ALEC: An Adaptive Learning Framework for Optimizing Artificial Neural Networks

Ajith Abraham and Baikunth Nath

School of Computing & Information Technology
Monash University (Gippsland Campus), Churchill 3842, Australia
{Email: Ajith.Abraham, Baikunth.Nath@infotech.monash.edu.au}

Abstract. In this paper we present ALEC (Adaptive Learning by Evolutionary Computation), an automatic computational framework for optimizing neural networks wherein the neural network architecture, activation function, weights and learning algorithms are adapted according to the problem. We explored the performance of ALEC and artificial neural networks for function approximation problems. To evaluate the comparative performance, we used three different well-known chaotic time series. We also report some experimentation results related to convergence speed and generalization performance of four different neural network-learning algorithms. Performances of the different learning algorithms were evaluated when the activation functions and architecture were changed. We further demonstrate how effective and inevitable is ALEC to design a neural network, which is smaller, faster and with a better generalization performance.

1. Introduction

In Artificial Neural Network (ANN) terminology, function approximation is simply to find a mapping $f: R^m \Rightarrow R^n$, given a set of training data. Even then, finding a global approximation (applying to the entire state space) is often a challenging task. The important drawback with the conventional design of ANN is that the designer has to specify the number of neurons, their distribution over several layers and interconnection between them. In this paper, we investigated the speed of convergence and generalization performance of backpropagation algorithm, conjugate gradient algorithm, quasi Newton algorithm and Levenberg-Marquardt algorithm. Our experiments show that architecture and node activation functions can significantly affect the speed of convergence of the different learning algorithms. We finally present the evolutionary search procedures wherein ANN design can evolve towards the optimal architecture without outside interference, thus eliminating the tedious trial and error work of manually finding an optimal network [1]. Experimentation results, discussions and conclusions are provided towards the end.

2. Artificial Neural Network Learning Algorithms

If we consider a network with differentiable activation functions, then the activation functions of the output units become differentiable functions of both the input variables and of the weights and biases. If we define an error function (E), such as

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2074, pp. 171–180, 2001.

sum of squares function, which is a differentiable function of the network outputs, then this error function is itself a differentiable function of the weights. We can therefore evaluate the derivatives of the error with respect to the weights, and these derivatives can then be used to find weight values, which minimize the error function, by using one of the following learning algorithms:

Backpropagation Algorithm (BP)

BP is a gradient descent technique to minimize the error E for a particular training pattern. For adjusting the weight (w_k), in the batched mode variant the descent is based on the gradient ∇E ($\frac{\delta E}{\delta w_k}$) for the total training set:

$$\Delta w_k(n) = -\varepsilon * \frac{\delta E}{\delta w_k} + \alpha * \Delta w_k(n-1) \tag{1}$$

The gradient gives the direction of error E. The parameters ε and α are the learning rate and momentum respectively. A good choice of both the parameters is required for training success and speed of the ANN.

Scaled Conjugate Gradient Algorithm (SCGA)

Moller [5] introduced the scaled conjugate gradient algorithm as a way of avoiding the complicated line search procedure of conventional conjugate gradient algorithm (CGA). According to the SCGA, the Hessian matrix is approximated by

$$E''(w_k)p_k = \frac{E'(w_k + \sigma_k p_k) - E'(w_k)}{\sigma_k} + \lambda_k p_k \tag{2}$$

where E' and E'' are the first and second derivative information of global error function $E(w_k)$. The other terms p_k, σ_k and λ_k represent the weights, search direction, parameter controlling the change in weight for second derivative approximation and parameter for regulating the indefiniteness of the Hessian. In order to get a good quadratic approximation of E, a mechanism to raise and lower λ_k is needed when the Hessian is positive definite. Detailed step-by-step description can be found in [5].

Quasi - Newton Algorithm (QNA)

Quasi-Newton method involves generating a sequence of matrices $G^{(k)}$ which represents increasingly accurate approximations to the inverse Hessian (H^{-1}). Using only the first derivative information of E [4], the updated expression is as follows:

$$G^{(k+1)} = G^{(k)} + \frac{pp^T}{p^T v} - \frac{(G^{(k)}v)\,v^T\,G^{(k)}}{v^T G^{(k)} v} + (v^T G^{(k)} v)uu^T \tag{3}$$

where $p = w^{(k+1)} - w^{(k)}$, $v = g^{(k+1)} - g^{(k)}$, $u = \frac{p}{p^T v} - \frac{G^{(k)}v}{v^T G^{(k)} v}$ and T

represents transpose of a matrix. A significant advantage of the QNA over the CGA is that the line search does not need to be performed with such great accuracy.

Levenberg-Marquardt (LM) Algorithm

When the performance function has the form of a sum of squares, then the Hessian matrix can be approximated to $H = J^T J$; and the gradient can be computed as $g = J^T e$, where J is the Jacobian matrix, which contains first derivatives of the network errors with respect to the weights, and e is a vector of network errors. The Jacobian matrix can be computed through a standard backpropagation technique that is less complex than computing the Hessian matrix. The LM algorithm uses this approximation to the Hessian matrix in the following Newton-like update:

$$x_{k+1} = x_k - [J^T J + \mu I]^{-1} J^T e \quad (4)$$

When the scalar μ is zero, this is just Newton's method, using the approximate Hessian matrix. When μ is large, this becomes gradient descent with a small step size. As Newton's method is more accurate, μ is decreased after each successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. By doing this, the performance function will always be reduced at each iteration of the algorithm [4].

3. Experimental Setup Using Artificial Neural Networks

We used a feedforward network with 1 hidden layer and the training was performed for 2500 epochs. The numbers of hidden neurons were varied (14,16,18,20,24) and the speed of convergence and generalization error for each of the four learning algorithms was observed. The effect of node activation functions, Log-Sigmoidal Activation Function (LSAF) and Tanh-Sigmoidal Activation Function (TSAF), keeping 24 hidden neurons for the four learning algorithms was also studied. Computational complexities of the different learning algorithms were also noted during each event. In our experiments, we used the following 3 different time series for training the ALEC/ANN and evaluating the performance. The experiments were replicated 3 times and the worst errors are reported.

a) Waste Water Flow Prediction

The problem is to predict the wastewater flow into a sewage plant [6]. The water flow was measured every hour. It is important to be able to predict the volume of flow *f(t+1)* as the collecting tank has a limited capacity and a sudden increase in flow will cause to overflow excess water. The water flow prediction is to assist an adaptive online controller. The data set is represented as [*f(t), f(t-1), a(t), b(t), f(t+1)*] where *f(t), f(t-1)* and *f(t+1)* are the water flows at time *t,t-1, and t+1* (hours) respectively. *a(t)* and *b(t)* are the moving averages for 12 hours and 24 hours. The time series consists of 475 data points. The first 240 data sets were used for training and remaining data for testing.

b) Mackey-Glass Chaotic Time Series

The Mackey-Glass differential equation is a chaotic time series for some values of the parameters *x(0)* and τ [9].

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1 + x^{10}(t-\tau)} - 0.1\,x(t). \quad (5)$$

We used the value *x(t-18), x(t-12), x(t-6), x(t)* to predict *x(t+6)*. Fourth order Runge-Kutta method was used to generate 1000 data series. The time step used in the method is 0.1 and initial condition were *x(0)*=1.2, τ=17, *x(t)*=0 for *t*<0. First 500 data sets were used for training and remaining data for testing.

c) Gas Furnace Time Series Data

This time series was used to predict the CO_2 (carbon dioxide) concentration *y(t+1)* [8]. In a gas furnace system, air and methane are combined to form a mixture of gases containing CO_2. Air fed into the gas furnace is kept constant, while the methane feed rate *u(t)* can be varied in any desired manner. After that, the resulting CO_2 concentration y(t) is measured in the exhaust gases at the outlet of the furnace. Data is represented as [*u(t), y(t), y(t+1)*] The time series consists of 292 pairs of observation and 50% of data was used for training and remaining for testing.

3.1 Experimentation Results Using ANNs

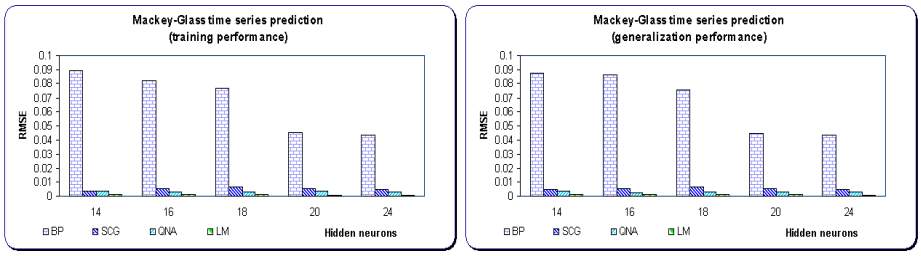


Fig. 1. Mackey-Glass time series (training and generalization performance)

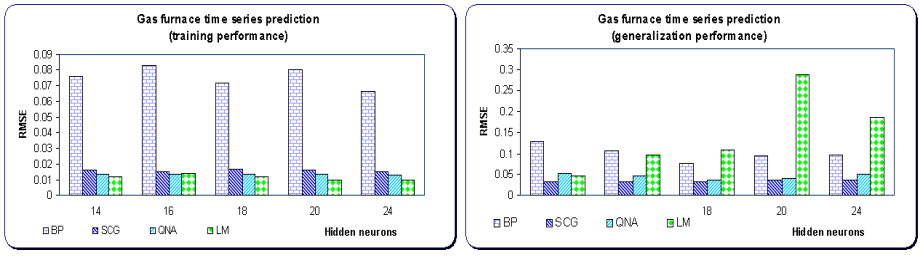


Fig. 2. Gas furnace time series (training and generalization performance)

Figures 1, 2 and 3 illustrate the training error and generalization performance of the 4 learning algorithms (BP, SCG, QNA and LM) for the three different time series. Figures 4 (a, b, c) reveal the convergence characteristics of the different learning

algorithms for the 3 time series predictions when the hidden neurons activation functions are varied. Figure 5 shows the plot of approximate computational complexity (in Billion Floating Operations - BFlops) for the four different learning algorithms using TSAF and varying the number of hidden neurons.

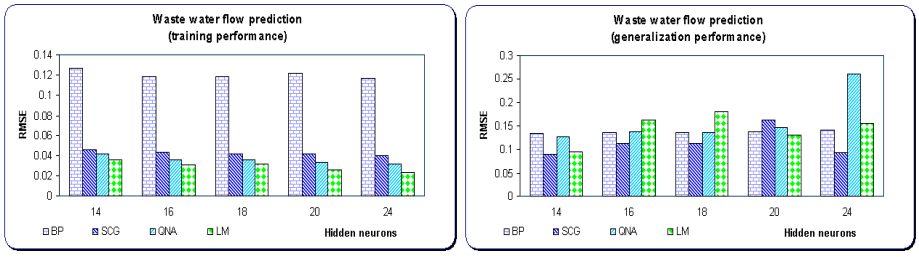
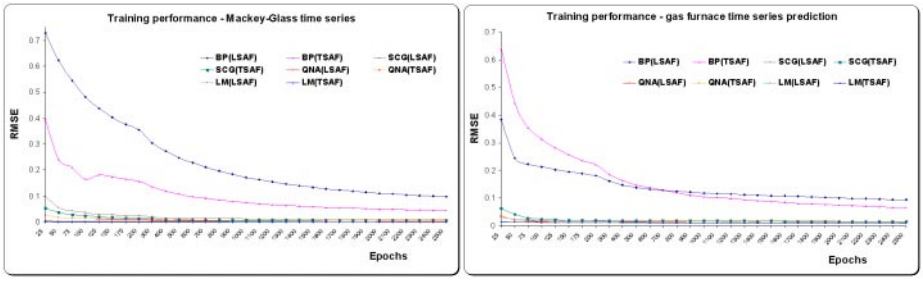
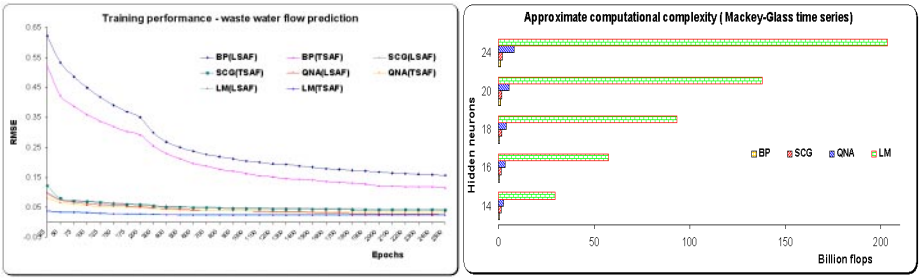


Fig. 3. Waste water flow prediction (training and generalization performance)



4 (a) **4 (b)**

Fig. 4 (a), (b), (c). Convergence characteristics due to change of hidden neuron activation functions (Mackey Glass, gas furnace and waste water prediction)



4 (c) **Fig. 5.** Computational complexity

3.2 Discussion of Results Obtained Using Artificial Neural Networks

Our experiments and the following discussion highlight the difficulty in finding an optimal ANN which is smaller in size, faster in convergence and with the best generalization error.

For Mackey Glass series all the 4 learning algorithms tend to generalize well as the hidden neurons were increased. However the generalization was better when the hidden neurons were using TSAF. LM showed the fastest convergence regardless of architecture and node activation function. However, the figures depicting computational complexity of LM are very amazing. For Mackey glass series (with 14 hidden neurons), when BP was using 0.625 BFlops, LM used 29.4 BFlops. When the hidden neurons were increased to 24, BP used 1.064 BFlops and LM's share jumped to 203.10 BFlops. LM gave the lowest generalization RMSE of 0.0009 with 24 hidden neurons.

For gas furnace series the generalization performance were entirely different for the different learning algorithms. BP gave the best generalization RMSE of 0.0766 with 18 hidden neurons using TSAF. RMSE for SCG, QNA and LM were 0.033 (16 neurons), 0.0376 (18 neurons) and 0.045 (14 neurons) respectively. QNA gave marginally better generalization error when the activation function was changed from TSAF to LSAF.

Wastewater prediction series also showed a different generalization performance when the architecture was changed for the different learning algorithms. BP's best generalization RMSE was 0.136 with 16 hidden neurons using TSAF and that of SCG, QNA and LM were 0.090, 0.1276 and 0.095 with 14 neurons each respectively. SCG's generalization error was improved (0.082) when the activation function was changed from TSAF to LSAF.

In spite of computational complexity, LM performed well for Mackey Glass series. For gas furnace and wastewater prediction SCG algorithm performed better. However the speed of convergence of LM in all the three cases is worth noting. This leads us to the following questions:

What is the optimal architecture for a given problem?
What activation function should one choose?
What is the optimal learning algorithm and its parameters?

The following sections will hopefully guide through some possible solutions to the above questions via our further experiments and inferences.

4. Evolutionary Algorithms (EA)

EAs are population based adaptive methods, which may be used to solve optimization problems, based on the genetic processes of biological organisms [3]. Over many generations, natural populations evolve according to the principles of natural selection and "Survival of the Fittest", first clearly stated by Charles Darwin in "On the Origin of Species". By mimicking this process, EAs are able to "evolve" solutions to real world problems, if they have been suitably encoded. The procedure may be written as the difference equation:

$$x[t+1] = s(v(x[t])) \tag{6}$$

$x[t]$ is the population at time t, v is a random variation operator, and s is the selection operator.

5. Adaptive Learning by Evolutionary Computation (ALEC)

Evolutionary computation has been widely used for training and automatically designing ANNs. However, not much work has been reported regarding evolution of learning mechanisms, which is surprisingly the most challenging part [2]. ALEC mainly focuses on the adaptive search of learning algorithms according to the problem. An optimal design of an ANN can only be achieved by the adaptive evolution of connection weights, architecture and learning rules which progress on different time scales [1]. Figure 6 illustrates the general interaction mechanism with the learning mechanism of the ANN evolving at the highest level on the slowest time scale. All the randomly generated architecture of the initial population are trained by four different learning algorithms and evolved in a parallel environment. Parameters of the learning algorithm will be adapted (example, learning rate and momentum for BP) according to the problem. Figure 7 depicts the basic algorithm of ALEC.

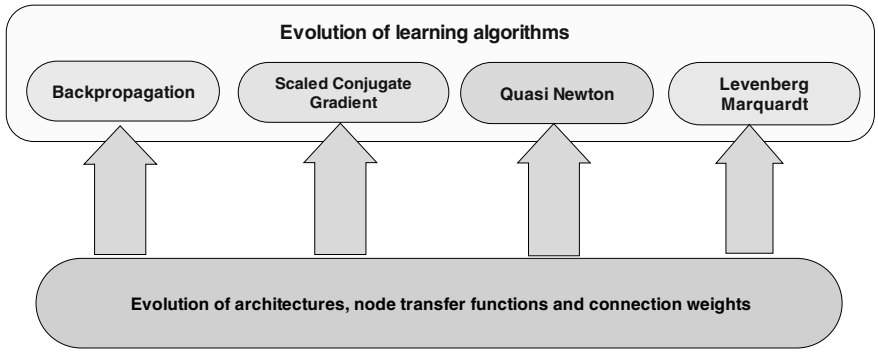


Fig. 6. Interaction of various evolutionary search mechanisms in ALEC

1. *Set t=0 and randomly generate an initial population of neural networks with architectures, node transfer functions and connection weights assigned at random.*
2. *In a parallel mode, evaluate fitness of each ANN using BP/SCG/QNA and LM*
3. *Based on fitness value, select parents for reproduction*
4. *Apply mutation to the parents and produce offspring (s) for next generation. Refill the population back to the defined size.*
5. *Repeat step 2*
6. *STOP when the required solution is found or number of iterations has reached the required limit.*

Fig. 7. ALEC algorithm for evolutionary design of artificial neural networks

6. ALEC: Experimentation Setup and Results

We have applied ALEC to the three-time series prediction problems mentioned in Section 3. For performance comparison, we used the same set of training and test data that were used for ANNs. The parameters used in our experiments were set to be the same for all the 3 problems. Fitness value is calculated based on the RMSE achieved on the test set. The best-evolved ANN is taken to be the best individual in the last generation. As the learning process is evolved separately, user has the option to pick the best ANN (e.g. less RMSE, fast convergence, less computational expensive etc.) among the four learning algorithms. Genotypes were represented using binary coding and the initial populations of network architectures were randomly created based on the following ALEC parameters.

Table 1. Parameters used for evolutionary design of artificial neural networks

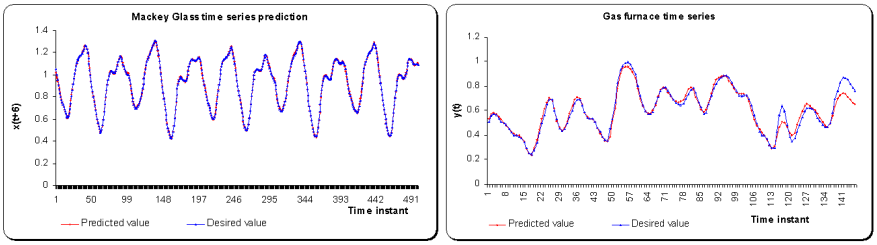
Population size	40
Maximum no of generations	50
Initial number of hidden nodes (random)	5-16
Activation functions	tanh, logistic, linear, sigmoidal, tanh-sigmoidal, log-sigmoidal
Output neuron	linear
Training epochs	2500
Initialization of weights	+/- 0.3
Ranked based selection	0.50
Mutation rate	0.1

Table 2. Performance comparison between ALEC and ANN

Time series	Learning algorithm	ALEC		ANN	
		‡RMSE	*Architecture	‡RMSE	*Architecture
Mackey Glass	BP	0.0077	7(T), 3(LS)	0.0437	24(TS)
	SCG	0.0031	11(T)	0.0045	24(TS)
	QNA	0.0027	6(T),4(TS)	0.0034	24(TS)
	LM	0.0004	8(T),2(TS),1(LS)	0.0009	24(TS)
Gas Furnace	BP	0.0358	8(T)	0.0766	18(TS)
	SCG	0.0210	8(T),2(TS)	0.0330	16(TS)
	QNA	0.0256	7(T),2(LS)	0.0376	18(TS)
	LM	0.0223	6(T),1(LS),1(TS)	0.0451	14(TS)
Waste Water	BP	0.0547	6(T),5(TS),1(LS)	0.1360	16(TS)
	SCG	0.0579	6(T),4(LS)	0.0820	14(LS)
	QNA	0.0823	5(T),5(TS)	0.1276	14(TS)
	LM	0.0521	8(T),1(LS)	0.0951	14(TS)

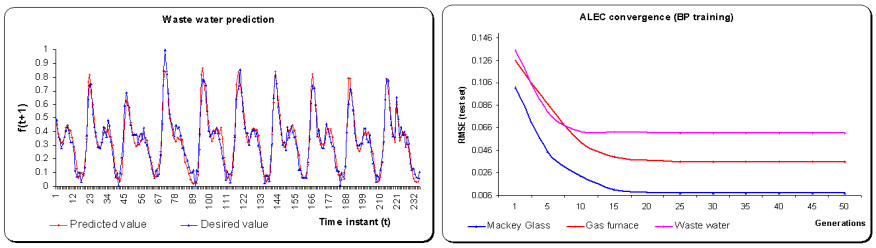
* Architecture = hidden neurons distribution and activation functions.
† Activation function (T=tanh, S=sigmoidal, LS-log-sigmoidal, TS=tanh-sigmoidal)
‡ RMSE on test set

We used a learning rate of (0.2-0.05) and a momentum of (0.2-0.05) for BP algorithm. For SCG the parameter controlling change in weight for second derivative approximation was chosen as 5e-05(+/-100%) and the parameter for regulating the indefiniteness of the Hessian as 5e-07(+/- 100%). In QNA we varied the scaling factors and step sizes after each generation. For LM we used 1 as the factor for memory/speed trade off to converge faster, adaptive learning rate of 0.001 (+/- 100%) and learning rate increasing and decreasing factor of 10 and 0.1 respectively. The experiments were repeated three times and the worst RMSE values are reported.



8(a) **8(b)**

Fig. 8(a). ALEC test results using BP algorithm for Mackey Glass series **(b)** Gas furnace series **(c)** Waste water flow series



8 (c)

Fig. 9. ALEC convergence using BP learning after 50 generations

7. Discussions

Table 2 shows comparative performance between ALEC and ANN. For BP algorithm RMSE error was reduced by 82.4% for Mackey Glass series while it was 52.3% for gas furnace and 60.3% for wastewater prediction. At the same time number of hidden neurons got reduced by 58.4% (Mackey Glass), 55.5% (Gas furnace) and 31.25 (wastewater) respectively. The percentage savings in RMSE and hidden neurons are very much similar for all the four algorithms. LM algorithm gave the best results for Mackey Glass and wastewater prediction and SCG performed well for gas furnace series. Overall LM algorithm gave the best RMSE error even though it is highly computational expensive. We deliberately terminated the training after 2500 epochs (regardless of early stopping in some cases) for ANN and ALEC problems, just to have a generalization performance comparison.

8. Conclusion

Selection of the architecture (number of layers, hidden neurons, activation functions and connection weights) of a network and correct learning algorithm is a tedious task for designing an optimal artificial neural network. Moreover, for critical applications and hardware implementations optimal design often becomes a necessity. In this paper, we have formulated and explored; ALEC: an adaptive computational framework based on evolutionary computation for automatic design of optimal artificial neural networks. Empirical results are promising and show the importance and efficacy of the technique.

In ALEC, our work was mostly concentrated on the evolutionary search of optimal learning algorithms. For the evolutionary search of architectures, it will be interesting to model as co-evolving sub-networks instead of evolving the whole network. Further, it will be worthwhile to explore the whole population information of the final generation for deciding the best solution [7].

References

[1] Abraham A and Nath B, *Optimal Design of Neural Nets Using Hybrid Algorithms*, In proceedings of 6th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2000), pp. 510-520, 2000.

[2] Yao X, *Evolving Artificial Neural Networks*, Proceedings of the IEEE, 87(9):1, 423-1447, 1999.

[3] Fogel D, *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*, 2nd Edition, IEEE press, 1999.

[4] Bishop C M, *Neural Networks for Pattern Recognition*, Oxford Press, 1995.

[5] Moller A F, *A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning*, Neural Networks, Volume (6), pp. 525-533, 1993.

[6] Kasabov N, Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering, The MIT Press, 1996.

[7] Yao X and Liu Y, *Making Use of Population Information in Evolutionary Artificial Neural Networks*, IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, 28(3): 417-425, 1998.

[8] Box G E P and Jenkins G M, *Time Series Analysis, Forecasting and Control*, San Francisco: Holden Day, 1970.

[9] Mackey MC, Glass L, *Oscillation and Chaos in Physiological Control Systems*, Science Vol 197, pp.287-289, 1977.