EVOLUTIONARY COMPUTATION FOR ON-LINE AND OFF-LINE PARAMETER TUNING OF EVOLVING FUZZY NEURAL NETWORKS

ZEKE S. H. CHAN and NIKOLA KASABOV

Knowledge Engineering and Discovery Research Institute
Auckland University of Technology, Aucklqnd, New Zealand
{shun.chan, nkasabov}@aut.ac.nz

Received 30 June 2004
Revised 13 August 2004

This work applies Evolutionary Computation to achieve completely self-adapting Evolving Fuzzy Neural Networks (EFuNNs) for operating in both incremental (on-line) and batch (off-line) modes. EFuNNs belong to a class of Evolving Connectionist Systems (ECOS), capable of performing clustering-based, on-line, local area learning and rule extraction. Through EC, its parameters such as learning rates and membership functions are continuously adjusted to reflect the changes in the dynamics of incoming data. The proposed methods are tested on the Mackey-Glass series and the results demonstrate a substantial improvement in EFuNN's performance.

Keywords: Evolving Neural Networks; Fuzzy logic system; Genetic Algorithm; On-line learning; Clustering.

1. Introduction

Many real-world problems, such as biological data processing, electricity load forecasting, adaptive speech recognition, are continuously changing non-linear processes that require fast, adapting non-linear systems capable of following the process dynamics and discovering the rules of these changes. Since the 80's, many Artificial Neural Network (ANN) models such as the popular multilayer perceptrons (MLP) and radial basis network (RBF) have been proposed to capture the process non-linearities that often fail classical linear systems. In the late 90's, a class of self-adaptive connectionist systems called Evolving Connectionist Systems (ECOS) was proposed for incremental learning and knowledge discovery.[1,2,4] ECOS are capable of performing the following functions: adaptive learning, incremental learning, lifelong learning, online learning, constructivist structural learning that is supported by biological facts, selectivist structural learning and knowledge-based learning. Incremental learning is supported by the creation and the modification of the number and the position of neurons and their connection weights in a local problem space as new data comes. Fuzzy rules can be extracted for knowledge discovery. Derivatives of ECOS, including Evolving Fuzzy Neural Network (EFuNN)[2,4], Evolving Clustering Method (ECM) and Dynamic Evolving Neural-Fuzzy Inference

System (DENFIS)[1,2] have been applied to dynamic time-series prediction, gene expression clustering and gene regulatory network inference, speech and image recognition, brain signal analysis and modeling.[2]

While ECOS models were designed to be self-evolving systems, it is the goal of this work to extend this capability through applying Evolutionary Computation (EC) [3], a robust global optimization algorithm, to perform both on-line and off-line optimization of the control parameters. We classify the control parameters as either dynamic or static parameters, depending on their sensitivity to changes during the training process, and apply on-line EC to optimize the dynamic control parameters during incremental learning, and off-line EC to optimize the static ones during batch learning. In this work, we introduce such EC application to a class of ECOS called EFuNN, and choose the learning rates of the neurons as the dynamic parameters and the fuzzy membership functions (MFs) as the static parameters to be optimized. As a preliminary experiment, the EC-optimized EFuNN is tested with the Mackey-Glass time-series. Results show that EC improves EFuNN's performance in both the incremental and batch learning modes.

This paper is organized as follows. In section 2, the basic principles of EFuNN and its mechanism of incremental and batch learning are described. In section 3, we introduce Evolutionary Computation (EC) methods for both on-line optimization of EFuNN incremental learning, and off-line optimization of batch learning. Experimental results on Mackey-Glass series are also presented. Finally, conclusions of this work are drawn in section 4.

2. Principles of the EFuNN models

EFuNNs[4] belong to the class of fuzzy neural networks that incorporate the interfaces of fuzzy rules and inferences into the neural network connectionist structures. It has a five-layer structure. The first layer is the input layer that receives the actual input vector $\mathbf{x}^{(t)}$ where t represents the time at which $\mathbf{x}^{(t)}$ is recorded. The input vector is mapped onto the fuzzy space Z_{in}: $\mathbf{x} \rightarrow \mathbf{d_{in}}$ through a pre-defined input fuzzy quantization function Z_{in}. The fuzzified input vectors represent the second layer. The third layer contains rule nodes $\{r_j\}_{j=1}^{rmax}$ that associate the fuzzy input vectors $\mathbf{d_{in}}$ to the fuzzy output vectors $\mathbf{d_{out}}$. Each rule node r_j is represented by two vectors of connection weights, $W1(r_j)$ and $W2(r_j)$ which correspond to the coordinates of node r_j in the fuzzy input space and fuzzy output space respectively, and by two scalar radii, R_j and E_j which define the input and output association thresholds respectively. Association between rule node r_j and the fuzzy vectors $\{\mathbf{d_{in}}, \mathbf{d_{out}}\}$ is determined by whether the normalized distance (defined next) between $\mathbf{d_{in}}$ and $W1(r_j)$ falls within the threshold R_j and that between $\mathbf{d_{out}}$ and $W2(r_j)$) falls within the threshold E_j, i.e. the following two conditions:

$$D(\mathbf{d_{in}},\ W1(r_j)\ ) < R_j \tag{1}$$

$$D(\mathbf{d_{out}},\ W2(r_j)\ ) < E_j \tag{2}$$

where $D(\mathbf{d}_1,\mathbf{d}_2)=\|\mathbf{d}_1-\mathbf{d}_2\|/\|\mathbf{d}_1+\mathbf{d}_2\|$ denotes the local normalized fuzzy difference. Finally, the fourth layer represents the fuzzy vector $\mathbf{d_{out}}$ and the fifth represents the defuzzified output $\mathbf{y}^{(t)}$, and the two quantities are linked by Z_{out}:$\mathbf{y} \rightarrow \mathbf{d_{out}}$ given Z_{out} the pre-defined output fuzzy quantization function.

The connection weights, association thresholds (the radii) and number of associated data of the rule nodes can be represented as a set of rules, e.g. **IF** [the input data vector $\mathbf{x}=(x_1,x_2)$ belongs rule node r_j defined by (x_1 is *Small* (0.7) and x_2 is *Large* (0.8)) and an input association threshold of R_j=0.3 and which has been associated

with Nex_j =10 data] **THEN** [the output vector **y**=(y) is in the area defined by (y is *Very Large* (0.95)) ± output (error) threshold $E^{(j)}$].

2.1. *Incremental learning and prediction in EFuNN*

Incremental learning of EFuNN proceeds as follows. When a new data pair {**x**,**y**} arrives, the input **x** is fed through the first layer and is fuzzified into $\mathbf{d}_{\mathbf{in}}$ in the second layer, while the output **y** is fed through the fifth layer and is fuzzified into $\mathbf{d}_{\mathbf{out}}$ in the fourth layer. If the fuzzified input-output pair {$\mathbf{d}_{\mathbf{in}}$,$\mathbf{d}_{\mathbf{out}}$} is associated with an existing rule node r_j, i.e. meeting both criteria stated in (1) and (2), then r_j becomes the "winning node" and both the connection weights $W1(r_j)$ and $W2(r_j)$ and the radii R_j and E_j will be updated, otherwise a new rule node r_{new} will be created and its connection weights {$W1(r_{new})$,$W2(r_{new})$} are simply set to the fuzzified input-output pair {$\mathbf{d}_{\mathbf{in}}$,$\mathbf{d}_{\mathbf{out}}$}. Note that the choice of fuzzy membership functions and the corresponding fuzzification/defuzzification methods are problem-dependent. In this work, we employ triangular membership function. The weights and radii are updated through the simple error-correction equations

$$W1(r_j^{(t+1)}) = W1(r_j^{(t)}) + \frac{l_{j,1}}{Nex_j + 1}(\mathbf{d}_{\mathbf{in}} - W1(r_j^{(t)})) \tag{3}$$

$$W2(r_j^{(t+1)}) = W2(r_j^{(t)}) + \frac{l_{j,2}}{Nex_j + 1}(\mathbf{d}_{\mathbf{out}} - W2(r_j^{(t)})) \tag{4}$$

$$R_j^{(t+1)} = R_j^{(t)} + \frac{l_{j,1}}{Nex_j + 1} D(\mathbf{d}_{\mathbf{in}} - W1(r_j^{(t)})) \tag{5}$$

$$E_j^{(t+1)} = E_j^{(t)} + \frac{l_{j,2}}{Nex_j + 1} D(\mathbf{d}_{\mathbf{out}} - W2(r_j^{(t)})) \tag{6}$$

Nex_j is the number of data previously associated with rule node r_j. The division of the error correction term in each (3)-(6) by (Nex_j+1) is used to statistically balance the weighting of the current data to Nex_j previously associated data. $l_{j,1}$ and $l_{j,2}$ are the learning rates for the weights $W1(\cdot)$ and $W2(\cdot)$ respectively. When Nex_j becomes large, the effect of the new data becomes small and learning decelerates. This can be alleviated by upper bounding Nex_j (this scheme is not implemented in this work). Here, we assume one uniform learning rate l_1 across all $l_{j,1}$ and one uniform learning rate l_2 across all $l_{j,2}$, i.e. l_1=$l_{j,1}$ and l_2=$l_{j,2}$, $\forall j$.

Prediction $\mathbf{y}_\mathbf{p}$ is generally made from the associated rule node r_j, given by simply defuzzifying the output connection weights $W2(r_j)$, i.e. $\mathbf{y}_\mathbf{p}$=$F_{out}^{-1}(W2(r_j))$. It can also be a weighted average of predictions (by distance from the rule nodes) made from multiple rule nodes.

In this work, we use on-line EC to optimize the learning rates l_1 and l_2 during incremental training. Details are discussed in section 3.2.

2.2. *Batch learning in EFuNN*

In EFuNN, batch learning brings the advantage of global parameter tuning and past prediction smoothing using the complete set of training data and not just single data points at the arrival time. In this work, we use the *K*-mean clustering algorithm to fine-tune the location of the rule nodes during unsupervised training and off-line EC to optimize the fuzzy input and output MFs. The off-line EC application is discussed in section 3.3.

2.3. *Alternating incremental and batch learning in EFuNN*

Incremental learning offers fast, local parameter tuning and continuous model evolution over new incoming data, while batch learning offers slower but nevertheless more accurate global parameter tuning based upon the entire set of data or a substantial portion of it. The choice of learning scheme is application-dependent. A particularly interesting scheme is to alternate between incremental and batch learning within one ECOS system, i.e. using the former when the model must operate on-line, and using the later when the model is allowed off-line. For example, a finance company may use incremental learning for on-line stock price prediction when data flows in during the market trading hours, and switch to batch learning for refining the model parameters (globally) and smoothing the past predictions when there is no incoming data after the market has closed.

3. Evolutionary Computation Methods for On-line and Off-line EFuNN Parameter Tuning

3.1. *Principles of Evolutionary Computation*

Evolutionary Computation (EC)[3,5,6] represents a general class of global optimization algorithms that imitate the evolutionary process explained by Darwinian Natural Selection. Its models include Evolutionary Strategies (ES) proposed by Schwefel et al. in the mid 1960s, Genetic Algorithms (GA) proposed by Holland in 1975, and Genetic Programming (GP) by Koza in 1992. EC searches with multiple search points and requires only the function evaluation of each point. It is therefore robust for optimizing complex problems that lack derivative information and contain multi-modality, which are the features that defeat classical, derivative-based search methods. However, the large number of function evaluations causes EC to be computationally intensive. For this reason although the earliest form of EC has existed since the mid 1960s, EC received research attention only after the availability of high-speed computers in the last two decades.

There is a rich literature on the application of EC to ANNs, in particular for optimizing the network weights and architecture[7-9] and for regularization[10]. These methods are mainly applied to conventional ANN models for enhancing batch learning. They differ from our work in two aspects: first, we apply EC to tune the control parameters only, since the network weights and architecture is self-evolving; second, our method is applied to both batch and incremental learning.

3.2. *Evolutionary Computation for on-line parameter tuning*

While EC is an intrinsically computation-intensive algorithm for offline problems, there are also recent reports on its application to online problems such as controller tuning and optimal planning.[11-17] In the example references, online-EC are characterized by small population size (6-50) and high selective pressure (use of elitist selection and low selection ratio), which aims to minimize computational intensity and to accelerate convergence (at the cost of lesser exploration in the search space) respectively. Their applications involve optimizing only a small number of parameters (2-10), which are often the high-level tuning parameters of a model. For applications involving larger number of parameters, parallel EC can be used to accelerate the process.

In this work, on-line EC is applied to tune the learning rates l_1 and l_2 of EFuNN during incremental training with the objective of minimizing the prediction error over

the last n_{last} data. The learning rates are sensitive to the dynamics of the data as they control how fast the model adapts to the data; they are therefore dynamic parameters that require on-line adjustment. For rapidly changing data, learning rates should take higher value to obtain fast model response, and vice-versa.

The on-line EC is implemented with Evolutionary Strategies (ES). Each individual $\mathbf{s}^{(k)}=(s_1^{(k)},s_2^{(k)})$ is a 2-vector solution to l_1 and l_2. Since there are only two parameters to optimize, ES requires only a small population and a short evolutionary time. In this case, we use μ=1 parent and λ=10 offspring and a small number of generations of gen_{max}=20. We set the initial values of $\mathbf{s}$ to (0.2,0.2) in the first run and use the previous best solution as the initial values for subsequent runs. Using a non-randomized initial population encourages a more localized optimization and hence speeds up convergence. We use simple Gaussian mutation (explained later) with standard deviation σ=0.1 (empirically determined) for both parameters to generate new points. For selection, we use the high selection pressure ($\mu+\lambda$) scheme to accelerate convergence, which picks the best μ of the joint pool of μ parents and λ offspring to be the next generation parents.

The fitness function or the optimization objective function is the prediction error over the last n_{last} data, generated by using EFuNN model at (t-n_{last}) to perform incremental learning and predicting over the last n_{last} data using the EFuNN model at (t-n_{last}). The optimal value of n_{last} is problem-dependent. Obviously, the smaller n_{last}, the faster the learning rates adapts and vice-versa. However, since the effect of changing the learning rates is usually not expressed immediately but after a certain period, the fitness function can be noisy and inaccurate if n_{last} is too small. In this work we set n_{last} =50. The overall algorithm is as follows:

Step 1: ***Population Initialization***	Reset generation counter *gen*. Initialize a population of μ parents with the previous best solution (l_1',l_2') $\mathbf{s_{PA}}^{(k)}=(l_1',l_2')$, $k=\{1,2,...\mu\}$
Step 2: ***Reproduction***	Randomly select one of the μ parents, $\mathbf{s_{PA}}^{(r)}$, to undergo Gaussian mutation to produce a new offspring $\mathbf{s}_{\mathbf{OF}}^{(i)} = \mathbf{s}_{\mathbf{PA}}^{(r)} + z^{(i)}$ where $z^{(i)} \sim N\left(0,\begin{bmatrix}\sigma^2 & 0\\ 0 & \sigma^2\end{bmatrix}\right)$, $i=\{1,2,...\lambda\}$ $N(\mathbf{a},\mathbf{B})$ represents a Normal distribution with mean $\mathbf{a}$ and covariance $\mathbf{B}$ and "~" denotes the sample taken from the corresponding distribution.
Step 3: ***Fitness Evaluation***	Apply each of the λ offspring to the EFuNN model at (t-n_{last}) to perform incremental learning and prediction using data in [t-n_{last} , t]. Set the respective prediction error as fitness.
Step 4: ***Selection***	Perform ($\mu+\lambda$) selection
Step 5:	Increment *gen*. Stop if $gen \geq gen_{max}$ or if no fitness improvement has been recorded over 3 generations; otherwise go to *step 2*.

3.2.1. *Testing on-line EC-EFuNN on Mackey Glass Series*

To verify the effectiveness of the proposed on-line ES, we first train EFuNN with the first 500 data of the Mackey-Glass series (using $x(0)=1.2$ and $\tau=17$) to obtain a stable model, and then apply on-line ES to optimize the learning rates over the next 500 data. The corresponding prediction error is recorded. Example results are shown in Fig. 1 and Fig. 2.

Fig. 1 shows an example of the evolution of the least RMSE (used as the fitness function) and the learning rates. The RMSE decreases uni-directionally, which is characteristics of $(\mu+\lambda)$ selection because the best individual is always kept. The optimal learning rates are achieved quickly after 14 generations.

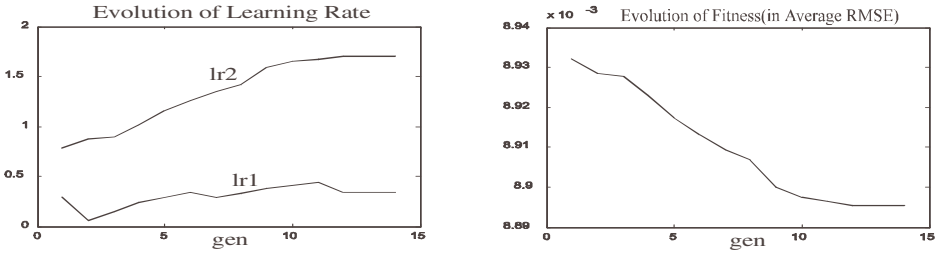

Fig. 1. Evolution of the best fitness and the learning rates over 15 generations.

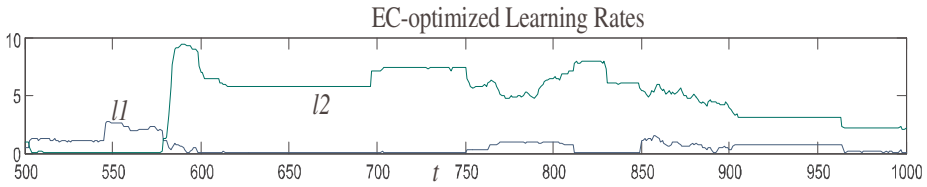

Fig. 2. Optimized learning rates l_1 and l_2 over the period t=[500,1000].

Fig. 2 shows the dynamics of the learning rate over the period t=[500,1000]. Both learning rates l_1 and l_2 vary considerably over the entire course with only short stationary moments, showing that they are indeed dynamic parameters. The average RMSE for on-line prediction one step ahead obtained with and without on-line EC are 0.0056 and 0.0068 respectively, showing that on-line EC is effective in enhancing EFuNN's prediction performance during incremental learning.

3.3. *Evolutionary Computation for off-line parameter tuning*

Most reported applications of EC are off-line problems, see e.g. Refs. 9,10,18, mainly because of EC's requirement of intense computation for the population search. In this work, we apply an off-line ES to optimize the fuzzy input and output membership functions (MFs) at the second and fourth layers respectively during batch learning with the objective of minimizing the training error. The approach is similar to that in Ref. 16. Relative to the learning rates, MFs are static control parameters that remain constant over time and hence require less frequent updating. For both the input and output MFs, we use the common triangular function, which is completely defined by the position of the MF center.

Given that there are p input variables and p_{MF} fuzzy quantization levels for each input variables, m output variables and m_{MF} fuzzy quantization levels for each output variable, there are $n_c=(p\times p_{MF}+m\times m_{MF})$ centers $\mathbf{c}=[c_1,c_2,...,c_{nc}]$ to be optimized, which exceeds that (only two) in the on-line case. Thus naturally, the offline ES requires a

larger population size and longer evolution period. This is not problematic as the learning is performed off-line and has lesser time constraint.

The off-line ES represents each individual as an ($p \times p_{MF} + m \times m_{MF}$) real vector solution to the positions of the MFs. We use μ=5 parent and λ=20 offspring and a relatively larger number of generations of gen_{max}=40. Each individual of the initial population is a copy of the evenly distributed MFs within the boundaries of the variables. Standard Gaussian mutation is used for reproduction with σ=0.1 (empirically determined). Every offspring is checked for the membership hierarchy constraint, i.e. the value of the higher MF must be larger than that of the lower MF. If the constraint is violated, the individual is resampled until a valid one is found.

Step 1: ***Population Initialization***	Reset generation counter *gen*. Initialize a population of μ parents with the evenly distributed MFs $[c_1', c_2', ..., c_{n_c}']$ $\mathbf{s}_{\mathbf{PA}}^{(k)} = [c_1', c_2', ..., c_{n_c}']$, $k=\{1,2,...\mu\}$
Step 2: ***Reproduction***	Randomly select one of the μ parents, $\mathbf{s}_{\mathbf{PA}}^{(r)}$, to undergo Gaussian mutation to produce a new offspring $\mathbf{s}_{\mathbf{OF}}^{(i)} = \mathbf{s}_{\mathbf{PA}}^{(r)} + z^{(i)}$ and $z^{(i)} \sim N\left(0, \sigma^2 \mathbf{I}\right)$ for $i=\{1,2,...,\lambda\}$ I is the $n_c \times n_c$ identity matrix. Resample if the membership hierarchy constraint is violated.
Step 3: ***Fitness Evaluation***	Apply each of the λ offspring to the EFuNN model at (t-n_{last}) to perform incremental learning and prediction using data in [t-n_{last}, t]. Set the respective prediction error as fitness.
Step4: Selection	Perform (μ+λ) selection
Step 5:End	Increment *gen*. Stop if $gen \geq gen_{max}$, otherwise go to *step 2*.

3.3.1. *Testing off-line EC-EFuNN on the Mackey-Glass Series*

The proposed off-line EC is tested on the same Mackey-Glass Series described above. We first perform incremental training on EFuNN with the first 1000 data of the Mackey-Glass series to obtain a stable model, and then apply off-line ES during batch learning (over the same 1000 data) to optimize the input and output MFs. Example results are shown in Fig. 3.

Fig. 3(a) shows the evolution of the best fitness recorded in each generation. The uni-directional drop in prediction error shows that the optimization of MFs has a positive impact on improving model performance.

Fig. 3(b) shows the initial MFs and the EC-optimized MFs, and Fig. 3 (c) shows the frequency distribution of the fourth input variable $\{x_4^{(t)}\}$. Clearly, the off-line ES evolves the MFs towards the high frequency positions, which maximizes the precision for fuzzy quantization and in turn yields higher prediction accuracies. The training RMSEs are 0.1129 and 0.1160 for EFuNN with and without off-line ES respectively, showing that the ES-optimized fuzzy MFs is effective in improving EFuNN's data tracking performance. Greater improvement can be achieved by increasing the number of generations and the population sizes for the ES.

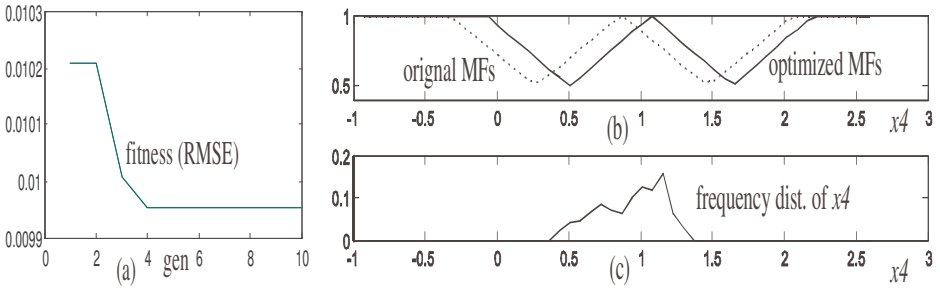

Fig. 3. (a) The evolution of the best fitness from the off-line ES. (b) Initial membership functions and EC-optimized membership functions, (c) Frequency distribution of the first input variable.

4. Conclusions

This paper reports a preliminary effort to incorporate both on-line and off-line EC to EFuNN, a class of ECOS, for parameter optimization. The on-line EC operates during incremental learning and optimizes the learning rates (which are sensitive to the dynamics of the data), while the off-line EC operates during batch learning and optimizes the fuzzy MFs which are relatively static over time. Both the on-line and off-line EC-optimized EFuNNs are tested on the Mackey-Glass series and show promising results. The control parameters are accurately evolved towards their optimal values and the overall prediction performance is improved.

This work presents the first step towards constructing a completely self-evolving ECOS that require minimal manual tuning. Future works include first, evolving and self-adapting of more control parameters, e.g. the rule node radii for both incremental and batch learning, and second, developing a totally integrated ECOS that is embedded with tailored-EC methods for seamless, unified operation.

Acknowledgments

This research is supported by the KEDRI postdoctoral fellow research fund. Discussions with D. Greer and Q. Song are gratefully acknowledged.

References

1. N. Kasabov and Q. Song, DENFIS: Dynamic, evolving neural-fuzzy inference systems and its application for time-series prediction, *IEEE Trans. on Fuzzy Systems* **10** (2002) 144-154.
2. N. Kasabov, *Evolving connectionist systems - methods and applications in bioinformatics, brain study and intelligent machines* (Springer Verlag, London-New York, 2002).
3. T. Baeck, *Evolutionary algorithm in theory and practice: evolution strategies, evolutionary programming, genetic algorithms* (Oxford University Press, New York, 1995).
4. N. Kasabov, Evolving fuzzy neural networks for on-line supervised/unsupervised, knowledge-based learning, *IEEE Trans. SMC - part B, Cybernetics* **31** (2001) 902-918.
5. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and machine Learning* (Addison-Wesley, MA, 1989).
6. J. H. Holland, *Adaptation in natural and artificial systems* (The University of Michigan Press, Ann Arbor, MI, 1975).
7. D. B. Fogel, Using evolutionary programming to create neural networks that are capable of playing tic-tac-toe, *Proc. International Joint Conference on Neural Networks*, New York, 1993.
8. D. B. Fogel, Evolving neural networks, *Biol. Cybern.* **63** (1990) 487-493.
9. X. Yao and Y. Liu, A New Evolutionary System for Evolving Artificial Neural Networks, *IEEE Transactions on Neural Networks* **8** (1997) 694-713.

10. Z. S. H. Chan, H. W. Ngan, A. B. Rad, and T. K. Ho, Alleviating "overfitting" via genetically-regularised neural network, *Electronics Letter* (2002).
11. W. G. da Silva, P. P. Acarnley, and J. W. Finch, Application of genetic algorithms to the online tuning of electric drive speed controllers, *IEEE Transactions on Industrial Electronics*, **47** (2000) 217-219.
12. C. F. Juang, A TSK-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithms, *IEEE Transactions on Fuzzy Systems*, **10** (2002) 155-170.
13. D. A. Linkens and H. O. Nyongesa, Genetic algorithms for fuzzy control.2. Online system development and application, *IEE Proceedings of Control Theory and Applications* **142** (1995) 177-185.
14. S. Mishra, P. K. Dash, P. K. Hota, and M. Tripathy, Genetically optimized neuro-fuzzy IPFC for damping modal oscillations of power system, *IEEE transactions on Power Systems* **17** (2002) 1140-1147.
15. I. K. Nikolos, K. P. Valavanis, N. C. Tsourveloudis, and A. N. Kostaras, Evolutionary Algorithm Based Offline/Online Path Planner for UAV Navigation, *Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics* (2003) 1.
16. A. Rajapakse, K. Furuta, and S. Kondo, Evolutionary learning of fuzzy logic controllers and their adaptation through perpetual evolution, *IEEE Transactions on Fuzzy Systems* **10** (2002) 309-321.
17. J. Tippayachai, W. Ongsakul, and I. Ngamroo, Parallel micro genetic algorithm for constrained economic dispatch, *IEEE Transactions on Power Systems* **17** (2002) 790-797.
18. D. Quagliarella, *Genetic algorithms and evolution strategy in engineering and computer science : recent advances and industrial applications* (John Wiley & Sons, New York, 1998).