Credit assignment among neurons in co-evolving populations

Vineet Khare, Xin Yao, Bernhard Sendhoff

2004

Preprint:

This is an accepted article published in Parallel Problem Solving from Nature (PPSN). The final authenticated version is available online at: https://doi.org/[DOI not available]

Parallel Problem Solving from Nature, PPSN, 2004, Springer Verlag

Credit Assignment among Neurons in Co-evolving Populations

Vineet R. Khare[1], Xin Yao[1], and Bernhard Sendhoff[2]

[1] Natural Computation Group, School of Computer Science,
The University of Birmingham, Birmingham B15 2TT, UK
{V.R.Khare, X.Yao}@cs.bham.ac.uk
WWW home page:http://www.cs.bham.ac.uk/research/NC/
[2] Honda Research Institute Europe GmbH, Carl-Legien-Straße 30,
63073 Offenbach/Main, Germany
Bernhard_Sendhoff@de.hrdeu.com
WWW home page:http://www.honda-ri.de

Abstract. Different credit assignment strategies are investigated in a two level co-evolutionary model which involves a population of Gaussian neurons and a population of radial basis function networks consisting of neurons from the neuron population. Each individual in neuron population can contribute to one or more networks in network population, so there is a two-fold difficulty in evaluating the effectiveness (or fitness) of a neuron. Firstly, since each neuron only represents a partial solution to the problem, it needs to be assigned some credit for the complete problem solving activity. Secondly, these credits need to be accumulated from different networks the neuron participates in. This model, along with various credit assignment strategies, is tested on a classification (Heart disease diagnosis problem from UCI machine learning repository) and a regression problem (Mackey-Glass time series prediction problem).

1 Introduction

Co-evolution is one of the ways, used in literature [1–4], to implement the *divide-and-conquer* strategy for tackling complex computational problems. These implementations differ from each other on the basis of interactions between individuals, species and populations. Co-evolution occurs either at intra or inter-population level. The idea is to co-evolve complete solutions (systems) and sub-solutions (modules) simultaneously. These can be divided into two categories – single and two-level co-evolutionary methods. In single-level co-evolutionary methods the sub-components/modules are evolved in separate genetically isolated sub-populations. Fitness evaluation for these modules is carried out either by combining representative individuals from these sub-populations and then passing back the fitness of the system, thus created, to the representative individual [1, 2] or by sharing the fitness of current best module with other modules with similar input-output function [3]. While in two-level co-evolutionary methods [4] modules are evolved in a separate population along with systems

in another. These modules are evaluated on the basis of their contribution to various systems in the second population.

In this work, we present one such two-level co-evolutionary model which co-evolves RBF Networks and Gaussian neurons in separate populations. This is similar to the Symbiotic Adaptive Neuro-Evolution or SANE [4]. Main differences between SANE and this model are the use of RBF Networks instead of Multi-Layer Perceptrons (MLPs) and the credit assignment strategies used for the fitness evaluation of neurons (which are Radial Basis Gaussian Functions (RBGFs) in this case). Also, in each generation, RBF networks can be trained using iRprop [5] (an improved version of Rprop [6]). SANE co-evolves a population of neurons and a population of two-layer feed forward networks. For the fitness of an individual in neuron population it uses the sum of fitnesses of a few good MLPs in the MLP population, in which that individual participates. We argue that this credit assignment strategy among neurons would work only if the problem is decomposable into modules that are mostly independent but otherwise, as is mostly the case, the fitness of a neuron should depend on the other neurons present in the MLP to model the interdependencies between the modules. This inspires the use of Gaussian Kernel Functions along with RBF networks instead of MLPs, where we can get a nice localized property. In other words, the influence of a RBGF on some distance from its center can be neglected. Thus, we should expect them to be relatively independent of each other, i.e., the effectiveness of one RBGF in a network should not be influenced by the presence of others. It does not hold for MLPs, where the effect of all hidden neurons should be taken into account at each point of the input space. Locality in RBF Networks has been exploited [7, 8] to solve the aforementioned credit assignment problem.

The rest of the paper is organized as follows. In Sect. 2 we describe the co-evolutionary model. Section 3 discusses various credit assignment strategies used, followed by the description of experiments and results in Sect. 4. Finally we conclude in Sect. 5 with discussion on results obtained.

2 Co-evolutionary Model

The co-evolutionary model consists of a network population (*netPop*) and a neuron population (*neuPop*). This is similar to SANE except for the use of RBGFs in *neuPop* and RBF networks in *netPop*. Figure 1 gives a pseudo code for the model. RBF networks in *netPop* consist of pointers to RBGFs. Each network is represented by an array of integers that stores the indices of RBGFs from *neuPop*. Each RBGF has the following real valued parameters associated with it

$$\begin{gathered}\mu_j, \sigma_j, w_k, \\ j \in \{0, \ldots, d-1\}, k \in \{0, \ldots, n-1\},\end{gathered}$$

where d and n are the dimensionalities of input and output spaces respectively. In *neuPop* each RBGF is initialized with its center ($\boldsymbol{\mu}$) randomly chosen

from one of the input data points. Widths ($\boldsymbol{\sigma}$) and weights ($\boldsymbol{w}$) are initialized randomly. Individuals in *netPop* point to random individuals in the *neuPop* initially.

```
Initialize neuPop and netPop
repeat
        clear fitness in neuPop and netPop
        for all networks in netPop do
                if LEARNING do
                        initialize all weights randomly
                        train the network partially on training data
                        evaluate the network on validation data
                        if LAMARCKIAN LEARNING do
                                accumulate the changes in neuron parameters in neuPop
                else evaluate the network on validation data
        sort netPop according to fitness
        if LAMARCKIAN LEARNING do
                write back the average changes in neuron parameters in neuPop
        evaluate neurons in neuPop using one credit assignment strategy (Sect. 3)
        sort neuPop according to fitness
        for all networks in netPop do
                reassign pointers to new positions of neurons
        apply variational operators to neuPop and netPop
until fixed number of generations
```

Fig. 1: Pseudo Code

For each individual i in *netPop* one RBF network *net*, consisting of the RBGFs from *neuPop* corresponding to the array entries of the individual i, is constructed. The fitness of this individual depends on how well the RBF Network performs on the validation data set

$$fitness^i = \frac{1}{MSE^{net}_{validation\_data} + \epsilon}, \tag{1}$$

which is the inverse of mean-squared error achieved by RBF network *net* on the validation data set and ϵ is a small constant to prevent very high values of fitness if the mean-squared error approaches zero. Before evaluating the network on validation data it can be trained partially on a training data set to help evolution find good solutions in fewer generations. It can be viewed as lifetime learning of an individual in the evolutionary process. Now we can choose to copy the modified (trained) values of various neuron parameters back to the RBGF population (Lamarckian evolution) or we can choose not to copy them back and just use the trained individual to evaluate fitness (Baldwinian evolution). Since one neuron can take part in many networks, we need to make averaged genotypic changes, in the *neuPop*, corresponding to training in each network in *neuPop*. After training all the networks in the network population average modifications

are made on neurons in *neuPop*, which is similar to the *Averaged Lamarckian Evolution Heuristic* proposed in [9]. Different fitness assignment strategies for fitness of neurons in *neuPop* are discussed in Sect. 3.

The breeding strategy used is similar to the one in SANE, though different crossover and mutation operators are used for the real coded parameters. After sorting the neuron population, each neuron from the top 25% of the population is crossed (crossover operator produces an equally distributed random number in between the two parent chromosomes) with another neuron chosen randomly from the top 25% neurons to produce an offspring. One random parent and the offspring replace two worst performing neurons in the *neuPop*. Thus in each generation 25% of population is replaced by new offspring. This aggressive strategy is balanced by a strong mutation strategy. For mutation, neuron centers and widths, for each neuron in the population, are perturbed with normally distributed noises with zero mean and one standard deviation. Weights are evolved only in absence of learning. Evolutionary algorithm used for the network level is identical to the neuron level. Since networks consist of pointers one point crossover is used. Each array value corresponding to the bottom 75% networks is then mutated to a random value with probability 0.01.

3 Fitness Assignment in Neuron Population

Fitness of a neuron in *neuPop* depends on the fitness of networks in *netPop*. It should depend on the number of networks it participates in, effectiveness of those networks and its contribution to those networks. The following three fitness assignment strategies try to take these factors into consideration.

3.1 Using A Few Good Networks

Each neuron gets the summed fitness of the top 25% networks, from *netPop*, in which the neuron participates. Though the fitness assignment strategy is exactly the same as SANE, yet improvement in performance is expected because of local characteristics of RBGFs. We will compare the performance of SANE and our model with this and other credit assignment strategies in Sect. 4.

3.2 Credit Sharing along Orthogonal Dimensions

Here a credit apportionment strategy (proposed in [8]) is used to split the credit for the overall performance of the RBF network, first, into orthogonal niches and then the credit available for each individual niche is apportioned among individual RBGFs depending on how much they contribute to that niche. In an RBF Network for a set of (say) m basis functions ϕ_is, singular value decomposition (SVD) of transformed training data matrix $\mathbf{A}$, whose elements are $a_{ki} = \phi_i(\mathbf{x}_k)$, gives $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathbf{T}$, where $\mathbf{U}$ has orthogonal columns $\mathbf{u}_1, \ldots \mathbf{u}_m$, $\mathbf{\Sigma}$ is a diagonal matrix with positive or zero elements (the *singular values*), and $\mathbf{V}$ an orthogonal

matrix with entries v_{ij}. Credit available to each basis function ϕ_i is given by (for details refer to [8])

$$credit(\phi_i) = \sum_{j'=0}^{m-1} v_{ij'} f_{j'} / \sigma_{j'} \sum_{j=0}^{m-1} \sigma_j v_{ij} f_j, \tag{2}$$

where $f_j (= \mathbf{u}_j \cdot f)$ is the inner product in the finite Euclidean basis $\{\delta_k\}, k = 1, \ldots, p$, where $\delta_k(\mathbf{x})$ is 1 at $\mathbf{x} = \mathbf{x}_k$ and 0 elsewhere and p is the number of training points. Credit available from a particular network is then multiplied with the network's fitness and summed over all networks in which the neuron participates to obtain neuron fitness.

3.3 Weights-based Credit Assignment

Since the output of a RBF Network (net_j) is a weighted sum of activations of each RBGF (ϕ_i), these weights (w_{ij}s) can be used to evaluate the relative contribution of a RBGF to network j as

$$contribution(\phi_i) = |w_{ij}|^\beta / E(|w_{i'j}|^\beta), \tag{3}$$

where $E(\cdot)$ denotes the mean value over all RBGFs (ϕ_is) in net_j. As discussed in [7], while evolving a single network we should have $\beta \in (1, 2)$ to encourage the desired behaviour of competition and cooperation among RBGFs with similar and different functionalities, respectively. Also, $\beta = 2.0$ produces too much of competition and $\beta = 1.0$ is insufficient to prevent competition among RBGFs with dissimilar activations, while $\beta = 1.5$ yields the desired behavior. In our experiments, we have used the three values of β ($1, 1.5$ and 2) although we do not expect to observe the aforementioned phenomenon as the neurons obtain their fitness from a group of networks rather than a single network.

4 Experimentation and Results

In the following, the co-evolutionary model (Sect. 2) along with different fitness assignment strategies (Sect. 3) are tested on a classification problem (Sect. 4.1) and a regression problem (Sect. 4.2). Different parameters used for experimentation are listed in table 1.

4.1 Heart Disease Diagnosis Problem

Heart disease diagnosis dataset, taken from UCI benchmark database [10], is used as the classification problem. This dataset has 270 instances with 13 continuously valued attributes; based on these attributes, any given pattern has to be classified into 2 classes, which are either presence or absence of the disease. The whole dataset is divided into - Training Set ($\frac{3}{4}$th of full dataset) and Testing Set (remaining $\frac{1}{4}$th). If learning is used then Training Set is further split into

Table 1. Parameters used for experiments.

Parameter	Heart Disease		Mackey-Glass	
	no-learning	learning	no-learning	learning
Size of training data set	202	135	-	500
Size of validation data set	-	67	-	-
Size of test data set	68	68	-	500
Neuron population size	360	360	360	360
Network population size	20	20	20	20
Neurons per network	18	18	18	18
Top Neurons	90	90	90	90
Top Networks	6	6	6	6
Number of best networks for neuron fitness	5	5	5	5
Mutation Rate	10%	10%	10%	10%
Num. of generations - no learning	1000	-	-	-
- Baldwinian learning	-	100	-	200
- Lamarckian learning	-	50	-	100

Training (half of full dataset) and Validation Set ($\frac{1}{4}$th of full dataset). Figure 2 shows the best and average fitnesses of individuals in the two populations for a simulation run with Baldwinian learning and a credit assignment strategy using weights (Sect. 3.3, $\beta = 1.5$).

Plot (a) in fig. 2 also shows the number of neurons that are in use at a given generation. From this plot we can observe that, with generations, the networks in *netPop* are specialized to certain neurons in *neuPop*. After 100 generations these networks use 54 neurons from *neuPop*, with maximum neuron fitness being 553.5. Similar plots with Lamarckian learning show that this number is approximately the same ($\sim$ 53 after 50 generations and 55 after 100 generations) with maximum neuron fitness being 480.4 after 50 and 470.1 after 100 generations along with a much sharper rise in fitness early on. This lower fitness of best neuron is the result of averaging in Lamarckian evolution. Also, we observe the co-evolutionary effect where networks in *netPop* first search for good neurons in *neuPop* and then concentrate on those neurons. In other words, neurons in *neuPop* first increase their use then their fitness. The number of neurons in use drops from approximately 220 to 30-60 and then it stays constant. Networks in *netPop* only use neurons from this subset. Though the plots do not show it, yet we can speculate that this subset does not change a lot as the average neuron fitness increases gradually without large variations. It is because of the aggressive breeding strategy inherent in SANE, which we have used for these experiments.

Table 2 gives the mean and standard deviation values of the percentage classification accuracies achieved by the co-evolutionary model with different credit assignment strategies, for 30 runs with different seeds. These values are listed for experiments without learning, Baldwinian learning and Lamarckian learning. In addition results obtained with SANE are also listed. Few observations can be made from these results - (1) Though SANE performs as good as others on test set, it is unable to fit the training data that well. On average, same credit assign-

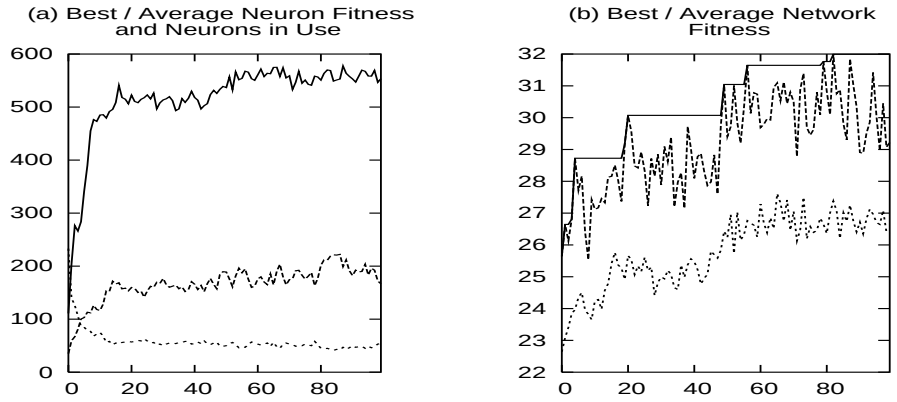

Fig. 2: Fitness variations in *neuPop* and *netPop* with number of generations. Plotted for a simulation run with Baldwinian learning. In plot (a) solid line (—), dashed line (- - -) and dotted line (···) show the maximum neuron fitness, average neuron fitness of the neurons in use and the neurons in use with the number of generations, respectively. In plot (b) they show the fitness of best network over all generations, fitness of best network in current generation and average fitness of all networks in *netPop* with the number of generations, respectively.

Table 2. Heart Disease diagnosis results (percentage correct classifications averaged over 30 runs).

Credit Assignment Strategy		**No learning**		**Learning**					
				Baldwinian			**Lamarckian**		
		Train -ing	Test -ing	Train -ing	Valid -ation	Test -ing	Train -ing	Valid -ation	Test -ing
SANE	Mean	73.89	80.19						
	Std Dev	2.83	3.09						
Using good networks	Mean	84.24	80.07	90.62	91.59	82.99	89.68	89.95	82.45
	Std Dev	3.46	4.41	2.09	2.30	3.06	2.13	2.71	2.78
Credit sharing along	Mean	73.68	75.22	89.95	91.69	82.55	88.58	88.47	83.87
orthogonal dimensions	Std Dev	3.22	4.79	2.41	2.44	3.18	1.27	1.80	2.51
Using weights $\beta = 1.0$	Mean	82.52	78.89	89.98	91.14	83.48	89.80	90.50	82.84
	Std Dev	6.85	5.71	2.27	2.25	2.67	2.63	2.20	1.94
Using weights $\beta = 1.5$	Mean	83.93	79.58	89.93	91.84	82.99	89.23	90.4	82.75
	Std Dev	3.95	3.55	2.15	2.10	3.15	2.07	2.21	3.30
Using weights $\beta = 2.0$	Mean	85.10	80.97	90.49	91.69	82.21	89.16	90.05	82.89
	Std Dev	2.07	3.88	2.18	2.02	1.98	1.96	2.58	2.40

ment strategy with RBF networks produce 170 correct classifications out of 202 as against 149 correct classifications by SANE. (2) Learning does improve the performance significantly. (3) Lamarckian learning does not improve the performance over Baldwinian learning, though it produces similar results in half the number of generations (table 1). (4) As expected (Sect. 3.3) different values of β do not produce significantly different results. (5) These results are comparable with the existing results available in the literature (83.9% [11] and 84.9% [12]).

4.2 Mackey-Glass Time Series Prediction Problem

The Mackey-Glass time series is generated by the following differential equation using fourth order Runge-Kutta method with initial condition $x(0) = 1.2$ and time step of 1.

$$\dot{x}(t) = \beta x(t) + \frac{\alpha x(t-\tau)}{1 + x^{10}(t-\tau)}, \tag{4}$$

where $\alpha = 0.2$, $\beta = -0.1$, $\tau = 17$ as used in [7, 8, 13–15]. Each network receives four past data points $x(t)$, $x(t-6)$, $x(t-12)$ and $x(t-18)$ as inputs and predicts 6 time steps ahead ($x(t+6)$). For predicting further steps ahead iterative predictions of $x(t+6)$, $x(t+12)$, ..., $x(t+84)$ are used during testing, e.g. the network uses its own prediction for time step $t+6$ and the input points $x(t)$, $x(t-6)$ and $x(t-12)$ to predict $x(t+12)$. For training $x(t+6)$ values are used as targets. This setup is same as that used in [13, 16]. 500 points starting from 118 are used for training and following 500 points are used for testing. No validation set is used for comparison purposes. For fitness evaluation of networks in (1) training set is used instead of validation set.

Table 3 gives the mean and standard deviation values, of the normalized root-mean-squared (RMS) errors achieved on test set by the co-evolutionary model with different credit assignment strategies, for 30 runs with different seeds. Here normalized RMS errors are obtained by dividing the absolute RMS error values by the standard deviation of $x(t)$ [13, 14, 16]. Results are listed only for experiments with Baldwinian learning as no-learning does not produce good generalization performance (normalized RMS error ~ 1.2 in 2000 generations) and Lamarckian learning results are again very similar to Baldwinian learning. Observations 2, 3 and 4 from Sect. 4.1 are again verified from these results. These results are comparable to those obtained by an ensemble of RBF networks with similar computational overhead (20 networks with 18 neurons each), though there are better results available in the literature [13] (see table 3). Each network in the ensemble is trained using iRprop with centers initialized using K-means-Clustering [17, pages 187–188], widths initialized randomly between 0 and 1 and weights and bias values initialized by means of linear regression. After training, best network was tested on testing data.

Table 3. Mackey-Glass time series prediction results (normalized RMS Error on test set, averaged over 30 runs) with Baldwinian learning.

Credit Assignment Strategy		**Generalization Performance**	
		$\Delta t = 6$	$\Delta t = 84$
Using good networks	Mean	0.0296	0.0995
	Std Dev	0.0019	0.0156
Credit sharing along	Mean	0.0299	0.1021
orthogonal dimensions	Std Dev	0.0020	0.0093
Using weights $\beta = 1.0$	Mean	0.0303	0.1032
	Std Dev	0.0015	0.0085
Using weights $\beta = 1.5$	Mean	0.0301	0.1013
	Std Dev	0.0019	0.0119
Using weights $\beta = 2.0$	Mean	0.0298	0.1043
	Std Dev	0.0019	0.0140
Other Results			
Ensemble of 20 RBF	mean	0.0358	0.1106
networks	Std Dev	0.0038	0.0206
EPNet [13]	mean	0.02	0.06

5 Conclusion

A two-level co-evolutionary model which co-evolves RBF Networks and Gaussian neurons in separate populations was presented in this work along with different credit assignment strategies for evaluating the fitness of neurons in neuron population. These strategies were evaluated, on the basis of their performance, in conjunction with the co-evolutionary model on two test problems. In between these two populations a co-evolutionary effect was observed, where networks in network population first search for good neurons in neuron population and then consolidate on these neurons.

It was argued that the use of RBF networks instead of MLPs was beneficial in such a co-evolutionary model because of local characteristics of Gaussian neurons. Which was also verified by a comparison of the model with SANE on heart disease diagnosis problem. Other than the use of RBF networks, this model only differed with SANE in real-coded parameters and variational operators and had same fitness assignment strategy and breeding strategy.

The introduction of learning produced a significant improvement in performance of the system on both heart disease diagnosis problem and Mackey-Glass time series prediction problem. The number of generations for experiments with or without learning were chosen to allow similar running times. Further, Lamarckian learning was able to produce similar results in half the number of generations.

The results obtained on the two test problems were comparable to the results available in literature, although different credit assignment strategies used did not produce significantly different results, specially in the presence of learning, for the two test problems used. Also, more experiments with other benchmarks

are needed to make firm conclusions about the applicability of the introduced approach.

References

1. Potter, M.A., Jong, K.A.D.: Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. Evolutionary Computation **8** (2000) 1–29
2. Yong, C.H., Miikkulainen, R.: Cooperative Coevolution of Multi-Agent Systems. Technical Report AI01-287, Department of computer Sciences, The University of Texas at Austin, Austin, TX 78712 USA (2001)
3. Smalz, R., Conrad, M.: Combining Evolution With Credit Apportionment: A New Learning Algorithm for Neural Nets. Neural Networks **7** (1994) 341–351
4. Moriarty, D.E., Miikkulainen, R.: Forming Neural Networks Through Efficient and Adaptive Coevolution. Evolutionary Computation **5** (1997) 373–399
5. Igel, C., Hüsken, M.: Empirical Evaluation of the Improved Rprop Learning Algorithm. Neurocomputing **50** (2003) 105–123
6. Riedmiller, M., Braun, H.: A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP algorithm. In: Proceedings of the IEEE International Conference on Neural Networks, San Francisco, CA (1993) 586–591
7. Whitehead, B.A., Choate, T.D.: Cooperative-Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction. IEEE Transactions on Neural Networks **7** (1996) 869–880
8. Whitehead, B.A.: Genetic Evolution of Radial Basis Function Coverage Using Orthogonal Niches. IEEE Transactions on Neural Networks **7** (1996) 1525–1528
9. Hüsken, M., Gayko, J.E., Sendhoff, B.: Optimization for Problem Classes - Neural Networks that Learn to Learn. In X.Yao, ed.: IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, IEEE Press (2000) 98-109.
10. Blake, C., Merz, C.: UCI Repository of machine learning databases (1998) http://www.ics.uci.edu/~mlearn/MLRepository.html.
11. Khare, V., X.Yao: Artificial Speciation and Automatic Modularisation. In Wang, L., Tan, K.C., Furuhashi, T., Kim, J.H., Yao, X., eds.: Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'02). 1, Singapore (2002) 56–60
12. Yao, X., Liu, Y.: Making Use of Population Information in Evolutionary Artificial Neural Networks. IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics **28** (1998) 417–425
13. Yao, X., Liu, Y.: A New Evolutionary System for Evolving Artificial Neural Networks. IEEE Transactions on Neural Networks **8** (1997) 694–713
14. Farmer, J.D., Sidorowich, J.J.: Predicting chaotic time series. Physical Review Letters **59** (1987) 845–848
15. Mackey, M.C., Glass, L.: Oscillation and chaos in physiological control systems. Science **197** (1977) 287–289
16. Martinetz, T.M., Berkovich, S.G., Schulten, K.J.: 'Neural-Gas' Network for Vector Quantization and its Application to Time-Series Prediction. IEEE Transactions on Neural Networks **4** (1993) 558–569
17. Bishop, C.M.: Neural Networks for Pattern Recogntion. Oxford University Press (1995)