although the constructive approach has been shown to have a number of advantages [5] over the pruning approach, there still is one major disadvantage in the constructive algorithms. We do not exactly know when to stop adding new hidden units into the network and, thus, the resulting possibly overly large network may lead to over-fitting and inferior generalization performance. One way to tackle this problem is to combine the constructive algorithms with the regularization approach. Regularization can prevent the constructive algorithm from learning too complex (and possibly erroneous) network mappings by setting the excessive network weights close to zero, which then decreases the effective size of the network. This way the network that may be too large in its true size acts like a smaller network, the size of which is closer to the optimal size needed in the particular application. Thus the key idea behind the regularization is that it encourages smoother network mappings [2], which is especially beneficial in regression (function approximation) problems. The degree of smoothness can easily be adjusted by regu-

larization parameters that can be varied according to the algorithm and application at hand.

This paper is focused on applying a regularization approach in the hidden and output unit training of constructive neural network algorithms. Especially a new feature of regularizing linear output units in some cascade-correlation type algorithms has been studied. The idea of the cascade-correlation algorithm is to add hidden units one by one, each on a separate hidden layer, to form a multilayer perceptron capable of solving a learning problem without prior assumptions about its size and structure.

2 Regularization in the Constructive Algorithms

We have studied four different constructive algorithms. The investigated algorithms were Cascade-Correlation (CC) [3], Modified Cascade-Correlation (MCC) with objective function $\sqrt{S_2}$ presented in [5], Cascade (CAS) [12], and Fixed Cascade Error (FCE) [6]. We studied three different versions of all these algorithms. In the first versions (*i.e.* non-regulated versions), regularization was not applied at all. In the second versions (*i.e.* partially regularized versions), we used regularization in the hidden unit training but not in the output unit training. Finally, in the third versions (*i.e.* totally regularized versions), we applied regularization both in the hidden unit and output unit training. The performance of all the versions of the algorithms was then assessed based on numerical simulations both with classification and regression problems.

2.1 Regularizing the Hidden Units

The first benchmark algorithm is *the standard Cascade-Correlation algorithm* designed by Fahlman and Lebiere [3]. The cascade-correlation learning begins with a minimal network and automatically adds new hidden units one by one until a satisfactory solution is achieved. Once a new hidden unit has been added to the network, its input weights are frozen. This unit then becomes a permanent feature detector in the network, and it produces outputs for possibly additional hidden units creating more complex feature detectors. The cascade-correlation architecture has been noticed to have several advantages over conventional non-constructive backpropagation algorithms [3].

The (partially and totally) regularized versions of the CC algorithm can be presented in the following way. First, q (q = 8 in our simulations with all the algorithms) candidate hidden units are created by initializing their weights with random numbers in predefined ranges (these are given in section 3 for the investigated algorithms). Next, all the q candidates are trained to their final values with RPROP algorithm [13] by employing the regularized version of the standard objective function in the update rule. For cascade-correlation, this function is given by

$$C_{C,j} = \left| \sum_l (V_{j,l} - \bar{V}_j)(E_l - \bar{E}) \right| - \nu \sum_i w_{ij}^2, j = 1, \ldots, q, \quad (1)$$

where $V_{j,l}$ is the output of the jth candidate unit for the lth training pattern, $\bar{V}_j$ the mean of the jth candidate unit outputs, E_l is the network output error (one output unit in all our simulations) for the lth training pattern, $\bar{E}$ is the mean of the network output errors, ν is the regularization parameter, and w_{ij} is the network weight that connects the input unit (or pre-existing hidden unit) i to the candidate unit j. It should be noticed that in the standard (non-regulated) version of CC algorithm, the objective function is otherwise the same but the last term (including the squared sum of the weights) in Equation 1 is missing. The same is true also with the other investigated algorithms.

Finally, after all the candidates have been trained and the values of the objective function have been recorded, the maximum magnitude of the regularized covariance $max(C_{c,j})$ among all the candidates is searched. The corresponding candidate unit j is selected to be the most promisingly trained hidden unit which is then installed in the active network. After this the network output unit is trained (with or without regularization, see section 2.2) and all the other candidate hidden units are deleted. The above procedure is repeated at each point when a new hidden unit is to be added to the network.

Modified Cascade-Correlation algorithm [5] is almost equivalent to CC. Only exceptions are due to the different objective function used in the hidden unit training. The averages of both the remaining network error and the candidate hidden unit output have been deleted when compared to the objective function of the cascade-correlation. Otherwise, the general structure of the algorithm has been maintained the same. The candidate hidden units are now trained by maximizing the regularized version of $\sqrt{S_2}$ [5], which is given by

$$\sqrt{S_2}_{,j} = \left| \sum_l V_{j,l} E_l \right| - \nu \sum_i w_{ij}^2, j = 1, \ldots, q. \quad (2)$$

All the other steps of this algorithm are kept the same as compared to CC. The standard multiple-candidate unit version of the algorithm is obtained just like in all the algorithms by deleting the last term of the objective function used in the hidden unit training.

There are two major changes in *Cascade algorithm* [12] when compared to the CC algorithm. Firstly, the regularized objective function employed in the candidate hidden unit training is not correlation (or covariance, actually) function but the squared error function, which is defined as

$$C_{E,j} = \sum_l (t_l - o_{j,l})^2 + \nu \sum_i w_{ij}^2, j = 1, \ldots, q, \quad (3)$$

where $o_{j,l}$ is the actual network output for lth training pattern with the jth candidate unit in the network and t_l is the target output of the network for the lth training pattern. This objective function means that we have to update also the output weight of the candidate unit after each epoch during the candidate unit training, which leads to higher computational complexity than in the other algorithms. Secondly, this objective function is minimized and not maximized as in the case of correlation (this also explains the use of a positive sign in front of the last term in Equation 3). Again all the other parts of the algorithm are similar to CC.

Fixed Cascade Error algorithm [6] is rather similar to the original cascade-correlation [3] and the modified CC that was presented by Kwok and Yeung [5]. The difference is that the objective function used in the regularized hidden unit training is now defined as

$$C_{FE,j} = \sum_l (E_l - \bar{E}) V_{j,l} - \nu \sum_i w_{ij}^2, j = 1, \ldots, q. \quad (4)$$

The best candidate unit after the training is yet again selected to be the one that has the maximal value of the above mentioned criterion $C_{FE,j}$ among all the q candidates. The other stages of the algorithm are kept similar to those of CC.

2.2 Regularizing the Output Units

The output unit regularization is carried out in a little bit different way when compared to the hidden unit regularization shown in the previous section. Now, the activation function of the unit being trained is a linear one, which enables us to solve the output unit weights analytically. This means that we do not have to use RPROP or any other iterative method in the training phase. The objective function of the output unit training in the totally regularized versions of the algorithms is the squared error function added with the regularization term [4], [9]

$$C_{out} = \sum_l E_l^2 + \mu \sum_i v_i^2 = \boldsymbol{e}\boldsymbol{e}^T + \mu \boldsymbol{v}\boldsymbol{v}^T, \quad (5)$$

where μ is the regularization parameter, $\boldsymbol{e}_{1\times n}$ is the output error vector, and $\boldsymbol{v}_{1\times(p+h+1)}$ is the weight vector (including bias) of the output unit. The number of the training patterns is n, the number of network inputs is p, and the number of the hidden units installed in the network is h. (At this point, we must notice that the *standard* objective function in the output unit training is the regularized objective function without the last term given in Eq. 5. That standard function is used in the partially and non-regulated versions of the algorithms.) Eq. 5 can be converted into

$$C_{out} = (\boldsymbol{o} - \boldsymbol{t})(\boldsymbol{o} - \boldsymbol{t})^T + \boldsymbol{v}\boldsymbol{U}\boldsymbol{v}^T = (\boldsymbol{v}\boldsymbol{R} - \boldsymbol{t})(\boldsymbol{v}\boldsymbol{R} - \boldsymbol{t})^T + \boldsymbol{v}\boldsymbol{U}\boldsymbol{v}^T \quad (6)$$

where $\boldsymbol{o}_{1\times n}$ is the actual output vector and $\boldsymbol{t}_{1\times n}$ is the target output vector of the network. Moreover, $\boldsymbol{R}_{(p+h+1)\times n}$ is the input matrix of the output unit and $\boldsymbol{U}_{(p+h+1)\times(p+h+1)}$ is a diagonal regularization matrix in which all the diagonal elements are μ's. Now, the optimum value for $\boldsymbol{v}$ which minimizes the objective function can be found by setting the gradient of the objective function to zero

$$\nabla C_{out} = \partial C_{out} / \partial \boldsymbol{v} \equiv 0. \quad (7)$$

Eq. 7 can easily be solved in the following way. First,

$$\partial C_{out} / \partial \boldsymbol{v} = \partial(\boldsymbol{v}\boldsymbol{R}\boldsymbol{R}^T\boldsymbol{v}^T - 2\boldsymbol{v}\boldsymbol{R}\boldsymbol{t}^T + \boldsymbol{t}\boldsymbol{t}^T + \boldsymbol{v}\boldsymbol{U}\boldsymbol{v}^T) / \partial \boldsymbol{v} \equiv 0. \quad (8)$$

This can be computed further as

$$\partial C_{out} / \partial \boldsymbol{v} = 2\boldsymbol{R}\boldsymbol{R}^T\boldsymbol{v}^T - 2\boldsymbol{R}\boldsymbol{t}^T + 2\boldsymbol{U}\boldsymbol{v}^T \equiv 0, \quad (9)$$

which then solves as

$$\boldsymbol{v}^T = (\boldsymbol{R}\boldsymbol{R}^T + \boldsymbol{U})^{-1}\boldsymbol{R}\boldsymbol{t}^T. \quad (10)$$

When compared to the solution of the standard case, which is defined as

$$\boldsymbol{v}^T = (\boldsymbol{R}\boldsymbol{R}^T)^{-1}\boldsymbol{R}\boldsymbol{t}^T, \quad (11)$$

we notice that only the regularization matrix $\boldsymbol{U}$ has to be added inside the inverse part in the final solution of the non-regulated case to obtain the final solution of the regulated case. By adding the matrix $\boldsymbol{U}$ into the solution we can even avoid some numerical difficul-

ties that are rather often met in inverting matrices (this adding method is called 'ridge regression' in mathematical literature). Also the easiness of the needed change in the equation encourages the use of regularization in the output unit training, since with only a tiny increase in the computational complexity we can expect a clear enhancement in the generalization performance of the neural network at hand.

3 Simulations and Results

The algorithms were tested in extensive simulations with four classification and four regression problems. The classification problems were Chess [8], Spiral [3], 10-bit Parity [3], and Cancer diagnosis problem [11]. The regression problems were Henon map [8], Laser time series [14], Additive function approximation [5], and Mackey-Glass time series [10]. The Cancer diagnosis problem and Laser time series are based on real-world data while the others are artificially generated. All the simulations were repeated twenty times due to the random initialization of the network weights. The candidate hidden unit weights were initialized with uniformly distributed random numbers of the range [-0.5, +0.5] in all the versions of all the algorithms.

The regularization parameter ν was set to 0.01 for the totally and partially regularized versions of the CC, MCC, and FCE algorithms in all the problems. For the totally and partially regularized versions of the CAS algorithm we used a parameter ν value of 10^{-5} in Chess, Cancer, Laser, and Additive problems while in Spiral, Parity, Henon, and Mackey problems we utilized a parameter ν value of 5×10^{-6}. The regularization parameter μ was set to 0.0001 for the totally regularized versions of all the algorithms. Setting the values of the regularization parameters was not highly critical to the performance of the algorithms. Only in CAS algorithm the parameter ν had to be set more carefully according to each simulation problem. It is also important to notice that the regularization parameter ν should not be too large in any of the algorithms since that would lead to some computational difficulties first in the hidden and then in the output unit training phases. On the other hand, if both of the regularization parameters are set to very low (positive) values, the regularized algorithms behave just like their standard versions. In our simulations, we used values that were noticed to perform relatively well with all the simulation problems we studied.

Since all the candidate hidden units had sigmoidal activation function (hyperbolic tangent), they were trained with RPROP algorithm [13]. The hidden unit training was continued until the changes in the objective function value were sufficiently small or the maximum number of the epochs was reached. The output units were trained by the pseudo-inverse method of linear regression as discussed in section 2.2. The network training was stopped when the network output error fell below the target error value or the maximum number of the hidden units was reached. The error values that we used for computing the simulation results were classification error (*CERR*) for the classification problems and normalized mean square error (*NMSE*) for the regression problems [7].

The final results are shown in Table 1. In each case, the averaged best result of the testing data and the average numbers of hidden units and MFLOPS (millions of floating point operations) are shown. The uppermost row in all the cases was obtained with the totally regularized version of the particular algorithm, the row in the middle is from the partially regularized version, and the lowest row is from the standard non-regulated version of the algorithm. In Chess and Spiral problems the results are shown according to the training data since there were no testing data available in these problems. For the readers' convenience, the best result among all the versions of the algorithms is shown in 'bold' for each problem. In case of many equally best results in a single problem, all the best results in that particular problem are given in 'bold'.

The results in Table 1 show that the output unit regularization offers us some clear benefits. When we compare the results of the totally regularized versions to the partially regularized versions of the algorithms, we can see that the changes in the error values are in the range of -22% to +6% which means that the output unit regularization enhances the generalization performance of the networks in most cases. When we consider the number of the hidden units, the changes are respectively in the range of -18% to +18% showing that the output unit regularization does not decrease the actual size of the networks. The enhancement in the generalization performance occurs thus due to the decreased *effective* size of the networks as could be expected. When we study the respective change in the number of MFLOPS we notice it to be in the range of -27% to +25%. This is not surprising since the output unit regularization is such a simple operation that it should not increase the

computational load of the algorithms. Thus, the variations in the computational load come mainly from the small variations in the number of the hidden units, especially when we remember that the computational load of the algorithms is proportional to the square of the number of the hidden units in the networks with fully cascaded architecture.

When we examine the effects of the output unit regularization problem-by-problem, we see that it is extremely beneficial in Parity, Henon, and Additive problems. Yet again it is interesting to notice that the output unit regularization enhances results in Parity problem even more than the hidden unit regularization alone [7]. What makes this interesting is that we have included some input noise in the training data of that particular problem. Since it has been shown that this kind of an addition of input noise is equivalent to regularization [1], it means that the results with the standard versions of the algorithms are already regularized in some sense. Despite of that fact, both of the regularized versions give furthermore enhancement in the results. It is also rather delightful to notice that the results in Additive function approximation problem are considerably enhanced, since this problem is one of the most difficult ones investigated in this study. Furthermore, we can see that the output unit regularization does not only enhance the results of the classification problems but also the results of the regression problems, although the linear output unit regularization alone could easily be thought to be favourable mainly for the classification performance. This observation confirms the fact that (any kind of) regularization is favourable for solving regression problems.

Finally, when we compare the totally regularized versions of the algorithms to their partially regularized versions one-by-one, we notice that the output unit regularization works best with CC algorithm. The enhancements are observable also with the rest three algorithms although there is some decline in results in some cases. As a summary of the results with totally regularized versions we can say that the FCE algorithm is computationally the easiest one, the CC algorithm needs the smallest amount of hidden units, and the MCC algorithm provides us the best error values on average. The CAS algorithm is found to be computationally the most demanding one. Discussion about the differences in results between the partially regulated and standard versions of the algorithms can be found in [7].

4 Conclusions

We presented a regularization approach to be used in the output unit training of some cascade-correlation type constructive neural network algorithms. The key idea of the regularization is to use an objective function containing a penalty term which is proportional to the output unit weight values while training that particular unit. The use of the penalty term encourages smoother network mappings by setting the unimportant network weights to smaller values. It was shown that the training of a linear output unit can still be done with the pseudo-inverse method of linear regression by only adding a regularization matrix in the middle of the standard solution. Since there is no need for iterative methods, the achieved enhancement in generalization should increase the computational complexity of the algorithm only slightly. The simulations showed that the totally regularized versions produced better generalization performance than the standard or partially regularized versions of the algorithms. Furthermore, in some cases the totally regularized versions needed less hidden units than the partially regularized versions and the overall computational load of the totally regularized versions was kept the same when compared to the other two versions of the algorithms.

References:

[1] C. M. Bishop, “Training with Noise is Equivalent to Tikhonov Regularization”, *Neural Computation*, Vol. 7, No. 1, 1995, pp. 108-116.

[2] C. M. Bishop, “Regularization and Complexity Control in Feed-Forward Networks”, *Technical Report: NCRG/95/022*, Neural Computing Research Group, Dept. of Computer Science and Applied Mathematics, Aston University, Birmingham, UK, 1995.

[3] S. E. Fahlman and C. Lebiere, “The Cascade-Correlation Learning Architecture”, *Technical Report CMU-CS-90-100*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990.

[4] S. Haykin, *Neural Networks: A Comprehensive Foundation*, MacMillan College Publishing Company, New York, 1994.

[5] T.-Y. Kwok and D.-Y. Yeung, “Objective Functions for Training New Hidden Units in Constructive Neural Networks”, *IEEE Transactions on Neural Networks*, Vol. 8, No. 5, Sep. 1997, pp. 1131-1148.

[6] J. Lahnajärvi, M. Lehtokangas, and J. Saarinen, “Fixed Cascade Error - A Novel Constructive Neural Network for Structure Learning”, *Proceedings of the Artificial Neural Networks in Engineering Conference*, ANNIE’99, St. Louis, Missouri, USA, Nov. 7-10, 1999, pp. 25-30.

[7] J. Lahnajärvi, M. Lehtokangas, and J. Saarinen, "Constructive Neural Networks with Regularization Approach in the Hidden Unit Training", *International NAISO Congress on Information Science Innovations*, ISI'2001, Dubai, UAE, Mar. 17-21, 2001, accepted paper, 7 pages.

[8] M. Lehtokangas, J. Saarinen, P. Huuhtanen, and K. Kaski, "Initializing Weights of a Multilayer Perceptron Network by Using the Orthogonal Least Squares Algorithm", *Neural Computation*, Vol. 7, No. 5, 1995, pp. 982-999.

[9] M. Lehtokangas, "Pattern Recognition with Novel Support Vector Machine Learning Method", *Proceedings of the X European Signal Processing Conference*, EUSIPCO-2000, Tampere, Finland, Sep. 5-8, 2000, Vol. 2, pp. 733-736.

[10] Oregon Graduate Institute of Science and Technology, *Data Distribution WWW site* (http://www.ece.ogi.edu/~ericwan/data.html).

[11] L. Prechelt, "PROBEN1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules", *Technical Report 21/94*, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, Sep. 1994.

[12] L. Prechelt, "Investigation of the CasCor Family of Learning Algorithms", *Neural Networks*, Vol. 10, No. 5, 1997, pp. 885-896.

[13] M. Riedmiller and H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm", *Proceedings of the IEEE International Conference on Neural Networks*, San Francisco, CA, Mar. 28 - Apr. 1, 1993, pp. 586-591.

[14] *The Santa Fe Time Series Competition Data WWW site* (http://www.stern.nyu.edu/~aweigend/Time-Series/SantaFe.html).

Table 1. The results of the algorithms with all the simulation problems. The values are the error value (ERR), the number of the hidden units (HU), and MFLOPS (MF) spent during the training phase, respectively. Problems marked with an asterisk (*) give *CERR* and the others give *NMSE* as their error values. In each case, the uppermost values are for the totally regularized versions (TR), the middle ones are for the partially regularized versions (PR), and the lowest values are for the standard (non-regularized) versions (NR) of the algorithms. The results are average values of twenty runs.

Algorithm		**CC**			**MCC**			**CAS**			**FCE**		
Problem		ERR	HU	MF	ERR	HU	MF	ERR	HU	MF	ERR	HU	MF
	TR	**0**	3.45	0.93	**0**	3.25	0.80	**0**	3.25	1.84	**0**	**3.05**	**0.69**
Chess*	PR	**0**	3.65	0.99	**0**	3.35	0.84	**0**	3.20	1.77	**0**	**3.10**	**0.70**
	NR	**0**	3.25	1.30	**0**	3.20	1.14	**0**	3.55	2.01	**0**	**2.95**	**0.92**
	TR	**0**	19.15	139	**0**	19.10	132	**0**	**17.35**	249	**0**	18.60	**121**
Spiral*	PR	**0**	19.40	142	**0**	18.85	130	**0**	**17.35**	247	**0**	18.50	**121**
	NR	**0**	17.25	142	**0**	18.60	150	**0**	17.80	253	**0**	**16.95**	**127**
	TR	0.0144	**5.80**	**144**	**0.0124**	6.75	167	0.0221	8.40	419	0.0148	7.30	174
Parity*	PR	0.0187	**6.60**	167	0.0166	6.75	**162**	0.0247	8.95	468	**0.0141**	7.00	168
	NR	0.0200	6.70	177	0.0186	**5.65**	142	0.0199	9.65	506	**0.0166**	5.75	**140**
	TR	0.0232	**2.45**	27.6	0.0231	3.35	40.2	**0.0215**	2.70	67.1	0.0221	2.50	**26.4**
Cancer*	PR	0.0244	2.75	32.0	0.0238	3.10	34.9	**0.0202**	**2.45**	58.4	0.0222	2.85	**31.7**
	NR	0.0238	**2.55**	**34.4**	0.0234	3.10	41.4	**0.0229**	2.85	70.3	0.0231	2.80	36.3
	TR	0.0295	**8.05**	18.0	**0.0279**	8.40	17.9	0.0372	8.40	47.0	0.0281	8.20	**16.5**
Henon	PR	0.0313	8.05	17.9	**0.0306**	8.40	17.9	0.0414	8.50	47.9	0.0357	**7.95**	**15.8**
	NR	0.0418	**7.85**	23.3	0.0422	8.10	23.2	**0.0353**	8.45	49.8	0.0374	**7.85**	**21.2**
	TR	0.0256	15.85	765	**0.0248**	**15.45**	712	0.0271	15.60	1610	0.0253	15.65	**708**
Laser	PR	0.0258	15.35	721	**0.0227**	**14.90**	**670**	0.0262	15.45	1590	0.0246	15.80	718
	NR	0.0284	16.00	982	0.0281	**15.45**	**895**	**0.0276**	15.50	1680	**0.0276**	16.45	971
	TR	**0.0317**	17.50	219	0.0320	17.55	210	0.0443	17.20	482	0.0362	**16.55**	**188**
Additive	PR	0.0370	17.90	232	**0.0335**	17.65	216	0.0444	17.40	489	0.0372	**16.75**	**190**
	NR	0.0402	**15.85**	269	**0.0374**	16.20	**259**	0.0511	17.15	502	0.0384	17.05	271
Mackey-	TR	0.3270	**6.20**	74.9	0.3264	6.85	81.6	0.3301	7.95	208	**0.3237**	6.40	**68.8**
Glass	PR	0.3239	7.45	100	0.3236	7.20	**88.9**	0.3372	**6.75**	166	**0.3148**	7.80	94.2
	NR	0.3353	**6.45**	102	**0.3341**	6.70	102	0.3380	7.60	199	0.3347	6.55	**91.8**