15

Development of Fuzzy Neural Networks: Current Framework and Trends

Fan Liu and Meng Joo Er
Nanyang Technological University
Singapore

1. Introduction

Fuzzy systems have been demonstrated their ability to solve different kinds of problems in classification, modeling control and in a considerable number of industry applications. It has been shown as a powerful methodology for dealing with imprecision and nonlinearity efficiently (Wang, 1994). However, one of the shortcomings of fuzzy logic is the lack of learning and adaptation capabilities. As we know, neural network (NN) is one of the important technologies towards realizing artificial intelligence and machine learning. Many types of neural networks with different learning algorithms have been designed and developed (Deng et al., 2002; Levin & Narendra, 1996; Narendra &Parathasarathy, 1998). Recently, there is an increasing interest to hybridize the approximate reasoning method of fuzzy systems with the learning capabilities of neural networks and evolutionary algorithms. Fuzzy neural network (FNN) system is one of the most successful and visible directions of that effort.

FNNs as hybrid systems have been proven to be able to reap the benefits of fuzzy logic and neural networks. In these hybrid systems, standard neural networks are designed to approximate a fuzzy inference system through the structure of neural networks while the parameters of the fuzzy system are modified by means of learning algorithms used in neural networks. One purpose of developing hybrid fuzzy neural networks is to create self-adaptive fuzzy rules for online identification of a singleton or Takagi-Sugeno-kang (TSK) type fuzzy model (Takagi & Sugeno, 1985) of a nonlinear time-varying complex system. The twin issues associated with a fuzzy system are 1) parameter estimation which involves determining parameters of premises and consequences and 2) structure identification which involves partitioning the input space and determining the number of fuzzy rules for a specific performance. FNN systems have been found to be very effective and of widespread use in several fields.

In recent years, the idea of self-organization has been introduced in hybrid systems to create adaptive models. Some adaptive approaches also have been introduced in FNNs whereby not only the weights but also the structure can be self-adaptive during the learning process (Er & Wu, 2002; Huang et al., 2004; Jang, 1993; Juang &Lin, 1998; Leng et al., 2004; Lin &Lee, 1996; Qiao & Wang, 2008).

The technology of FNNs combines the profound learning capability of neural networks with the mechanism of explicit and easily interpretable knowledge presentation provided by fuzzy logic. In a word, FNN is able to represent meaningful real-world concepts in

comprehensive knowledge bases. The typical approach of designing an FNN system is to build standard neural networks first, and then incorporate fuzzy logic in the structure of neural networks. The key idea is as follows: Assuming that some particular membership functions have been defined, we begin with a fixed number of rules by resorting to either trial-and-error methods or expert knowledge. Next, the parameters are modified by learning algorithm such as backpropagation (BP) algorithm (Siddique & Tokhi, 2001). The BP is a gradient descent search algorithm. It is based on minimization of the total mean squared error between the actual output and the desired output. This error is used to guide the search of the BP algorithm in the weight space. The BP is widely used in many applications in that it is not necessary to determine the exact structure and parameters of neural networks in advance. However, the problem of the BP algorithm is that it is often trapped in local minima and the learning speed is very slow in searching for global minimum of the search space. The speed and robustness of the BP algorithm are sensitive to several parameters of the algorithm and the best parameters vary from problems to problems. Therefore, many adjustment methods have been developed, notably evolutionary algorithms such as genetic algorithms (GAs) or particle swarm optimization (PSO). By working with a population of solutions, the GA can seek many local minima, and thus increase the likelihood of finding global minimum. This advantage of GA can be applied to neural networks to optimize the topology and parameters of weights. The key point is to employ an evolutionary learning process to automate the designing of the knowledge base, which can be considered as an optimization or search problem. The GA is used to optimize the parameters of neural networks (Seng et al., 1999; Siddique & Tokhi, 2001; Zhou & Er, 2008) or identify the optimal structure of neural networks (Chen et al., 1999; Tang et al., 1995,). Moreover, the adaptation of neural network parameters can be performed by different methods such as orthogonal least square (OLS) (Chen et al.,1991), recursive least square (RLS) (Leng et al., 2004), linear least square (LLS) (Er & Wu, 2002), extended Kalman filter (EKF) (Kadirkamanathan & Niranjan, 1993; Er et al., 2010) and so on.

The objective of this chapter is to develop FNNs by hybrid learning techniques so that these systems can be used for online identification, model and control nonlinear and time-varying complex systems. In this chapter, we propose two kinds of self-organizing FNN that attempt to combine fuzzy logic with neural network and apply these learning algorithms to solve several well-known benchmark problems such as static function and linear and nonlinear function approximation, Mackey-Glass time-series prediction and real-world benchmark regression prediction and so on.

The chapter is organized as follows. The general frame of self-organizing FNN is described in Section 2. The first learning algorithm combined FNN with EKF is presented in Section 3. It is simple and effective and is able to generate a FNN with high accuracy and compact structure. Furthermore, a novel neuron pruning algorithm based on optimal brain surgeon (OBS) for self-organizing FNN is described in Section 4 in detail. Simulation studies on several well-known benchmark problems and comparisons with other learning algorithms have been conducted in each section. The summary of FNN associated with conclusions and future work are discussed in Section 5.

2. General frame of self-organizing FNNs

The self-organizing fuzzy neural network system primarily implements TSK or TS type (Sugeno & Kang, 1988) fuzzy model. The general architecture is depicted in Fig. 1. This five-

layer self-organizing FNN implements a TSK type fuzzy system. Without loss of generality, we consider a multi-input-single-output (MISO) fuzzy model with input vector $X=(x_1,x_2,\ldots,x_r)$ and output variable y.

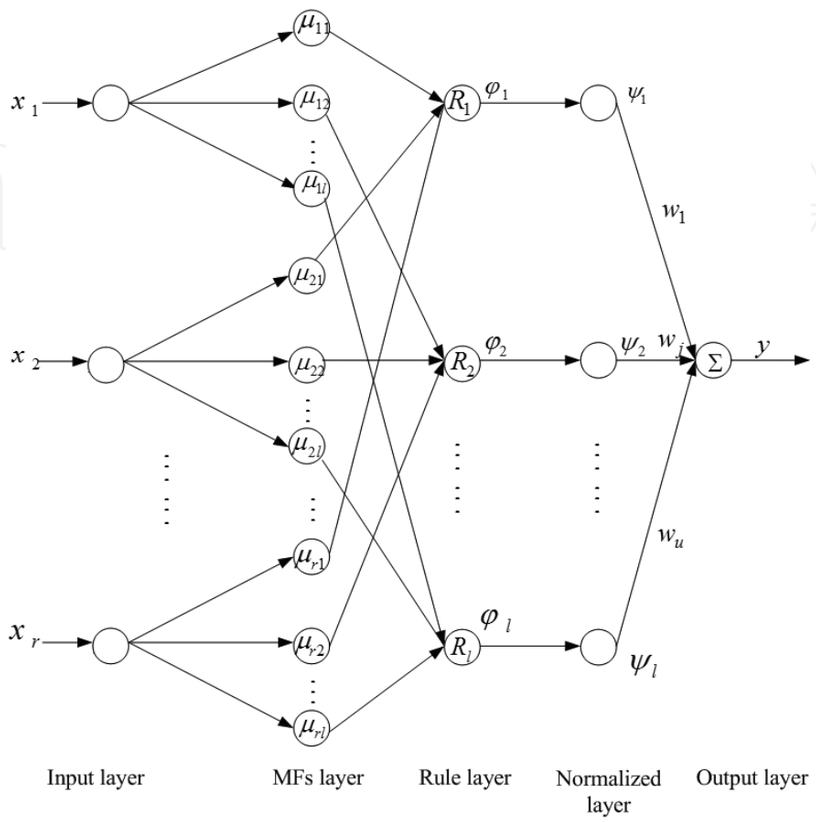

Fig. 1. General architecture of FNNs.

Layer 1 is an input layer and transmits values of input linguistic variable $x_i(i=1,2,\cdots r)$ to layer 2 directly, where r is the number of input variables. Each input variable x_i has l membership functions (MFs) $\mu_{ij}(i=1,2,\cdots r, j=1,2,\cdots,l)$ as shown in layer 2. The MFs can be triangular function (G. Leng et al., 2009) or Gaussian function (Wu and Er, 2004) or other functions. In this chapter, we choose Gaussian functions given by

$$\mu_{ij}(x_i)=\exp\left(-\frac{(x_i-c_{ij})^2}{\sigma_{ij}^2}\right) \quad i=1,2,\cdots,r, j=1,2,\cdots,l \tag{1}$$

where μ_{ij} is the *jth* membership function of the *ith* input variable x_i, c_{ij} and σ_{ij} are the center and width of the *jth* membership function with respect to the *ith* neuron, respectively. Layer 3 is the rule layer. Each node in this layer represents a possible IF-part of fuzzy rules. If the T-norm operator used to compute each rule's firing strength is multiplication, the output of the *jth* rule $R_j(j=1,2,\cdots l)$ is

$$\varphi_j(x_1,x_2,\ldots,x_r)=\exp\left(-\sum_{i=1}^{r}\frac{(x_i-c_{ij})^2}{\sigma_{ij}^2}\right) \quad j=1,2,\ldots,l\ . \tag{2}$$

Layer 4 is the normalized layer. The number of neurons in this layer is equal to that of layer 3. The output of the *jth* neuron in this layer is

$$\psi_j = \frac{\varphi_j}{\sum_{i=1}^{l} \varphi_i}, \quad j = 1,2,\cdots,l \tag{3}$$

Layer 5 is output layer and each node represents an output linguistic variable. The output of this layer is the weighted summation of incoming signals given by

$$y(x_1 \cdots x_r) = \sum_{j=1}^{l} w_j \psi_j \tag{4}$$

where y is the output variable and w_j is the THEN-part or connection weight of the *jth* rule. For the TSK model, weights are the polynomials of the input variables

$$w_j = A_j \cdot B = a_{j0} + a_{j1}x_1 + \cdots a_{jr}x_r \tag{5}$$

where A_j=[$\alpha_{j0}\alpha_{j1}\ldots\alpha_{jr}$] is the weight vector of input variables in *jth* rule and B=[1$x_1\ldots x_r$] is a column vector.
Suppose the *nth* data pair is arriving, and u RBF units are generated for the n training data pairs. The overall output of the FNN is

$$\mathrm{Y} = \mathrm{W}\Psi \tag{6}$$

where for the TSK model, W and Ψ are given by

$$W = [a_{10}a_{20}\cdots a_{u0}\ a_{11}a_{21}\cdots a_{u1}\ a_{r1}a_{r2}\cdots a_{ur}] \tag{7}$$

$$\Psi = \begin{bmatrix} \psi_{11} & \cdots & \psi_{1n} \\ \vdots & \vdots & \vdots \\ \psi_{u1} & \cdots & \psi_{un} \\ \psi_{11}x_{11} & \cdots & \psi_{1n}x_{1n} \\ \vdots & \vdots & \vdots \\ \psi_{u1}x_{11} & \cdots & \psi_{un}x_{1n} \\ \vdots & \vdots & \vdots \\ \psi_{11}x_{r1} & \cdots & \psi_{1n}x_{rn} \\ \vdots & \vdots & \vdots \\ \psi_{u1}x_{r1} & \cdots & \psi_{un}x_{rn} \end{bmatrix} \tag{8}$$

where ψ_{ij} is the output of the *ith* neuron in the normalized layer when *jth* training data arrives.

3. Self-constructing Fuzzy Neural Networks with Extended Kalman Filter (SFNNEKF)

In this section, a self-constructing fuzzy neural network employing extended Kalman filter (SFNNEKF) is designed and developed. The learning algorithm based on EKF is simple and effective and is able to generate a FNN with a high accuracy and compact structure. The

proposed algorithm comprises of three parts: (1) criteria of rule generation; (2) pruning technology and (3) adjustment of free parameters. The EKF algorithm is used to adjust the free parameters of the SFNNEKF. The performance of the SFNNEKF is compared with other learning algorithms in the tasks of function approximation, nonlinear system identification and time-series prediction. Simulation studies and comparisons with other algorithms demonstrate that a more compact structure with high performance can be achieved by the proposed algorithm.

3.1 Learning algorithm of SFNNEKF

3.1.1 Error Reduction Ratio (ERR)

Suppose that for n observations, the FNN has generated uRBF neurons. The output matrix of the hidden layer is given by (8) then (6) can be written as

$$T = \Phi W + E \tag{9}$$

where $T \in R^n$ is the desired output vector and E is the error vector, $\Phi = \Psi^T$ can be rewritten as follows:

$$\Phi = KA \tag{10}$$

where K is a $n \times u$ matrix with orthogonal columns and A is a $u \times u$ upper triangular matrix. Substituting (10) into (9), we obtain

$$T = KAW + E = KG + E \tag{11}$$

The orthogonal lease squares solution G is given by $G = (K^T K)^{-1} K^T T$ or equivalently

$$g_i = \frac{k_i^T T}{k_i^T k_i} \quad 1 \le i \le u \tag{12}$$

An error reduction ratio (ERR) due to k_i as defined in (Shen et al., 1991) is given by

$$err_i = \frac{g_i^2 k_i^T k_i}{T^T T} \quad 1 \le i \le u \tag{13}$$

Substituting (12) into (13) yields

$$err_i = \frac{(k_i^T T)^2}{k_i^T k_i T^T T} \quad 1 \le i \le u \tag{14}$$

The ERR offers a simple and effective way of seeking a subset of significant regressors. In this chapter, we use it as a growing criterion to evaluate the network generalization capability. We define

$$\eta = \sum_{i=1}^{u} err_i \tag{15}$$

If $\eta < k_{err}$, where k_{err} is a prespecified threshold, the FNN needs more hidden nodes to achieve good generalization performance and a neuron will be added to the network. Otherwise, no neurons will be added.

3.1.2 Criteria of growing neurons

For the ith observation(P_i,t_i), calculate the output error of the SFNNEKF, e_i, the distance $d_i(j)$ and the ERR as follows:

$$\|e_i\| = \|t_i - y_i\| \tag{16}$$

$$d_i(j) = \|P_i - C_j\|,\ j = 1,\cdots,u \tag{17}$$

$$\eta = \sum_{i=1}^{u} err_i \tag{18}$$

If

$$\|e_i\| > k_e, d_{\min} > k_d \ and \ \eta < k_{err} \tag{19}$$

where $d_{\min} = \arg\min(d_i(j))$, $j = 1,2,\cdots,u$, k_e, k_d are two predetermined parameters which are chosen as follows:

$$k_e = \max[e_{\max} \times \beta^i, e_{\min}] \quad 0 < \beta < 1 \tag{20}$$

$$k_d = \max[d_{\max} \times \gamma^i, d_{\min}] \quad 0 < \gamma < 1, \tag{21}$$

a new neuron is added to the SFNNEKF network and the center, width and the output layer weight for the newly generated neuron are set as follows:

$$C_i = P_i \tag{22}$$

$$w_i = e_i \tag{23}$$

$$\sigma_i = k \times d_{\min} \tag{24}$$

where k is an overlap factor that determines the overlap of responses of the RBF units.

3.1.3 Criteria of pruning neurons

To facilitate the following discussion, we rewrite the output matrix of the hidden layer, as follows:

$$\Phi = \begin{bmatrix} p_{11} & \cdots & p_{u1} \\ \vdots & \vdots & \vdots \\ p_{1n} & \cdots & p_{un} \end{bmatrix} \tag{25}$$

In order to evaluate the contribution of each hidden neuron to the network output, the root mean square error (RMSE) of each hidden node is calculated as follows:

$$h_i = \sqrt{\phi_i^T \phi_i / n}, i = 1,\cdots,u \tag{26}$$

where $\phi_i = [p_{i1},\cdots,p_{in}]^T$ is the column vector of matrix Φ. If $h_i < k_{rmse}$ where k_{rmse} is a prespecified threshold, the ith hidden neuron is inactive and should be deleted.

3.1.4 Adjustment of weights

When no neurons are added or pruned from the network, the network parameter vector W_{EKF} is adjusted using the EKF method (Kadirkamanathan & Niranjan, 1993)

$$W_{EKF}(n) = W_{EKF}(n-1) + e_n k_n \tag{27}$$

where $W_{EKF} = [w_1, C_1^T, \sigma_1, \cdots, w_u, C_u^T, \sigma_u]$ is the network parameter vector and k_n is the Kalman gain vector given by

$$k_n = [R_n + a_n^T P_{n-1} a_n]^{-1} P_{n-1} a_n \tag{28}$$

Here, a_n is the gradient vector and has the following form

$$\begin{aligned} a_n = [&\psi_1(X_n), \psi_1(X_n)(2w_1\sigma_1)(X_n - C_1)^T, \\ &\psi_1(X_n)(2w_1\sigma_1^3)\|X_n - C_1\|^2, \cdots, \\ &\psi_u(X_n), \psi_u(X_n)(2w_u\sigma_u)(X_n - C_u)^T, \\ &\psi_u(X_n)(2w_u\sigma_u^3)\|X_n - C_u\|^2]^T \end{aligned} \tag{29}$$

where R_n is the variance of the measurement noise and P_n is the error covariance matrix which is updated by

$$P_n = [I - k_n a_n^T] P_{n-1} + QI \tag{30}$$

where Q is a scalar that determines the allowed random step in the direction of gradient vector and I is the identity matrix. When a new neuron is allocated, the dimensionality of the P_n increases to

$$P_n = \begin{pmatrix} P_{n-1} & 0 \\ 0 & P_0 I \end{pmatrix} \tag{31}$$

where P_0 is an estimate of uncertainties in the initial values assigned to the parameters.

3.2 Simulation results

3.2.1 Function approximation

First, a very popular function, Hermite polynomial, is used to evaluate the performance of the proposed SFNNEKF. The function is given by

$$f(x) = 1.1(1 - x + 2x^2)\exp(-\frac{x^2}{2}) \tag{32}$$

Parameters of the SFNNEKF are set as follows: $d_{\max} = 1$, $d_{\min} = 0.2$, $e_{\max} = 0.8$, $e_{\min} = 0.01$, $k_{err} = 0.99$, $k_{rmse} = 0.005$, $k = 0.5$, $\beta = 0.97$, $\gamma = 0.97$, $P_0 = 1.1$ and $Q = 0.01$. A total of 200 training samples are randomly chosen from the interval [−4, 4] for all the methods. Fig. 1 shows the growth of hidden neurons and Fig. 2 shows the RMSE with respect to the training samples. A comparison of structure and performance with MRAN (Lu et al., 1997) and DFNN (Er & Wu, 2002) is summarized in Table 1. It can be seen that the DFNN algorithm achieves the

best RMSE performance in this case, but it needs more training time, because the TSK model makes the DFNN network structure more complicated. The SFNNEKF algorithm can obtain almost the same RMSE performance as the DFNN with faster learning speed and it has better performance than the MRAN algorithm in terms of training time and RMSE.

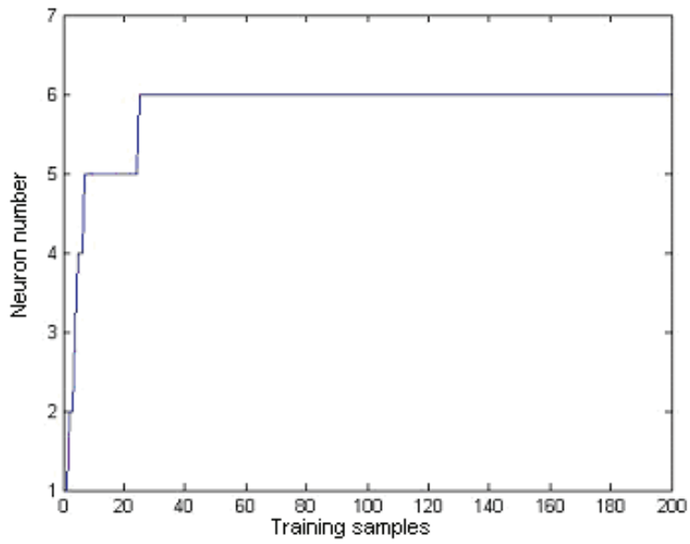

Fig. 1. Growth of neurons

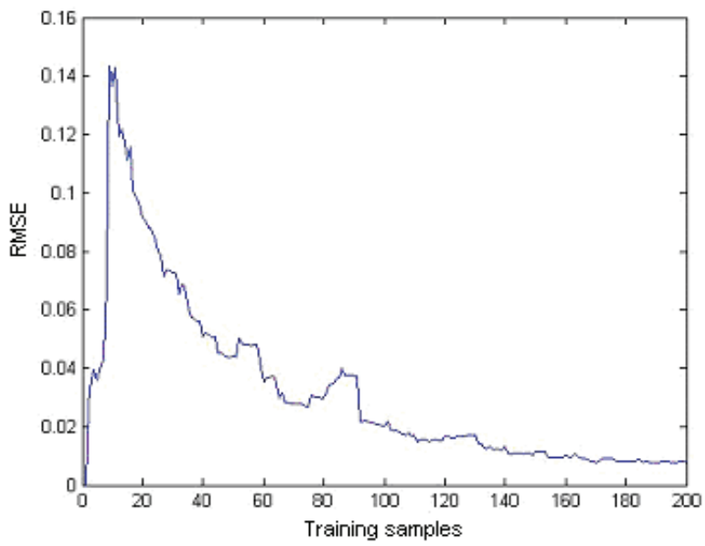

Fig. 2. RMSE during training

Algorithm	Number of neurons	RMSE	Training time(s)
SFNNEKF	6	0.0078	0.38
DFNN	6	0.0052	2.97
MRAN	7	0.0376	0.53

Table 1. Comparison of structure and performance of different algorithms (Example 3.2.1)

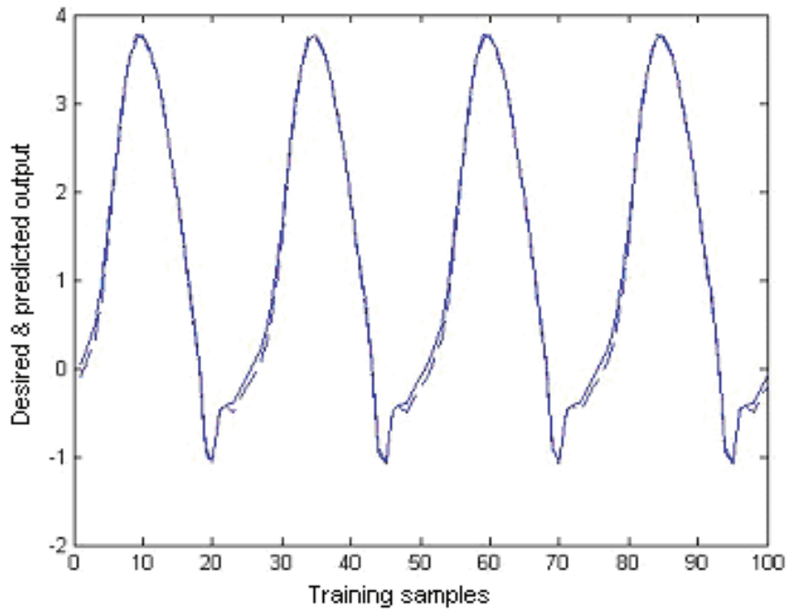

Fig. 3. Identification result: Desired (-) and Identified (---)

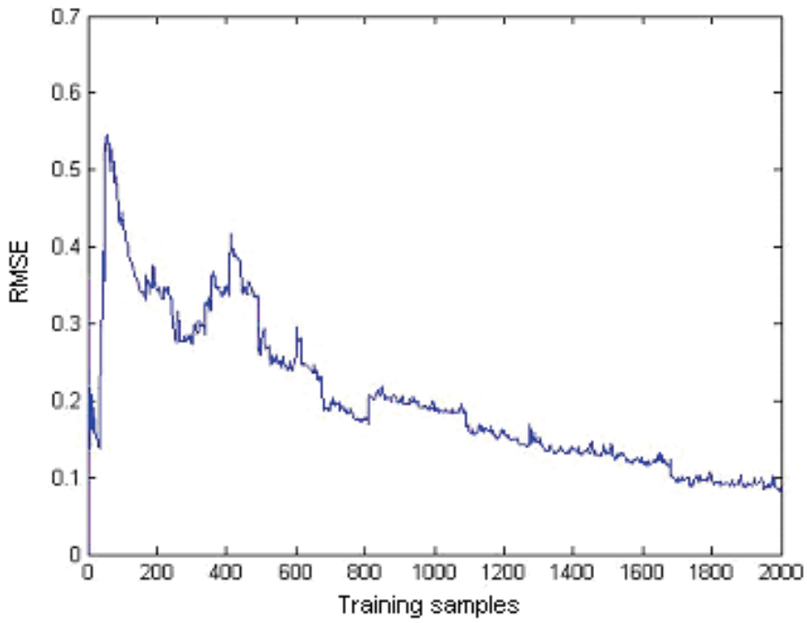

Fig. 4. RMSE during training

3.2.2 Nonlinear plant identification

The plant be identified is described by the second-order difference equation (Narendra & Parathasarathy, 1990)

$$y(t+1) = f[y(t), y(t-1)] + u(t) \tag{33}$$

where

$$f[y(t), y(t-1)] = \frac{y(t)y(t-1)[y(t)+2.5]}{1+y^2(t)+y^2(t-1)} \tag{34}$$

Parameters of the SFNNEKF are set as follows:
$d_{\max} = 1$, $d_{\min} = 0.2$, $e_{\max} = 0.8$, $e_{\min} = 0.01$, $k_{err} = 0.96$, $k_{rmse} = 0.005$, $k = 0.5$, $\beta = 0.97$, $\gamma = 0.97$, $P_0 = 1.1$, and $Q = 0.01$.
The input *u(k)* is assumed to be independent identically distributed (i.i.d) random signal uniformly distributed in the interval [−2, 2] and a total of 2000 samples are randomly chosen for the training process. After this, the sinusoidal input signal $u(t) = \sin(2\pi t / 25)$ is used to test identified model. Fig. 3 depicts the identified results using the SFNNEKF algorithm. In order to observe the result clearly, only the first 100 samples are shown in Fig. 3. Here, the solid-line curve is the desired plant output and the dash-line curve is for the identified model output. It can be seen that the SFNNEKF algorithm can identify the plant very well. The RMSE for training samples is shown in Fig. 4. The SFNNEKF algorithm generated a total of 15 neurons at the end of the training process. Table 2 shows a comparison of structure and performance of different algorithms. It is clear that the SFNNEKF algorithm can achieve a satisfactory RMSE performance with a simple network structure. Although the RMSE of the DFNN algorithm is 0.056, it needs more neurons (i.e. complicated network structure) to realize it. For the training speed, the SFNNEKF algorithm is the fastest method in all the three sequential learning algorithms. As a whole, the SFNNEKF algorithm demonstrates superior performance than the DFNN and MRAN algorithms in terms of networks complexity and training time.

Algorithm	Number of neurons	RMSE	Training time (s)
SFNNEKF	15	0.087	30.25
DFNN	26	0.056	608.51
MRAN	40	0.1525	69.79

Table 2. Comparison of structure and performance of different algorithms (Example 3.2.2)

3.2.3 Mackey-Glass time-series prediction

The Mackey-Glass time-series prediction is a benchmark problem which has been considered by a number of researchers (Lu et al., 1997; Wu & Er, 2000; Cho &Wang, 1996; Juang & Tin, 1998; Kadirkamanathan & Niranjan, 1993). The time series in our simulation is generated by

$$x(t+1) = (1-a)x(t) + \frac{bx(t-\tau)}{1+x^{10}(t-\tau)} \tag{35}$$

for a=0.1, b=0.2 and τ=17. A sampled version of the series is obtained by the above equation. The prediction model is given by

$$x(t+p) = f[x(t), x(t-\Delta t), x(t-2\Delta t), x(t-3\Delta t)] \tag{36}$$

For the purpose of training and testing, 6000 samples are generated between t=0 and t=6000 from (33) with initial conditions $x(t)$=0 for t<0 and x(0)=1.2. Hence, these data are used for preparing for the input and output pairs in the (34).

Suppose p=6, Δt=6, select first 500 sample pairs between 118≤t≤617 as training data. Parameters of the SFNNEKF are set as follows:

$d_{\max}=1$, $d_{\min}=0.18$, $e_{\max}=1.1$, $e_{\min}=0.02$, $k_{err}=1.2$, $k_{rmse}=0.00025$, $k=0.5$, $\beta=0.95$, $\gamma=0.98$, $P_0=1.1$, and $Q=0.01$

The results for n=500, p=6 are shown in Fig. 5 and Fig. 6.

Figures illustrate the simulation results of the SFNNEKF in the case that p is 6 and the number of rules is 34. It indicates that the SFNNEKF possesses remarkable generalization capability. If we selected p=50 and Δt=6, for the convenience of comparison, the generalization capability is evaluated by the normalized root mean squared error (NRMSE) or the non-dimensional error index (NDEI), which is defined as the RMSE divided by the standard deviation of the target series (Platt, 1991). Generalization comparisons between the SFNNEKF, DFNN (Er & Wu, 2002), RAN (Platt, 1991) RANEKF (Kadirkamanathan & Niranjan, 1993), and MRAN (Lu et al., 1997) is listed in table 3. It is shown that SFNNEKF can obtain better performance even it has generated more rules than DFNN. However, the SFNNEKF shows superiority compared with the RAN and RANEKF algorithms in terms of the performance of the NRMSE, which measures the average prediction performance. The final value of the NRMSE is 0.0327 for SFNNEKF with 41 rules compares with 0.071 for the RAN and 0.056 for the RANEKF.

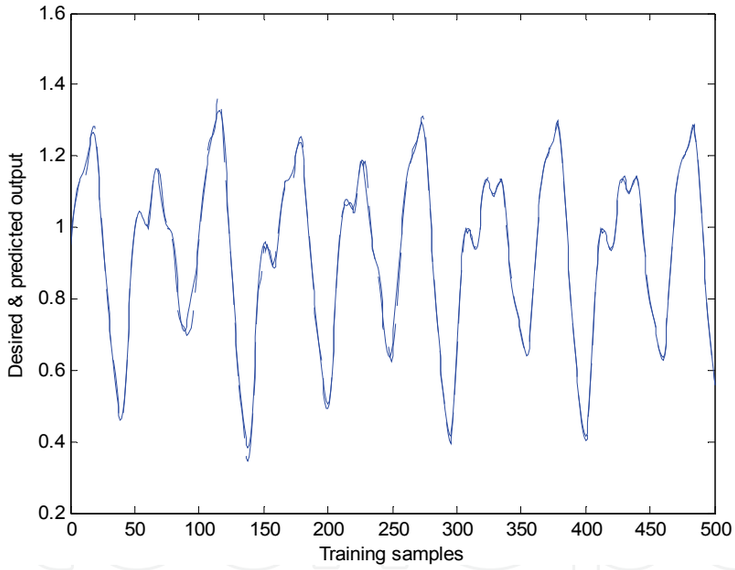

Fig. 5. Mackey-Glass time series from 642 to 1141 and six-step

Algorithm	Pattern length	Number of rules	NRMSE
SFNNEKF	2000	41	0.0327
DFNN	2000	25	0.0544
RAN	5000	50	0.071
RANEKF	5000	56	0.056
MRAN	5000	28	[a]

[a] The generalization error is measured by weighted prediction error (WPE) (example 3.2.3)

Table 3. Generalization comparison between SFNNEKF, DFNN, RAN, RANEKF and MRAN

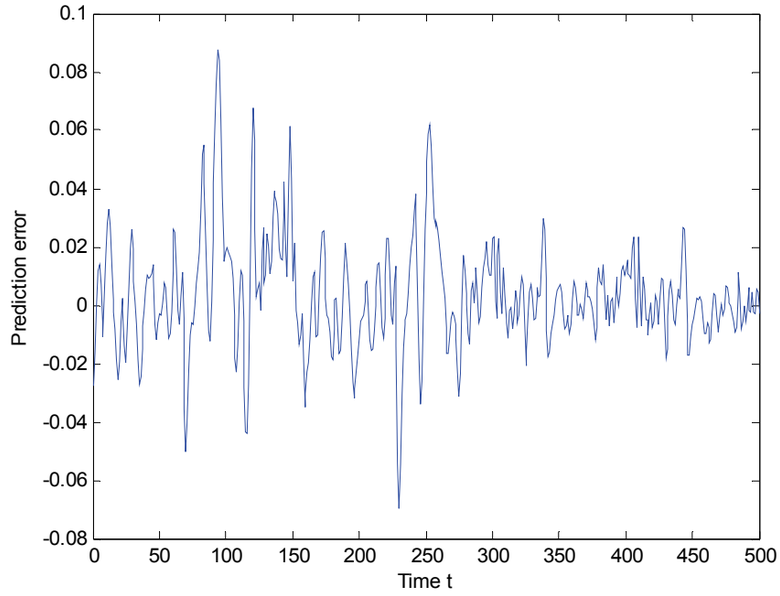

Fig. 6. Prediction error

3.3 Conclusions

A new self-constructing fuzzy neural network has been proposed in this part. The basic idea of the proposed approach is to construct a self-constructing fuzzy neural network based on criteria of generating and pruning neurons. The EKF algorithm has been used to adapt the consequent parameters when a hidden unit is not added. The superior performance of the SFNNEKF over some other learning algorithms has been demonstrated in three examples in this part. Simulation results show that a more effective fuzzy neural network with high accuracy and compact structure can be self-constructed by the proposed SFNNEKF algorithm.

4. A neuron pruning algorithm based on optimal brain surgeon for self-organizing fuzzy neural networks with parsimonious structure

In this section, a novel learning algorithm for creating a self-organizing FNN to implement TSK type fuzzy models is proposed. The optimal brain surgeon (OBS) (Hassibi & Stork, 1993) is employed as a neuron pruning mechanism to remove unimportant neurons directly during the training procedure. Distinguished from other pruning strategies based on the OBS, there is no need to calculate the inverse matrix of Hessian, we simplify the calculation by using LLS method to obtain the Hessian matrix. To acquire precision model, the LLS method is performed to obtain the consequent parameters of the network. The proposed algorithm has a parsimonious structure and is generated with high accuracy. The effectiveness of the proposed algorithm is demonstrated in several well-known benchmark problems such as static function approximation, two-input nonlinear function approximation and real-world non-uniform benchmark regression problems. Simulation studies are compared with other existing published algorithms. The results indicate that the proposed algorithm can provide comparable approximation and generalization performance with a more compact structure and higher accuracy.

4.1 Learning algorithm of the FNN

The learning procedure of the proposed self-organizing FNN comprises two stages. In the first stage, the structure of the FNN is generated based on the growth criteria and the proposed OBS-based pruning method. In the second stage, parameters of newly created neurons are assigned and consequent parameters of all existing neurons will be updated by the LLS method.

4.1.1 Structure design of the FNN

To determine the proper number of neurons or fuzzy rules of FNNs, we adopt two growth criteria to generate neurons. In order to obtain a compact structure, the OBS algorithm (Hassibi & Stork, 1993) is employed as a pruning criterion. An initial structure of the FNN is first constructed and the importance of each hidden neuron or fuzzy rule is evaluated by the OBS algorithm, the least important neuron will be deleted if the performance of the entire network is accepted after deleting this unimportant neuron. This procedure will repeat until the desired accuracy can be satisfied. We will describe the learning algorithm in detail in the sequel.
The learning stage is based on a data set composed by n input-output pairs:

$$P = \{(X_k, t_k\} = \{(x_{k1}, x_{k2}, \cdots, x_{kr}), t_k\}, k = 1, 2 \cdots, n \,. \tag{37}$$

When the *kth* observation (X_t,t_k) arrives, the overall FNN output of the existing structure using (2)-(4) is denoted by y_k, the system error can be defined by (16)
If

$$\|e_k\| > k_e \tag{38}$$

where k_e is a predefined error tolerance, a new fuzzy rule should be considered if other criteria of generation have been satisfied simultaneously. The term k_e decays during the learning process as follows

$$k_e = \begin{cases} e_{\max} & 1 < i < n/3 \\ \max[e_{\max} \times \beta^i, e_{\min}], & n/3 \le i \le 2n/3 \\ e_{\min} & 2n < i \le n \end{cases} \tag{39}$$

where $e_{\max}$ is the maximum error chosen, $e_{\min}$ is the desired accuracy of the FNN output and $\beta \in (0,1)$is a convergence constant.
The second growth criterion is the accommodation criterion. It can be described as follows: when the *kth* observation (X_t,t_k) arrives, calculate the distance d_{kj} between the observation X_k and the center C_j of the u existing neurons, i.e.

$$\|d_{kj}\| = \|X_k - C_j\|, k = 1, 2, \cdots, n, j = 1, 2, \cdots, u \,, \tag{40}$$

the minimum distance between the *kth* observation and the nearest center is described as

$$d_{\min} = \arg\min(d_{kj}), k = 1, 2, \cdots, n, j = 1, 2 \cdots, u \,. \tag{41}$$

If

$$d_{\min} > k_d \tag{42}$$

where k_d is determined by (21).
Here, $d_{\max}$ is the largest length of the input space, $d_{\min}$ is the smallest length of interest and $\gamma \in (0,1)$ is the decay constant. If these two growth criteria are satisfied simultaneously, a new hidden neuron will be generated.
When growth procedure is complete, a pruning algorithm is employed to optimize the structure of the FNN to obtain a compact network structure. For this algorithm, we use the OBS as a pruning strategy. The OBS algorithm uses the second-order derivative information of the error function to find the unimportant neuron to get a tradeoff between the network complexity and good performance of entire network. The basic idea of the OBS algorithm is described as follows. Firstly, we assume that the pruning algorithm is used after the training process has converged or is fully trained. It means that the network reaches a local minimum of the error surface. The functional Taylor series of the cost function E with respect to parameters w is given by

$$\Delta E = \left(\frac{\partial E}{\partial w}\right)^T \cdot \Delta w + \frac{1}{2}\Delta w^T \cdot \frac{\partial^2 E}{\partial w^2} \cdot \Delta w + O(\|\Delta w\|^3) \tag{43}$$

where $H = \frac{\partial^2 E}{\partial w^2}$ is the Hessian matrix which contains all second-order derivative information. For a network trained to a local minimum in error,

$$\left(\frac{\partial E}{\partial w}\right)^T \cdot \Delta w = 0 \tag{44}$$

and we also ignore the third and all higher order terms. The goal is to set one of the parameters w_q to zero to minimize the increase in error given by (41). Eliminating w_q can be expressed as

$$\Delta w_q + w_q = 0 \ . \tag{45}$$

Then the optimal model of OBS can be formulated as an optimization problem as follows

$$\begin{aligned} \min \quad & \frac{1}{2}\Delta w^T H \Delta w \\ s.t. \quad & I_q^T \Delta w + w_q = 0 \end{aligned} \tag{46}$$

where I_q is the unt vector in parameter space corresponding to parameter w_q.
Solving the constrained problem using the Lagrangian method

$$L = \frac{1}{2}\Delta w^T H \Delta w + \lambda (I_q^T \Delta w + w_q) \tag{47}$$

where λ is a Lagrange undetermined multiplier. After taking functional derivatives, employ the constraints and obtain resulting change in error are

$$\Delta w = -\frac{w_q}{[H^{-1}]_{qq}} H^{-1} \cdot I_q \tag{48}$$

$$L = \frac{1}{2}\frac{w_q^2}{[H^{-1}]_{qq}} \tag{49}$$

where H^{-1} is the inverse of the Hessian matrix H and $[H^{-1}]_{qq}$ is the qth element of H^{-1}.

Note that the OBS algorithm needs to calculate the inverse of the Hessian matrix. In fact, for very small dimension of the Hessian matrix, its inverse can be obtained by standard matrix methods while inverting a matrix of hundreds or thousands of terms seems computationally intractable. For our algorithm, we use the LLS method to avoid calculating the inverse of the Hessian matrix. The cost function used in our algorithm is defined as the training error $E_t(\Theta)$

$$E_t(\Theta) = \frac{1}{2}\sum_{k=1}^{n}[t(k) - \phi^T(k)\Theta]^2 \tag{50}$$

where $\Theta = W^T$ is the parameter vector, $t(k)$ is the desired output of the FNN and $\phi^T(k) = [\phi_1(k)\ \phi_2(k)\cdots\phi_{u\times(r+1)}(k)]$, $k = 1, 2\cdots, n$.

The functional Taylor series of the cost function of the FNN with respect to parameter vector Θ is

$$\Delta E_t(\Theta) = \left(\frac{\partial E_t(\Theta)}{\partial\Theta}\right)^T \cdot \Delta\Theta + \frac{1}{2}\Delta\Theta^T \cdot \frac{\partial^2 E_t(\Theta)}{\partial\Theta^2} \cdot \Delta\Theta + O(\|\Delta\Theta\|^3) \tag{51}$$

where $H = \dfrac{\partial^2 E_t(\Theta)}{\partial\Theta^2}$ is the Hessian matrix. For a network trained to a local minimum in error,

$$\left(\frac{\partial E_t(\Theta)}{\partial\Theta}\right)^T \cdot \Delta\Theta = 0 \tag{52}$$

and the third and all higher order terms are ignored. Then (49) can be rewritten as

$$\Delta E_t(\Theta) \approx \frac{1}{2}\Delta\Theta^T \cdot \frac{\partial^2 E_t(\Theta)}{\partial\Theta^2} \cdot \Delta\Theta = \frac{1}{2}\Delta\Theta^T H \Delta\Theta \tag{53}$$

In the following part, we will discuss how to use the Hessian matrix H to measure the increase in cost function $E_t(\Theta)$ due to deleting some parameters and then present the pruning algorithm. From (11) and (53) the Hessian matrix can be rewritten as

$$H = \frac{\partial^2 E_t(\Theta)}{\partial\Theta^2} = \Phi^T\Phi \tag{54}$$

After obtaining the Hessian matrix H, we can calculate the increase in cost function $E_t(\Theta)$ due to a change in parameters Θ. Since deleting the ith parameter in Θ is equivalent to setting the value of the corresponding parameter to zero, then the change in $E_t(\Theta)$ due to the removal of a subset G of parameters is

$$\Delta E_t(\Theta) \approx \frac{1}{2}\Delta\Theta_G^T H \Delta\Theta_G \tag{55}$$

where

$$[\Delta\Theta]_i = \begin{cases} [\Theta]_i & if\ i \in G \\ 0 & if\ i \notin G \end{cases} \tag{56}$$

where $[\Theta]_i$ denotes ith element of Θ .

After all, the importance of each hidden neuron or fuzzy rule can be evaluated by the change in cost function or training squared error $\Delta E_t(\Theta)$, the smaller the value of the change in the cost function, the less important is the neuron. Based on these descriptions, the proposed OBS-based pruning algorithm can be summarized as follows

1. Calculate the output of the existing FNN after growing all coming data.
2. According (50, calculate the RMSE for training, $E_t(\Theta)$.
3. Compute the Hessian matrix H according to (54).
4. Evaluate the importance of each hidden neuron in the network: $\Delta E_t(\Theta_i) = \frac{1}{2}\Delta\Theta_i^T H \Delta\Theta_i$ for $i = 1, 2\cdots, M$ where M is the number of parameters.
5. Define the tolerance limit of the change in the training for the RMSE as λE_{RMSE}, where $\lambda$is a predefined constant and $0<\lambda<1$.
6. Define the expected RMSE for training as k_{RMSE}. In general, k_{RMSE} is a very small positive value.
7. Find the smallest value of change in cost function

$$[\Delta E_{t\min}, ind_{\min}] = \min_i \Delta E_t(\Theta_i), i = 1, 2, \cdots, M \tag{57}$$

   where $\Delta E_{t\min}$ denotes the smallest value of $\Delta E_t(\Theta)$ and $ind_{\min}$ denotes the index of the smallest value.
8. Calculate the RMSE after deleting the least importance hidden neuron E_{RMSE}^D , choose $E^D = \max(\lambda E_{RMSE}, k_{RMSE})$, if $E_{RMSE}^D < E^D$, remove this neuron, and go to step 7) for the second least important neuron, and so on. Otherwise, do not remove any neurons and proceed to the weight adjustment stage.

Therefore, the structure (number of hidden neurons or fuzzy rules) of FNN is obtained based on the growth and pruning criteria.

4.1.2 Update of parameters and weight adjustment

After the network structure is established, the network enters the second stage: parameters of newly created neurons are assigned and the consequent parameters of all existing neurons will be updated by the simple but efficient LLS method.

For the first observation (X_1,t_1) arrives, since there are no hidden neurons at the beginning, some initializations are conducted. Parameters of the first neuron are assigned as follows

$$C_1 = X_1 \tag{58}$$

$$\sigma_1 = width0 \tag{59}$$

$$w_1 = t_1 \tag{60}$$

where C_1 and σ_1 is the center and width of the first neuron respectively, and w_1 is the connective weight of the first neuron.
The objective of weight adjustment is as follows

$$Y = W\Psi \tag{61}$$

$$\tilde{E} = \|T - Y\| \tag{62}$$

We try to find an optimal weight vector $W^* \in R^{u\times(r+1)}$ such that the error energy $\tilde{E}^T\tilde{E}$ is minimized. By using the LLS method, we can find the optimal W^* is in the form of

$$W^* = T\Psi^+ = T(\Psi^T\Psi)^{-1}\Psi^T \tag{63}$$

Since the structure of our FNN is very compact after structure learning process, the dimension of W and Ψ is small, the LLS method provides a computationally simple and efficient weight adjustment process.

4.2 Illustrative examples

In this section, the effectiveness of the proposed algorithm is demonstrated in three kinds of well-known benchmark examples: static function approximation, two-input nonlinear approximation and real-world non-uniform benchmark regression problems (Boston housing, auto-mpg and abalone) (Frank & Asuncion, 2010). The simulation results are also compared with other algorithms such as the OLS (Chen et al., 1991), RAN (Platt, 1991), RANEKF (Kadirkamanathan & Niranjan, 1993), MRAN (Lu et al., 1997), GAP-RBF(Huang et al., 2004), OS-ELM (Liang et al., 2006) GDFNN (Wu et al., 2001), SOFNN (Leng et al., 2005), FAOS-PFNN(Wang et al., 2009).

4.2.1 Static function approximation

The first example is on static function approximation. The underlying function to be approximated is a three-input nonlinear function *f*(*x*,*y*,*z*) which is widely used to verify the effectiveness of proposed algorithms of Khayat et al., 2009; Wu et al., 2001; Qiao & Wang, 2008; Chen et al., 1991.

$$f(x,y,z) = (1 + x^{0.5} + y^{-1} + z^{-1.5})^2 \tag{64}$$

A total of 216 uniformly training data are randomly sampled from the input ranges $[1,6]\times[1,6]\times[1,6]$ and the corresponding target data. Another 125 uniformly testing data are randomly selected from the same operating range to check the generalization of the proposed FNN. The parameters are selected as follows: $e_{\max} = 5$, $e_{\min} = 0.8$, $d_{\max} = 0.83$, $d_{\min} = 0.47$, $\beta = 0.97$, $\gamma = 0.99$, $\lambda = 0.8$, $width0 = 2$ and $k_{RMSE} = 0.3$.
To be in conformity with other learning algorithms, the same performance index, termed average percentage errors (APE) is used

$$APE = \frac{1}{n}\sum_{k=1}^{n}\frac{|t(k) - y(k)|}{|t(k)|}\times 100\% \tag{65}$$

where *n* is the number of data pairs, *t(k)* and *y(k)* are the *kth* desired output and actual output of the FNN, respectively.

Simulation results are shown in Fig. 7, where we see that a four-hidden-neuron structure is generated. The RMSE curving of training data with 216 epochs is shown in Fig. 8. The input variables x,y and z define four membership functions respectively. As we can see from the resulting figures, the trained network obtains a high precision and the RMSE is also confined to a very small value.

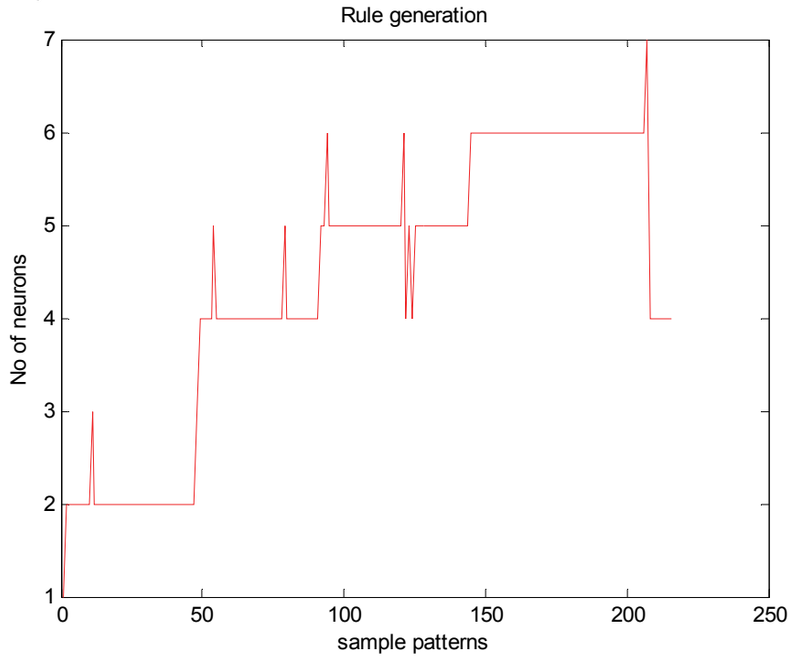

Fig. 7. Growth of fuzzy rules

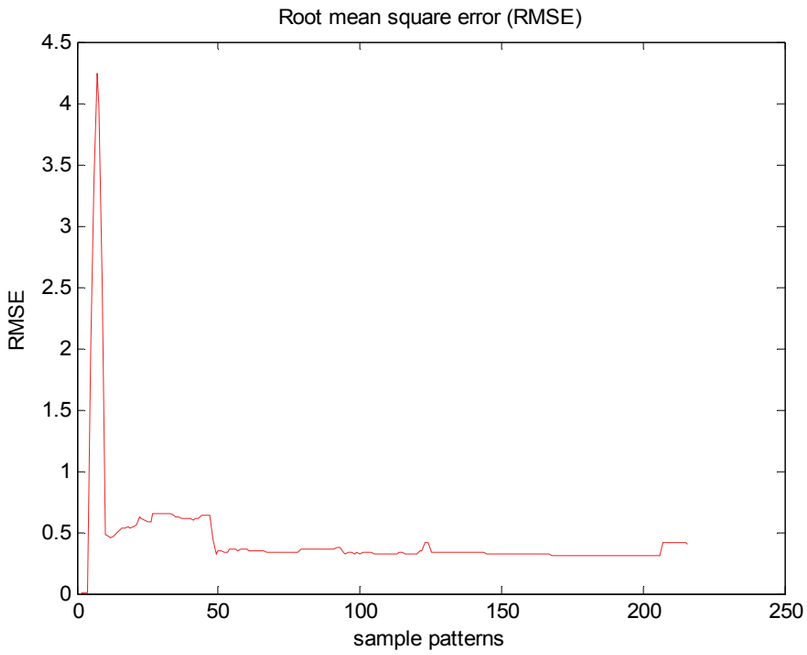

Fig. 8. Root mean squared error (RMSE) during training

Model	APE_{trn} %	APE_{chk} %	Parameter number	Training set size	Testing set size
OLS	2.43	2.56	66	216	125
GDFNN	2.11	1.54	64	216	125
SOFNN	1.1380	1.1244	60	216	125
The proposed algorithm	2.76	1.55	40	216	125

Table 4. Comparisons of the Proposed Algorithm with Other Methods (Static Function Approximation) (example 4.2.1)

Comparative studies of the proposed algorithm with other published works, such as the OLS (Chen et al., 1991), GDFNN (Wu et al., 2001) and SOFNN (Leng et al., 2005) are shown in Table 4. APE_{trn} % and APE_{chk} % are APE of training data and testing data, respectively. As it can be seen from Table 4, the proposed algorithm has the least number of parameters which means that it achieves the most compact structure compared with other algorithms while the approximation performance is worse than that of the SOFNN and GDFNN and nearly the same as that of the OLS. It should be noted that the proposed algorithm has a satisfactory generalization performance as its APE of testing is only 1.55 which is less than OLS and almost the same as that of the GDFNN. In brief, the proposed algorithm achieves the most parsimonious structure with good accuracy.

4.2.2 Two-input nonlinear sinc function

This example is used to demonstrate the performance of the proposed algorithm in noisy and noise-free environments. This function is also used in Leng et al., 2004 and Leng et al., 2009. It is defined as follows

$$z = \sin c(x, y) = \frac{\sin(x)\sin(y)}{xy}, x \in [-10, 10], y \in [-10, 10] \tag{66}$$

A total of 121 two-input data without noise and the corresponding target data are chosen as training data. Another set of uniformly sampled 121 input-target data are selected as testing data. The training parameters are selected as follows: $e_{\max} = 1.1$, $e_{\min} = 0.02$, $d_{\max} = 2$, $d_{\min} = 0.2$, $\beta = 0.9$, $\gamma = 0.97$, $\lambda = 0.8$, $width0 = 2$ and $k_{RMSE} = 0.005$.

A total of 5 hidden neurons (i.e. fuzzy rules) FNN is generated. The number of membership functions associated inputs x and y is five. When the training data is without noise, the RMSE for training is 0.0073 whilst the RMSE for testing is 0.0057. In order to determine the affect of noise, the training data are mixed with Gaussian white noise sequences which have zero mean and different variances as shown in Table 5.

Variances σ^2	NUMBER OF Neurons	NUMBER OF MFS	RMSE for training	RMSE for testing
$\sigma = 0$	5	5,5	0.0073	0.0057
$\sigma = 0.01$	5	5,5	0.0072	0.0056
$\sigma = 0.05$	6	6,6	0.0061	0.0048
$\sigma = 0.1$	7	7,7	0.0063	0.0050

Table 5. Performance of two-input sinc function with noise (example 4.2.2)

It can be observed from Table 5, the number of hidden neurons increases with the level of noise. That is because more neurons will be required to filter the noise. Both the RMSE for training and testing are small values though the data mixed with high level of noise. It means that the proposed algorithm is robust even when the training data is perturbed by noise. It demonstrates that our algorithm is suitable for noisy environment.

4.2.3 Boston housing prediction

The Boston housing prediction problem is to predict the median value of owner-occupied homes in suburbs of Boston. The database has 506 observations and each observation consists of 13 inputs (12 continuous inputs and one binary-valued input) and one continuous output (median value of owner-occupied homes). The 13 inputs and the output are normalized to the range [0, 1]. For this problem, 481 training data and 25 testing data are randomly generated from the Boston housing data set in each trial of simulation studies. The training parameters are selected as follows: $e_{\max} = 5$, $e_{\min} = 0.8$, $d_{\max} = 0.83$, $d_{\min} = 0.47$, $\beta = 0.99$, $\gamma = 0.99$, $\lambda = 0.8$, $width0 = 2$ and $k_{RMSE} = 0.05$. Performance comparisons with other algorithms are shown in Table 6.

Model	NUMBER OF Neurons	RMSE FOR Training	RMSE for testing	Training set size	Testing set size
RAN	18.8	0.3449	0.3432	481	25
RANEKF	19.98	0.1328	0.1437	481	25
MRAN	13.58	0.1440	0.1356	481	25
GAP-RBF	3.5	0.1507	0.1418	481	25
The proposed algorithm	2	0.0916	0.0583	481	25

Table 6. Comparisons of the proposed algorithm with other methods (Boston housing)

As observed from Table 6, the proposed algorithm obtains the smallest RMSE for training and testing, which means that the approximation and generalization performance is better than that of other learning algorithms. It is highlighted that the average number of hidden neurons generated by the system is only 2, which is less than that of other learning algorithms. Therefore, the proposed algorithm has the most compact structure of all the learning algorithms. In other words, our algorithm needs the least number of neurons to conduct the Boston housing prediction with comparable accuracy simultaneously.

4.2.4 Auto-mpg prediction

The auto-mpg problem is to predict the fuel consumption (miles per gallon) of different models of cars based on the displacement, horsepower, weight and acceleration of cars. A total of 392 observations are collected for the prediction problem. Each observation consists of seven inputs (four continuous inputs: displacement, horsepower, weight, acceleration, and three discrete inputs: cylinders, model year and origin) and one continuous output (the fuel consumption). For simplicity, the seven input attributes and one output have been normalized to the range [0, 1]. A total of 320 training data and 72 testing data are randomly chosen from the auto-mpg data set in each trial of simulation studies. The training parameters are selected as the previous Boston housing problem except the expected RMSE for training set is as k_{RMSF}=0.008. Performance comparisons among RAN (Platt, 1991),

RANEKF (Kadirkamanathan & Niranjan, 1993), MRAN (Lu et al., 1997), GAP-RBF (Huang et al., 2004), OS-ELM (Liang et al., 2006), FAOS-PFNN (Wang et al., 2009) and our proposed algorithm are presented in Table 7. It can be seen that the RMSE for testing our algorithm is the smallest among these learning algorithms while the RMSE for training is less than other algorithms except the FAOS-PFNN. It can be summarized that our proposed algorithm has the best generalization ability of all the learning algorithms though the approximation performance is a little worse than the FAOS-PFNN. Moreover, the number of fuzzy rules of our algorithm is only 2 and is less than other algorithms so that it has the most parsimonious structure among these learning algorithms. In brief, the overall performance of our algorithm is superior to other algorithms.

Model	NUMBER OF Neurons	RMSE FOR Training	RMSE for testing	Training set size	Testing set size
RAN	4.44	0.2923	0.3080	320	72
RANEKF	5.14	0.1088	0.1387	320	72
MRAN	4.46	0.1086	0.1376	320	72
GAP-RBF	3.12	0.1144	0.1404	320	72
FAOS-PFNN	2.9	0.0321	0.0775	320	72
OS-ELM	25	0.0696	0.0759	320	72
The proposed algorithm	2	0.0646	0.0513	320	72

Table 7. Comparisons of the proposed algorithm with other methods (Auto-mpg)

4.2.5 Abalone age prediction

The abalone problem has 4177 cases predicting the age of abalone from physical measurements. Each observation consists of 8 continuous inputs and 1 integral output. Similar to the age-auto prediction problem, the 8 inputs and 1 output are normalized to the range [1, 0]. A total of 3000 training data and 1177 testing data are randomly generated from the abalone data set in each trial of simulation studies. The training parameters are set as the same of the previous Boston housing problem.

Tab le 8 presents the performance comparisons of our algorithm with other learning algorithms. It is clear from the table that the RMSE for testing the proposed algorithm is the smallest of all algorithms. It means that our algorithm has the best generalization performance and obtained the smallest network size.

Model	NUMBER OF Neurons	RMSE FOR Training	RMSE for testing	Training set size	Testing set size
RAN	345.58	0.0931	0.0978	3000	1177
RANEKF	409	0.0738	0.0794	3000	1177
MRAN	87.571	0.0836	0.0837	3000	1177
GAP-RBF	23.62	0.0963	0.0972	3000	1177
FAOS-PFNN	4.54	0.0311	0.0807	3000	1177
OS-ELM	25	0.0761	0.0770	3000	1177
The proposed algorithm	2	0.0746	0.0735	3000	1177

Table 8. Comparisons of the proposed algorithm with other methods (Abalone)

4.3 Discussions

In this part a novel learning algorithm for creating a self-organizing fuzzy neural network (FNN) to implement the TSK type fuzzy model with a parsimonious structure was proposed. The OBS is employed as a pruning algorithm to remove unimportant neurons directly during the training process. Apart from other pruning strategies based on the OBS, there is no need to calculate the inverse matrix of Hessian; we simplify the calculation by using the LLS method to obtain the Hessian matrix.

The effectiveness of the proposed algorithm has been demonstrated in four well-known benchmark problems: namely static function approximation, nonlinear dynamic system identification, two-input nonlinear function and real-world non-uniform benchmark problems. Moreover, performance comparisons with other learning algorithms have also been presented in this part. The results indicate that the proposed algorithm can provide comparable approximation and generalization performance with a more compact parsimonious structure and higher accuracy.

5. Conclusions and future work

In this chapter, the development of fuzzy neural networks has been reviewed and the main issues for designing fuzzy neural networks including growing and pruning criteria and different adjustment methods of consequent parameters have been discussed. The general frame of fuzzy neural networks based on radial basis function neural networks has been described in Section 2. Two self-organization FNNs have been developed. For the first FNN, the SFNNEKF algorithm employs ERR as a generation condition in constructing the network which makes the growth of neurons smooth and fast. The EKF algorithm has been used to adjust free parameters of the FNN to achieve an optimal solution. Simulation results show that a more effective fuzzy neural network with high accuracy and compact structure can be self-constructed by the proposed SFNNEKF algorithm. For the second FNN, it is composed of two stages: the structure identification stage and the parameter adjustment stage. The structure identification consists of constructive and pruning procedures. An initial structure starts with no hidden neurons or fuzzy rule sets and grows neurons based on the criteria of neuron generation. Then the OBS is employed as a pruning strategy to further optimal the obtained initial structure. At last, the well-known LLS method is adopted to tune the free parameters in the parameter adjustment stage for sequentially arriving training data pairs. Simulation studies are compared with other algorithms. The simulation results indicate that the proposed algorithm can provide comparable approximation and generalization performance with a more compact structure and higher accuracy.

In a word, fuzzy neural networks are hybrid systems that combine the advantages of fuzzy logic and neural networks, there existed many kinds of FNN developed by researchers. Recently, the idea of self-organizing has been introduced in FNN. The purpose is to develop self-organizing fuzzy neural network systems to approximate fuzzy inference through the structure of neural networks to create adaptive models, mainly for approximate linear and nonlinear and time-varying systems. FNNs have been widely used in many fields. For our future work, studies will focus on the structure learning since appropriate number of fuzzy rules or find proper network architecture and developing optimal parameter adjustment methods.

6. References

Chen, S.; Cowan, C. F. N. & Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function network, *IEEE Transactions on Neural Networks*, vol. 2, pp. 302-309

Chen, S.; Wu, Y. & Luk, B. L.(1999). Combined genetic algorithm optimization and regularized orthogonal least squares learning for radial basis function networks, *IEEE Trans. Neural Networks*, vol. 10, no. 5, pp. 1239-1243

Cho, K. B. & Wang, B. H.(1996). Radial basis function based adaptive fuzzy systems and their applications to system identification and prediction, *Fuzzy Sets and Systems*, vol. 83, no. 3, pp. 325-339

Deng, J.; Sundararajan, N. & Saratchandran, P.(2002). Communication channel equalization using complex –valued minimal radial basis function neural networks, *IEEE Transactions on Neural Networks*, vol.13, no. 3, pp. 687- 696

Er, M. J. & Wu, S. Q. (2002) A fast learning algorithm for parsimonious fuzzy neural systems, *Fuzzy Sets and Systems*, vol. 126, pp. 337-351

Er, M. J.; Liu, F & Li, M. B. (2010). Self-constructing fuzzy neural networks with extended Kalman filter, *International Journal of Fuzzy Systems*, vol. 12, no. 1, pp. 66-72

Frank, A. & Asuncion, A. (2010), UCI Machine Learning Repository [http://archive.ics.uci.edu/ml], Irvine, CA: University of California, School of Information and Computer Science

Hassibi, B. & Stork, D. G.(1993). Second order derivatives for network pruning: optimal brain surgeon, in: *Advances in Neural Information Processing Systems,* Morgan Kaufman, San Mateo, CA, pp. 164-171

Huang, G. B.; Saratchandran, P. & Sundararajan, N.(2004). An efficient sequential learning algorithm for growing and pruning (GAP-RBF) networks, *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 34, pp. 2284-2292

Jang, J. S. R. (1993). ANFIS: Adaptive-Network-Based Fuzzy Inference System, *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 23, no. 3, pp. 665-684

Juang, C. F. & Lin, C. T.(1998). An on-line self-constructing neural fuzzy inference network and its applications, *IEEE Trans. Fuzzy Syst.* 6 pp.12–32

Kadirkamanathan, V & Niranjan, M.(1993). A function estimation approach to sequential learning with neural networks, *Neural Computation*, vol. 5, no. 6, pp. 954-975

Khayat, O.; Ebadzadeh, M. M.; Shahdoosti, H. R.; Rajaei R. & hajehnasiri, I. K.(2009). A novel hybrid algorithm for creating self-organizing fuzzy neural networks, *Neurocomputing*, 73(1-3) pp. 517-524

Leng, G.; Prasad, G. & McGinnity, T. M.(2004). An on-line algorithm for creating self-organizing fuzzy neural networks, *Neural Networks*, vol. 17, pp. 1477-1493

Leng, G.; McGinnity, T.M. & Prasad, G.(2005). An approach for on-line extraction of fuzzy rules using a self-organizing fuzzy neural network, *Fuzzy Sets and Syst.* 150, pp.211-243.

Leng, G.; Zeng, X. J. & Keane, J. A .(2009). A hybrid learning algorithm with a similarity-based pruning strategy for self-adaptive neuro-fuzzy systems, *Applied Softing Computing*, 9, pp. 1354-1366

Levin, A.U.& Narendra, K. S. (1996). Control of nonlinear dynamical system using neural networks-part II: Observability, identification, and control, *IEEE Transactions on Neural Networks,* vol. 7, no. 1, pp. 30-42

Liang, N. Y.; Huang, G. B.; Saratchandran, P. & Sundararajan, N.(2006). A fast and accurate online sequential learning algorithm for feedforward networks, *IEEE Trans. Neural Networks* 17, pp.1411-1423

Lin, C. T. & Lee, C. S. G.(1996). *Neural Fuzzy Systems: a Neural-Fuzzy Synergism to Intelligent Systems*, Englewood Cliffs, NJ: Prentice-Hall

Lu, Y.; Sundararajan, N. & Saratchandran, P.(1997). A sequential learning scheme for function approximation using minimal radial basis function neural networks, *Neural Computation*, vol. 9, no. 2, pp. 461-478

Narendra, K. S. & Parathasarathy, K. (1990). Identification and control of dynamic systems using neural networks, *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4-27

Platt, J. (1991). A resource-allocating network for function interpolation, *Neural Computation*, vol. 3, no. 2, pp. 213-225

Qiao, J. F. & Wang, H. D.(2008). A self-organizing fuzzy neural network and its applications to function approximation and forecast modeling, *Neurocomputing*, 71, pp. 564-569

Seng, T. L.; Khalid, M. B. & Yusof, R.(1999). Tuning of a neuro-fuzzy controller by genetic algorithm, *IEEE Trans. Syst., Man, Cyhern. B, Cybern.*, vol. 9, no. 2, pp. 226-236

Siddique, M. N. H. & Tokhi, M. O. (2001). Training Neural Networks: Backpropagation vs Genetic Algorithms, *Proceeding of the International Joint Conference on Neural Networks 4*, pp. 2673-2678

Sugeno, M. & Kang, G. T.(1988). Structure identification of fuzzy model, *Fuzzy Sets and Systems*, 28, pp. 15-33

Takagi, T. & Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control, *IEEE Transactions on Sytems, Man, and Cybernetics* 15 pp. 116-132

Tang, K. S.; Chan, C. Y.; Man, K. F.; & Kwong, S. (1995). Genetic Structure for NN Topology and Weights Optimization, *IEE Conference Publication*, No. 414, pp.250-255

Wang, L.X. (1994). *Adaptive Fuzzy Systems and Control: Design and Stability Analysis*, Englewood Cliffs, NJ: Prentice-Hall

Wang, N.; Er, M. J. & Meng, X. Y. (2009). A fast and accurate online self-organizing scheme for parsimonious fuzzy neural networks, *Neurocomputing*, 72, pp. 3818-3829

Wu S. Q. & Er, M. J.(2000). Dynamic fuzzy neural networks - a novel approach to function approximation, *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics,* vol. 30, no. 2, pp. 358-364

Wu, S. Q.; Er, M. J. & Gao, Y. (2001). A fast approach for automatic generation of fuzzy rules by generalized dynamic fuzzy neural networks, *IEEE Trans. Fuzzy Syst.* 9 pp. 578-594.

Zhou, Y. & Er, M. J. (2008). An evolutionary approach toward dynamic self-generated fuzzy inference systems, *IEEE Transactions on system, man and cybernetics, part b: cybernetics,* vol. 38, no. 4, pp. 963-969

New Trends in Technologies: Control, Management, Computational Intelligence and Network Systems
Edited by Meng Joo Er

ISBN 978-953-307-213-5
Hard cover, 438 pages
Publisher Sciyo
Published online 02, November, 2010
Published in print edition November, 2010

The grandest accomplishments of engineering took place in the twentieth century. The widespread development and distribution of electricity and clean water, automobiles and airplanes, radio and television, spacecraft and lasers, antibiotics and medical imaging, computers and the Internet are just some of the highlights from a century in which engineering revolutionized and improved virtually every aspect of human life. In this book, the authors provide a glimpse of the new trends of technologies pertaining to control, management, computational intelligence and network systems.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen