Construction of fuzzy systems using least-squares method and genetic algorithm

Cheol W. Lee, Yung C. Shin*

School of Mechanical Engineering, Purdue University, 1288 Mechanical Engineering Building, West Lafayette, IN 47907, USA

Received 28 November 2000; received in revised form 16 April 2002; accepted 21 June 2002

Abstract

The fuzzy basis function network which was proposed in Wang and Mendel (IEEE Trans. Neural Networks 3(5) (1992b) 807) provides a way of representing fuzzy inference systems in a simple structure similar to those of radial basis function networks. In this paper, two new algorithms based on the least-squares method and genetic algorithm are proposed for autonomous learning and construction of fuzzy basis function networks when training data are available. The proposed algorithms add a significant fuzzy basis function node at each iteration during training, based on error reduction measures. The first, a least-squares algorithm, provides a way of sequentially constructing meaningful fuzzy systems which are not possible to achieve with the orthogonal least-squares algorithm, while the second, an adaptive least-squares algorithm based on the combined least-squares and genetic algorithm, realizes hybrid structure-parameter learning without human intervention. Simulation studies are performed with numerical examples for comparison of its performance against the orthogonal least-squares algorithm, backpropagation algorithm, and conventional genetic algorithm. The adaptive least-squares algorithm is also applied to a real world problem to construct a fuzzy basis function network model for surface roughness in a grinding process using experimental data.

Keywords: Fuzzy basis function network; Autonomous learning; Hybrid structure-parameter learning; Genetic algorithms

1. Introduction

Neuro-fuzzy networks are based on the fusion of ideas from fuzzy inference and neural networks [6,9,10,13,14,17–19,21,22]. They are based on the observation that fuzzy inference systems can be represented as layered feedforward neural networks. Neuro-fuzzy networks represent fuzzy inference systems with learning abilities from data. Contrary to artificial neural networks, they provide abilities...
to incorporate expert knowledge expressed as IF-THEN rules and their parameters can be related to those of physical systems. Various neuro-fuzzy networks have been proposed and can be classified depending on the type of fuzzy systems from which the neuro-fuzzy networks are derived. Most of them are, however, derived from fuzzy systems whose output membership functions are singletons \([6,10,14,17–19,21,22]\) or linear combinations of input variables \([9]\) since they allow the fuzzy systems to be converted to neural networks of simple structure. Among them, the fuzzy basis function networks (FBFNs), which are similar in structure to radial basis function networks \([3]\), have gained much attention \([10,14,17,19,21,22]\). In addition to their simple structure, FBFNs possess another advantage that they can readily adopt various learning algorithms already developed for radial basis function networks. It has also been shown that FBFNs are capable of uniformly approximating any continuous nonlinear functions to a prescribed degree of accuracy with a finite number of basis functions \([22]\). This paper presents autonomous learning algorithms for FBFNs when input and desired output data are available.

Since the backpropagation algorithm is the most popular training algorithm developed for feedforward neural networks, it is natural that gradient-based algorithms also prevail among various learning algorithms developed for neuro-fuzzy networks \([6,9,21]\). Backpropagation algorithms, in general, suffer from drawbacks such as slow convergence and convergence to a local minimum. However, since many remedies and advanced algorithms based on numerical optimization theories have been developed to improve the performance of gradient-based algorithms, the drawback of applying a gradient-based algorithm should be found in another aspect. Gradient-based algorithms require that the number of fuzzy rules and membership functions be predefined. Although heuristic knowledge can be incorporated to adaptively add or remove membership functions, it does not replace the functions of structure learning algorithms. To address this problem, several hybrid structure-parameter learning schemes have been proposed for FBFNs, such as those based on the clustering algorithm \([17]\) or the projection theorem \([10]\). However, all of them require a certain form of information about the structure or parameters of FBFN before learning, i.e., either the number of fuzzy rules must be predefined or at least some of nonlinear parameters must be fixed. When FBFNs are applied to industrial problems or unknown systems, this requirement leads to a difficulty in implementation. Therefore, the need for developing learning algorithms for FBFNs that do not require human intervention has increased.

The orthogonal least-squares algorithm which was originally developed for radial basis function networks in \([2]\) has been applied to FBFNs \([22]\). It determines significant basis functions from candidate fuzzy basis functions according to the amount of error reduction. The orthogonal least-squares algorithm has attracted much attention due to its simple and straightforward computation with reliable performance. However, its performance largely depends on the preset input membership functions because those parameters remain unchanged during learning. Furthermore, it fails to yield a meaningful fuzzy system after learning as will be explained in the next section.

The two novel algorithms proposed in this paper provide new ways of sequentially constructing FBFNs based on the least-squares (LS) method. The first, an LS algorithm, is analogous to the orthogonal least-squares algorithm. It selects significant fuzzy rules from candidate basis functions based on an error reduction measure which is calculated by a projection matrix instead of orthogonalization. The most valuable feature of the LS algorithm which lacks in the conventional orthogonal least-squares algorithm is that it can generate a true fuzzy inference system while the orthogonal least-squares algorithm fails to do so. While the LS algorithm is a computationally efficient way of
constructing a fuzzy system, its performance still depends on the preset parameters of basis functions. The second, an adaptive least-squares (ALS) algorithm, avoids this problem by globally searching for candidate fuzzy rules in the universe of discourse using the genetic algorithm (GA) and makes it possible to construct a more parsimonious network without requiring any parameters or structure of the networks to be predefined.

GA is a search algorithm utilizing the population that is composed of many individuals encoded in binary strings. By imitating evolution processes in nature, the GA provides a faster way of achieving global optimum than mere random search methods [7]. Hence, the evolution of neural networks such as radial basis function networks and neuro-fuzzy networks using the GA has been a subject of interest in the research community.

Among several approaches to radial basis function networks, the simplest way of adopting GA is to encode a whole radial basis function network into each single individual in the population of GA [1,3,11]. Most applications of GA to training of neuro-fuzzy networks have also been based on encoding an entire network into each single individual of population [14,18,19]. In their training methods, a binary string can be excessively long for a large network, leading to long computation time and large search domain. Another disadvantage, similar to that of the backpropagation algorithm, can be found in the fact that the number of hidden nodes needs to be predefined by the user. More recent approaches include the local competition among hidden nodes [23] and cooperative evolution of radial basis function network using the evolutionary programming [20], in which a radial basis function network is represented by the population of GA.

The sequential learning method of FBFNs using GA, proposed in this study, provides valuable advantages over the existing GA-based learning algorithms. With a sequential learning mechanism where the network size increases gradually, one does not need to decide the number of hidden nodes before training since it can be determined automatically during training by proper stopping conditions. Furthermore, the computational time associated with GA-based search is less affected by the network size with the proposed algorithm since each binary string represents only one hidden node of the network.

Some researchers have also worked on the idea of constructing fuzzy logic systems in sequence [4,8]. Since both approaches were developed in the context of original fuzzy logic systems, a number of criteria based on heuristics were needed [8] or several runs of GA were used to find input membership functions and output membership functions at separate stages [4]. The proposed ALS algorithm, which has been developed for neuro-fuzzy networks, is more straightforward than the above two algorithms since the fitness function of the GA in the ALS algorithm is based on a single criterion of reducing modeling errors. In addition, the proposed algorithm can simultaneously determine input and output membership functions of a new fuzzy rule.

In Section 2, FBFN is introduced and the limitations of the conventional orthogonal least-squares algorithm are pointed out. The two new algorithms are described and their application results are presented in Sections 3 and 4. Section 5 concludes the paper.

2. Fuzzy basis function network

The FBFN started from an idea that a fuzzy inference system with product inference, singleton output fuzzy membership function, a centroid defuzzifier, and Gaussian input membership function
has a very similar structure to that of radial basis function network. In this section, the FBFN that will be used throughout this study is defined, and then the conventional orthogonal least-squares algorithm is described. The problems of applying the orthogonal least-squares algorithm to FBFN are pointed out and the needs for developing new algorithms based on the LS method are explained.

2.1. Definition of FBFN

Consider a multi-input-single-output fuzzy inference system: \( X \subset \mathbb{R}^n \rightarrow Y \subset \mathbb{R} \). Suppose we have \( M \) fuzzy logic rules in the following form:

\[
R_j : \text{IF } x_1 \text{ is } A_{1j} \text{ AND } x_2 \text{ is } A_{2j} \text{ AND } \cdots \text{ AND } x_n \text{ is } A_{nj}, \text{ THEN } y \text{ is } b_j, \tag{1}
\]

where \( A_{ij} \) is a linguistic term characterized by a fuzzy membership function \( \mu_{A_{ij}}(x_i) \), \( b_j \) is a singleton, and \( j = 1, 2, \ldots, M \). It is also assumed that a singleton fuzzifier, product inference, a centroid defuzzifier, and Gaussian input membership functions are used. Then for a given crisp input vector \( x = (x_1, x_2, \ldots, x_n)^T \in X \), the following defuzzified inferenced output can be obtained

\[
y = f(x) = \frac{\sum_{j=1}^{M} b_j(\prod_{i=1}^{n} \mu_{A_{ij}}(x_i))}{\sum_{j=1}^{M}(\prod_{i=1}^{n} \mu_{A_{ij}}(x_i))}, \tag{2}
\]

where \( f : X \subset \mathbb{R}^n \rightarrow Y \subset \mathbb{R} \) and \( \mu_{A_{ij}}(x_i) \) is the Gaussian membership function defined by

\[
\mu_{A_{ij}}(x_i) = \exp \left[ -\frac{1}{2} \left( \frac{x_i - m_{ij}}{\sigma_{ij}} \right)^2 \right], \tag{3}
\]

where \( m_{ij} \) and \( \sigma_{ij} \) are real-valued parameters.

Hence, fuzzy basis functions can be defined as

\[
p_j(x) = \frac{\prod_{i=1}^{n} \mu_{A_{ij}}(x_i)}{\sum_{j=1}^{M}(\prod_{i=1}^{n} \mu_{A_{ij}}(x_i))}, \quad j = 1, 2, \ldots, M. \tag{4}
\]

It can be observed that the \( j \)th fuzzy basis function can be defined by a nonlinear parameter set \( \lambda_j = \{m_j, \sigma_j\} \), where \( m_j = (m_{1j}, m_{2j}, \ldots, m_{nj})^T \in \mathbb{R}^n \) and \( \sigma_j = (\sigma_{1j}, \sigma_{2j}, \ldots, \sigma_{nj})^T \in \mathbb{R}^n \) are the center and width vectors of input membership functions, respectively. Then, the fuzzy inference system is equivalent to a fuzzy basis function expansion or FBFN:

\[
f(x) = \sum_{j=1}^{M} p_j(x)w_j, \tag{5}
\]

where \( w_j = b_j \in \mathbb{R} \) are constants. Therefore, the fuzzy inference system can be viewed as a linear combination of fuzzy basis functions. Fig. 1 shows an example of FBFN with 4 fuzzy rules for 3 input variables and 1 output variable.
Assume that $N$ input–output training pairs are given: $(x(i), d(i)), i = 1, 2, \ldots, N$. The task of training FBFN is to design an FBFN $f(x)$ such that

$$d(i) = f(x(i)) + e(i) = \sum_{j=1}^{M} p_j(x(i))w_j + e(i). \quad (6)$$

The above equation can be arranged from $i = 1$ to $N$ in matrix form:

$$d = Pw + e,$$ \quad (7)

where $d=[d(1), \ldots, d(N)]^T$, $P=[p_1, \ldots, p_M]$ with $p_j=[p_j(x(1)), \ldots, p_j(x(N))]^T$, $w=[w_1, \ldots, w_M]^T$, and $e=[e(1), \ldots, e(N)]^T$. Hence, constructing the FBFN becomes a simple linear least-squares problem once the response vector matrix $P$, whose column vectors are response vectors of fuzzy basis function nodes, is determined. However, the remaining question is how the fuzzy basis function nodes, specifically the nonlinear parameter sets $\lambda_j = \{m^j, \sigma^j\}$, are chosen or determined.

2.2. Conventional orthogonal least-squares algorithm for FBFN

The orthogonal least-squares algorithm is an efficient model selection method for linear regression models [2,22]. In the orthogonal least-squares algorithm, all the training data points become the candidate centers of the FBFN and the width of each candidate fuzzy basis function node is prefixed by the user. Accordingly, the response vectors of all the candidate fuzzy basis function nodes are generated and then stored. Among $N - l$ response vectors, $p_j$’s, from candidate fuzzy basis function nodes, where $l$ denotes the number of fuzzy basis function nodes already found, the orthogonal least-squares algorithm sequentially chooses the best fuzzy basis function node that yields the maximum error reduction measure, which is given by the squared norm of the projection of $d$ onto $p_j$. At
each iteration, the desired output vector \( d \) and the remaining response vectors \( p_j \)'s are updated by an orthogonalization procedure for the next step.

Since the orthogonal least-squares algorithm utilizes the computationally efficient orthogonal least-squares techniques, it has attracted the attention of many researchers who were interested in a fast and reliable implementation of radial basis function networks and FBFNs. However, the performance of networks trained by the orthogonal least-squares algorithm can significantly depend on the prefixed width of hidden units. In addition, setting all the widths of hidden nodes uniformly regardless of the position of centers tends to restrict the performance of the orthogonal least-squares in some applications.

In order to observe another problem of the orthogonal least-squares algorithm applied to the FBFN, consider the original fuzzy basis functions before the orthogonal least-squares learning when \( N \) training data pairs are given. According to the orthogonal least-squares algorithm the following \( N \) candidate fuzzy basis function nodes are generated at the beginning:

\[
p_k(x) = \frac{\prod_{i=1}^{n} \mu_{A_k}(x_i)}{\sum_{k=1}^{N}(\prod_{i=1}^{n} \mu_{A_k}(x_i))}, \quad k = 1, 2, \ldots, N.
\]  

(8)

Assume that \( M \) (\( M < N \)) FBFN nodes have been selected by the orthogonal least-squares algorithm, then the resultant fuzzy basis function expansion produced by the orthogonal least-squares algorithm is represented by

\[
f(x) = \sum_{j=1}^{M} p_{kj}(x)w_j
\]

\[
= \frac{\sum_{j=1}^{M} w_j(\prod_{i=1}^{n} \mu_{k_j}(x_i))}{\sum_{k=1}^{N}(\prod_{i=1}^{n} \mu_{A_k}(x_i))},
\]  

(9)

where \( k_j \in \{1, 2, \ldots, N\} \). Compare Eq. (9) with the original definition of FBFN rewritten as follows:

\[
f(x) = \frac{\sum_{j=1}^{M} w_j(\prod_{i=1}^{n} \mu_{A_j}(x_i))}{\sum_{j=1}^{M}(\prod_{i=1}^{n} \mu_{A_j}(x_i))}.
\]  

(10)

It is not difficult to see that the fuzzy basis functions selected by the orthogonal least-squares algorithm can never be interpreted as true fuzzy inference systems since they retain the normalization factor in the denominator from the original FBFN before training.

The above problem can be easily seen from the following example. Suppose there are 4 pairs of data points for a 3 input and 1 output system. Then, the orthogonal least-squares algorithm starts from an FBFN with 4 candidate basis functions as shown in Fig. 1. If two significant fuzzy basis function nodes are chosen by the conventional orthogonal least-squares algorithm, the final network would have the structure as shown in Fig. 2(a). It can be easily seen that this network cannot be interpreted as a true fuzzy inference system from which a set of linguistic fuzzy rules can be generated. This deficiency voids the very rationale behind pursuing any neuro-fuzzy networks whose structure and parameters can be clearly understood in terms of fuzzy rules defined in physical domains. The two
new learning algorithms proposed in this paper have been developed to produce FBFNs from which meaningful fuzzy logic systems can be derived.

3. Sequential construction of fuzzy basis function network based on least-squares algorithm

The LS algorithm proposed in this section is based on a linear least-squares technique. Similar to the orthogonal least-squares algorithm, it starts with the preset basis functions and sequentially selects the basis functions that will decrease errors the most. However, the proposed LS algorithm adopts pseudo-fuzzy basis functions instead of fuzzy basis functions and calculates a projection matrix instead of performing Gram–Schmidt orthogonalization. These differences make it possible to obtain a meaningful fuzzy system after learning while retaining computational efficiency.
3.1. Algorithmic description

In order to present the new LS algorithm, we define the pseudo-fuzzy basis function, which is the product of all membership functions for the linguistic terms in the IF part of $R_i$, as follows:

$$q_j(x) = \prod_{i=1}^{n} \mu_{A_i'}(x_i).$$  \hfill (11)

Suppose $M$ fuzzy rules are picked from the data points, and then the fuzzy basis function can be expressed in terms of pseudo-fuzzy basis functions as follows:

$$p_j = \frac{q_j(x)}{\sum_{k=1}^{M} q_k(x)}. \hfill (12)$$

For $N$ input-output training pairs: $(x(i), d(i)), i = 1, 2, \ldots, N$, the response vector of the $j$th pseudo-fuzzy basis function is defined as follows:

$$q_j = [q_j(x(1)), \ldots, q_j(x(N))]^T. \hfill (13)$$

Selected $M$ pseudo-response vectors are arranged to form the following pseudo-response matrix:

$$Q = [q_1, \ldots, q_M]. \hfill (14)$$

When there is only one fuzzy rule present ($M=1$), the pseudo-response matrix $Q$ corresponds to the response matrix $P$. When $M > 1$, the response matrix $P$ in Eq. (7) can be easily calculated from the pseudo-response matrix $Q$ using the following equation:

$$P(i,j) = Q(i,j)/\text{sum}(i),$$

$$\text{sum}(i) = \sum_{j=1}^{M} Q(i,j), \hfill (15)$$

where $(i,j)$ denotes the index of entry in the $i$th row of the $j$th column of the matrix. Hence, whenever a new fuzzy rule expressed as a pseudo-fuzzy basis function needs to be added to the already found fuzzy system, one can simply obtain a new response matrix $P$ by first adding a new pseudo-response vector to the pseudo-response matrix $Q$ in Eq. (14) as a new column and then applying Eq. (15). These features provide a way of sequentially constructing a fuzzy inference system. The detailed description of the algorithm is given below. The objective is to find $M$ fuzzy rules when $N$ data points are available.

First, $N$ initial pseudo-fuzzy basis functions, $q_j(x)$, are formed from the training data with the parameters determined as follows:

$$m^j = x(j),$$

$$\Delta x_i = \max_{j=1,\ldots,N} x_i(j) - \min_{j=1,\ldots,N} x_i(j),$$

$$\sigma_i^j = \Delta x_i/M, \hfill (16)$$
where \(i = 1, 2, \ldots, n\) and \(j = 1, 2, \ldots, N\). At the same time, \(N\) pseudo-response vectors are generated according to Eqs. (11) and (13). In the proposed algorithm, one pseudo-fuzzy basis function is selected from the candidate pseudo-fuzzy basis functions at each iteration until the total number of chosen pseudo-fuzzy basis function reaches \(M\). Consider, for example, when the algorithm reaches the \((l+1)\)th iteration, where \(l\) denotes the number of pseudo-fuzzy basis functions found in the previous iterations. At this stage, the pseudo-response matrix \(Q\) consists of \(l\) pseudo-response vectors. Among remaining \(N-l\) pseudo-response vectors, the algorithm picks up a pseudo-fuzzy basis function which will maximize the following error reduction measure \([\text{err}]\):

\[
[\text{err}] = \| PP^+ d \|, \tag{17}
\]

where \(P^+\) denotes the pseudoinverse of \(P\). The response matrix \(P\) can be obtained by first adding a new pseudo-response vector to the existing pseudo-response matrix \(Q\) as a new column and then applying Eq. (15). It can be seen that \(PP^+\) is the orthogonal projection onto the column space of \(P\).

Once \(M\) fuzzy basis function nodes are found, the weight vector \(w\) can be calculated using the following equation:

\[
w = P^+ d. \tag{18}\]

The final FBFN is

\[
f(x) = \sum_{j=1}^{M} p_j(x)w_j. \tag{19}\]

**Note:** There could be several ways to obtain the projection matrix or pseudoinverse in Eqs. (17) and (18) [5]. In fact, there is no need at all to get the explicit form of projection matrix or pseudoinverse to evaluate the equations. The authors suggest use of the QR decomposition algorithm which is computationally more efficient than the singular value decomposition algorithm.

### 3.2. Simulation

In order to evaluate how effectively the new algorithm constructs a fuzzy system from the candidate pseudo-fuzzy basis functions, its performance was compared with that of the network constructed by applying the orthogonal least-squares algorithm proposed in [22]. The performance was measured by comparing the capabilities of approximating two nonlinear functions. The base of membership functions in both algorithms was fixed to cover the entire input domain uniformly as in Eq. (16).

**Example 1.**

\[
F(x_1, x_2) = \frac{1}{5}[\tanh(9x_2 - 9x_1) + 1], \quad 0 \leq x_1 \leq 1, \quad 0 \leq x_2 \leq 1. \tag{20}\]

From this function, 100 uniformly distributed data were used for training, while another 81 uniformly distributed data were used for testing the generalizing capabilities of FBFN. The number of fuzzy rules \(M\) was set to 5. In both algorithms, the number of fuzzy rules starts from 0 and continue to increase one by one. At each iteration when a new fuzzy rule is added, the training and testing errors were calculated and then plotted in Fig. 3. The horizontal axis indicates the number of fuzzy
rules found, while the vertical axis represents the nondimensional error index of the training and testing data which is calculated by the following equation [9]:

$$\text{Nondimensional error index} = \sqrt{\frac{\sum_{k=1}^{N} [d(k) - y(k)]^2}{\sum_{k=1}^{N} [d(k) - \bar{d}]^2}},$$

(21)

where $\bar{d} \in \mathbb{R}$ is the mean value of entries in $\mathbf{d}$. It can be observed from Fig. 3 that the new algorithm achieves lower error than the orthogonal least-squares algorithm with the same number of fuzzy rules larger than 2. Fig. 4 compares the approximation capabilities of the FBFNs obtained by the two algorithms with the original function. As it can be expected from Fig. 3, the FBFN obtained by applying the new algorithm provides almost identical shape to the original function with only 5 fuzzy rules, while the networks obtained by applying the orthogonal least-squares algorithm failed to approximate the original function. Additional rules would be needed for the orthogonal least-squares algorithm to achieve a better approximation.

**Example 2.**

$$F(x_1, x_2, x_3) = (1 + x_1^{0.5} + x_2^{-1} + x_3^{-1.5})^2, \quad 1 \leq x_1 \leq 6, \quad 1 \leq x_2 \leq 6, \quad 1 \leq x_3 \leq 6.$$  

(22)
For this function, 300 samples were randomly chosen for learning, while another 300 points were used for testing the generalizing capabilities of FBFN. Fig. 5 shows the training and testing errors during learning till the number of fuzzy rules becomes 4. The FBFN trained by the LS algorithm achieved less than one tenth of the initial error with 4 rules, while the network constructed by the orthogonal least-squares algorithm failed to achieve that same low level error.

It should be noted that the networks constructed by the orthogonal least-squares algorithm cannot be considered as meaningful fuzzy systems due to the reasons explained in Section 2.2 although they can still be used for approximating nonlinear functions. From the above examples, it can be observed that the new algorithm based on the LS method not only provides a way of constructing meaningful fuzzy systems, but also shows superior capabilities in training FBFNs for nonlinear function approximation over the orthogonal least-squares algorithm. It seems that the $N - M$ unnecessary normalization terms in the network constructed by the orthogonal least-squares algorithm, as shown in (9), lead to poor performance of the network.

Searching for all the possible combination of fuzzy rules will be very time-consuming especially when the number of data points is large. Although it has not been proven whether the fuzzy system obtained by applying the new algorithm is the optimal combination of fuzzy rules among all the possible combinations of candidate fuzzy rules, the new algorithm provides a computationally efficient way of sequentially constructing fuzzy systems with acceptable performance.
4. Sequential construction of fuzzy basis function network based on adaptive least-squares algorithm

While the LS algorithm in the previous section shows satisfactory performance, it has several limitations because the performance of the previous algorithm depends on the preset parameters of pseudo-FBF. The ALS algorithm in this section improves the accuracy of the previous algorithm by searching for the best pseudo-fuzzy basis function node in the universe of discourse using the GA. This will make it possible to construct a parsimonious network in a sequential manner.

4.1. Algorithmic description

Due to introduction of the pseudo-fuzzy basis function \( q_j(x) \), construction of the FBFN becomes a problem of sequentially finding a new pseudo-fuzzy basis function which is the product of input membership functions of a new fuzzy rule. From Eq. (3), it can be observed that a pseudo-fuzzy basis function can be defined by a nonlinear parameter set \( \lambda = \{ m, \sigma \} \), where \( m = (m_1, \ldots, m_n)^T \in \mathbb{R}^n \) and \( \sigma = (\sigma_1, \ldots, \sigma_n)^T \in \mathbb{R}^n \) are the center and width vectors of input membership functions, respectively. The ALS algorithm tries to locate one pseudo-fuzzy basis function or parameter set \( \lambda \) at a time in sequence, which will maximize the following error reduction measure using the GA:

\[
[err] = \|PP^+d\|^2. 
\]
While the LS algorithm selects one hidden node at a time from a given pool of candidate hidden nodes that are predefined, the ALS algorithm directly searches for the best hidden node in the input domain. In this way, the ALS algorithm can be completely immune to the initial setting of nonlinear parameters.

The ALS algorithm is described below. In the proposed algorithm, \( l \) denotes the number of fuzzy rules found. The objective is to find \( M \) fuzzy rules when \( N \) data points are available.

**Step 1.** Initialize as \( l = 0 \). The pseudo-response matrix \( \mathbf{Q} \) is initialized as an empty matrix.

**Step 2.** Search for a new pseudo-fuzzy basis function that will maximize the error reduction measure using the GA. A detailed description is given below.

**Step 3.** Insert the response vector \( \mathbf{q} \) of a newly found pseudo-fuzzy basis function into \( \mathbf{Q} \). Set \( l = l + 1 \).

**Step 4.** If \( l < M \), go to step 2.

**Step 5.** Generate the \( \mathbf{P} \) matrix from \( \mathbf{Q} \) using Eq. (15). The weight vector \( \mathbf{w} \) can be calculated using the following equation:

\[
\mathbf{w} = \mathbf{P}^+ \mathbf{d}. \tag{24}
\]

Obtain the final FBFN as

\[
f(\mathbf{x}) = \sum_{j=1}^{M} p_j(\mathbf{x})w_j. \tag{25}
\]

For the GA procedure in step 2, the search space of GA has to be defined first according to the input domain boundary of training data. Assume that the domain of \( i \)th input variable has been found to be \([\min(x_i), \max(x_i)]\) from training data, then the domains of \( m_i \) and \( \sigma_i \) are given as \([\min(x_i) - \delta_i, \max(x_i) + \delta_i]\) and \((0, (\max(x_i) - \min(x_i))/2]\), respectively, where \( \delta_i \) is a small positive value. By adopting the \( \delta_i \), the search space of center \( m_i \) can be extended over the domain boundary of training data. In this study, the \( \delta_i \) value was chosen at \((\max(x_i) - \min(x_i)) / 10\). After the search space is defined, the initial population is created simply at random in the given space according to the following encoding scheme.

The parameter set of a hidden node, \( \lambda = \{\mathbf{m}, \sigma\} \) is encoded into a binary string, which becomes an individual in the population of the GA. For example, if the number of input variables is 2, then the nonlinear parameter set is given by \( \lambda = \{(m_1, m_2)^T, (\sigma_1, \sigma_2)^T\} \). Suppose a 4-bit resolution is employed, the parameter set can be encoded into a binary string as follows:

\[
\begin{array}{cccc}
m_1 & m_2 & \sigma_1 & \sigma_2 \\
\downarrow & & & \\
\begin{array}{cccc} a_1 & a_2 & a_3 & a_4 \end{array} & \begin{array}{cccc} b_1 & b_2 & b_3 & b_4 \end{array} & \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} & \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \end{array} \\
\end{array}
\]

where \( a_i, b_i, c_i, \) and \( d_i \) \((i = 1, 2, 3, 4)\) denote the binary digits with the values of 0 or 1. Decoding of binary strings into real numbers is realized by an inverse mapping.

The population of GA is composed of a pre-fixed number of binary strings. The pre-fixed number is termed population size and each binary string corresponds to the parameter set of a candidate fuzzy basis function node according to the encoding scheme shown above. The population of GA evolves from a generation to a next generation by undergoing reproduction, crossover, and mutation.
Reproduction is a genetic operator in which the probability of a binary string being regenerated is proportional to its fitness value. The fitness function $g(\lambda)$ of each individual in the population of ALS algorithm is given by the following procedure:

(A) Using the given nonlinear parameter set $\lambda$ of the new fuzzy rule, construct a new pseudo-fuzzy basis function and its response vector according to Eqs. (3), (11), and (13).

(B) Add the new pseudo-response vector to the pseudo-response matrix $Q$ as a new column.

(C) Apply Eq. (15) to obtain a new response matrix $P$.

(D) Calculate the error reduction measure using Eq. (23).

(E) The error reduction measure is linearly scaled to produce the fitness function value $g(\lambda)$ as follows:

$$g(\lambda) = a[err] + b$$

where $a$ and $b$ are scalar parameters chosen to prevent premature convergence or random walk [7].

An individual string with a larger fitness value has more chance to be reproduced. This can be achieved by the biased roulette wheel method in which the probability of an individual to be reproduced is linearly proportional to its fitness divided by the summation of all the fitness values in the population. An exact copy of the chosen individual is made and stored in a temporary mating pool waiting for the mating process by the crossover operator. This process is repeated until the number of reproduced individuals reaches the population size.

The crossover operator enables two different individuals in the population to exchange genetic information each other [16]. First, two binary strings are selected at random from the mating pool. It depends on the crossover probability whether the two individuals are to be mated or not. If it is decided that a mating process is to be skipped, the two binary strings become offsprings without any changes. Otherwise, the two individuals go through the mating process. It should be noted here that the crossover point is not limited to the boundaries between parameters such as $m_i$ and $\sigma_i$, but can be located anywhere in the binary string. This scheme can increase the chance of obtaining the global optimum by allowing the offsprings to have versatile genetic information.

A mutation process is applied to the offsprings to add new genetic information into the binary strings. For each bit in the binary strings, a random number is generated uniformly between 0 and 1. If the random number is smaller than the mutation probability, which is usually preset at a very low value, the corresponding bit is inverted. Otherwise, the bit remains unchanged.

In step 2, the population undergoes the selection, reproduction, and mutation for a user-specified number of generations and evolves to an optimum hidden node that maximizes the error reduction measure in Eq. (23). The individual with the greatest fitness value during the whole GA procedure is chosen and then decoded into one hidden node.

The above GA procedure is repeated $M$ times until a set of $M$ hidden nodes are generated. The linear weights, which correspond to the output membership functions, are obtained by using Eq. (24). Although the above algorithm requires condition for the number of fuzzy rules to be found, the algorithm can be easily modified to run continuously until certain stopping conditions are met. It must be noted that the above algorithm does not require any form of information before learning on the final fuzzy system while most other parameter or structure-parameter learning algorithms require that the user specify at least one of the parameters
of neuro-fuzzy network [6,9,10,13,17,19,21,22]. Therefore, the new algorithm can be considered as a true hybrid structure-parameter learning scheme for FBFNs.

4.2. Evaluation

The performance of the proposed ALS algorithm is evaluated through simulation with two numerical examples and modeling of one physical system. The capabilities of the ALS algorithm in approximating nonlinear functions are compared with those of the backpropagation algorithm and the conventional GA-based algorithm in the next section. The proposed algorithm is, then, used to construct a process model for surface roughness in a surface grinding process using experimental data.

4.2.1. Simulation studies

Example 3. In this example, the proposed algorithm is compared with the backpropagation algorithm which is modified from the gradient-based algorithm proposed in [21]. An incremental training method was derived in [21] for on-line tuning based on the gradient of the objective function, which is the squared error of each training pair. In this study, the sum of squares of entire training error is used instead as the objective function in order to achieve batch-mode training, and corresponding updating equations are given below. Please refer to [21] for detailed derivation.

\[
\begin{align*}
    w_j(\eta + 1) &= w_j(\eta) - \alpha \sum_{h=1}^{N} p_j(h)[f(h) - d(h)], \\
    m_i^j(\eta + 1) &= m_i^j(\eta) - \alpha \sum_{k=1}^{N} p_j(h)[f(h) - d(h)][w_j(\eta) - f(h)] \frac{x_i(h) - m_i^j(\eta)}{\sigma_i^j(\eta)}, \\
    \sigma_i^j(\eta + 1) &= \sigma_i^j(\eta) - \alpha \sum_{k=1}^{N} p_j(h)[f(h) - d(h)][w_j(\eta) - f(h)] \frac{[x_i(h) - m_i^j(\eta)]^2}{\sigma_i^j(\eta)},
\end{align*}
\]

where \( \eta \) and \( \alpha \) denote the number of epochs and a learning constant, respectively. In order to ensure stable convergence and expedite the training, the adaptive learning constant scheme is adopted, which has been widely used to improve the classic gradient-based algorithm [9,13]. If the error decreased consecutively, the learning rate was increased by a factor of 0.05, while the learning rate was decreased by a factor of 0.3 and the update was cancelled if the error increased. Training was terminated when the change of training error was less than 0.0001% for 10 consecutive epochs.

In order to evaluate the performance of the ALS algorithm against the backpropagation algorithm, the following nonlinear function was chosen:

\[
F(x_1, x_2) = 3(1 - x_1)^2e^{-x_1^2 - (x_2 + 1)^2} - 10 \left( \frac{x_1}{5} - x_1^3 - x_2^4 \right)e^{-x_1^2 - x_2^2} - \frac{1}{3} e^{-(x_1 + 1)^2 - x_2^2},
\]

\[
-3 \leq x_1 \leq 3, \quad -3 \leq x_2 \leq 3.
\]

For the above functions, 900 uniformly distributed samples were generated and used for training, while another 841 uniformly distributed data were used for testing the generalizing capabilities of
FBFN. The parameters of GA were chosen as follows: population size = 200, generation gap = 1.0, crossover probability = 1.0, mutation probability = 0.005, number of generations = 100, number of bits used for building chromosomes = 12 bits for each $m_i^j$ and $\sigma_i^j$.

For the backpropagation algorithm, the number of fuzzy rules had to be fixed before training because backpropagation is strictly a parameter learning algorithm. The initial centers of membership functions were chosen to be uniformly distributed in the universe of discourse while initial widths of membership functions were calculated to cover the universe of discourse with $\varepsilon$-completeness of $\exp(-0.5)$ [13]. The $w_j$'s were initialized by applying Eq. (24).

The simulation results of Example 3 are shown in Figs. 6–9. Fig. 6 shows the training and testing errors versus number of fuzzy rules as the fuzzy systems are sequentially constructed by the ALS algorithm. Except when the number of fuzzy rules increases from 1 to 2, both the training and testing errors decrease as a new fuzzy rule is added to the previous fuzzy system. Fig. 7 shows an example of error curves versus number of epochs for $M = 25$ with the backpropagation algorithm. It can be seen that after the initial decrease of error, it stays at a high level of error even after 30,000 epochs because learning is trapped in a local minimum. The comparison of training and testing errors between two algorithms for $M = 10, 15, 20$, and 25 are shown in Fig. 8. The proposed ALS algorithm provides superior results over the backpropagation algorithm except the case when $M = 15$. It can be seen that, with the backpropagation algorithm, a larger number of hidden nodes in the FBFN do not necessarily lead to a smaller training error or testing error. This is due to convergence to a local minimum by the backpropagation algorithm for $M = 20$ and 25.

It should be noted that the backpropagation algorithm had to be applied individually for each predefined number of fuzzy rules to obtain the plot while the ALS algorithm provided all the results in one pass. It means that the number of fuzzy rules required to achieve a certain error goal can be determined only after many trials with the backpropagation algorithm while it can be determined...
Fig. 7. An example of error curves for Example 3 when the backpropagation algorithm is applied to FBFN with 25 fuzzy rules.

Fig. 8. Comparison between training and testing errors for the ALS algorithm and the backpropagation algorithm applied to Example 3. (a) Training errors, (b) testing errors.
in just one trial with the ALS algorithm. It can be observed from the graphs of testing error that the ALS algorithm also shows outstanding performance in generalization. Fig. 9 shows the function approximation of the FBFNs obtained from two algorithms compared with the original function. As it can be expected from the error curves in Fig. 8, the FBFNs trained by the backpropagation algorithm did not approximate the original function accurately with either 15 or 25 fuzzy rules, while the ALS algorithm yielded a very close approximation of the function with 25 fuzzy rules.

The simulation results of Example 3 show that the ALS algorithm provides superior capabilities in training FBFNs for nonlinear function approximation over the backpropagation algorithm. Although a more sophisticated algorithm based on numerical optimization theories with a judicious choice of initial parameters might achieve better performance than the backpropagation algorithm used in this study did, the advantage of the proposed algorithm should be stressed by the fact that the proposed ALS algorithm is a hybrid structure-parameter learning scheme which does not require any parameters of a fuzzy system to be predefined.

Example 4. In this example, the proposed algorithm is compared with a conventional way of applying the GA to training of neuro-fuzzy networks where all the parameters of the entire network
are encoded into a binary string \([14,18,19]\). A detailed description of the conventional GA-based algorithm implemented in this example is given below.

Suppose the objective is to find \(M\) fuzzy rules for a given set of training data. The nonlinear parameters such as centers and widths of \(M\) hidden nodes are encoded into a binary string and becomes an individual. For example, if the nonlinear parameter set of the \(j\)th hidden node with \(n\) input variables is represented by \(\lambda_j = \{m_j, \sigma_j\}\), where \(m_j \in \mathbb{R}^n\) and \(\sigma_j \in \mathbb{R}^n\), then an individual corresponds to a binary string which is obtained by encoding a concatenation of \(M\) parameter sets, \(\{\{m_1, \sigma_1\}, \{m_2, \sigma_2\}, \ldots, \{m_M, \sigma_M\}\}\). Note the difference between encoding schemes for the conventional GA-based algorithm and the proposed ALS algorithm, where an individual for the latter corresponds to a parameter set of one hidden node, \(\lambda = \{\mathbf{m}, \sigma\}\). The length of a binary string in conventional GA-based algorithms is \(Mnb\), where \(b\) is the number of binary bits used for building chromosomes, while that of the ALS algorithm is \(nb\). It can be seen that the search space of conventional GA-based algorithms is bound to increase substantially with increase in the number of hidden nodes, while that of the ALS algorithm remains constant regardless of network size. This difference leads to superior approximating performance of the proposed algorithm over the conventional GA-based algorithm when the same parameters, such as the population size and the number of generations, are chosen for the GA part in both algorithms. It can also provide a considerable amount of saving in computing time for the same number of GA runs.

Individuals in the conventional GA-based algorithm also evolve from a generation to a generation by undergoing reproduction, crossover, and mutation. The fitness function of each individual in the population is obtained by the following procedure. First, each individual is decoded into \(M\) hidden nodes and then their pseudo-response matrix \(\mathbf{Q}\) is calculated by Eqs. (11), (13), and (14). According to Eq. (15), a pseudo-response matrix \(\mathbf{Q}\) is converted into a response matrix \(\mathbf{P}\), based on which the error reduction measure is calculated by Eq. (17). The fitness function is obtained by applying a linear-scaling procedure, which is similar to the one in Eq. (26), to the error reduction measure in order to prevent premature convergence or random walk [7]. Although the GA procedure in the conventional GA-based algorithm is similar to that of the ALS algorithm, a significant difference can be found in the way of calculating a response matrix. Every column in the pseudo-response matrix \(\mathbf{Q}\) in Eq. (14) needs to be recalculated anew at each generation in the conventional GA-based algorithm since all the hidden nodes change from generation to generation. With the ALS algorithm, on the other hand, only one column needs to be calculated anew at each generation, which leads to a significant reduction in computing time for problems with a large number of training data points.

In order to evaluate the performance of the ALS algorithm against the conventional GA-based algorithm, the following nonlinear function was chosen:

\[
F(x_1, x_2) = \sin(x_1 x_2) e^{-|x_1 x_2|}, \quad -3 \leq x_1 \leq 3, \quad -3 \leq x_2 \leq 3.
\]  

For both of the above functions, 900 uniformly distributed samples were generated and used for training, while another 841 uniformly distributed data were used for testing the generalizing capabilities of FBFN. The parameters of GA for both algorithms were chosen as the same as those used for the ALS algorithm in Example 3. The backpropagation algorithm described in Example 3 was applied to Example 4 again for reference.
The simulation results of Example 4 are shown in Figs. 10–12. Fig. 10 shows the training and testing errors versus number of fuzzy rules for the backpropagation algorithm, the conventional GA-based algorithm, and the ALS algorithm. When the number of fuzzy rules was smaller than 8, the conventional GA-based algorithm achieved training and testing errors comparable to those of the ALS algorithm. However, the performance of the conventional GA-based algorithm dropped significantly when the number of fuzzy rules was larger than 8.

The above simulation result in Fig. 10 provides an interesting insight into application of GA to training of FBFNs. In an ideal situation where the GA can always find the global optimum, the sequential learning algorithm such as the ALS algorithm should never outperform the conventional way of finding all the fuzzy rules simultaneously. However, the performance of the GA becomes less effective with increase in the length of binary strings in many applications. The GA in the ALS algorithm, on the other hand, can maintain its high efficiency since the length of binary strings remains constantly small regardless of the number of fuzzy rules.

An attempt was made to compare the computation time of the ALS algorithm with that of the conventional GA-based algorithm. It was assumed that no information was available before training.
regarding the number of fuzzy rules required for approximation. One way of determining the number of fuzzy rules to approximate an unknown function would be observing the change of training and testing errors as increasing the number of fuzzy rules one by one. With the conventional GA-based algorithm, the cumulative computing time it takes until $M$ fuzzy rules are obtained can be calculated by summing computing times for $1, 2, \ldots, M$ fuzzy rules as follows:

$$\text{Cumulative computing time} = T_1 + T_2 + \cdots + T_M,$$

where $T_i$ is the computing time it takes to train a FBFN with $i$ fuzzy rules. With the proposed ALS algorithm, the total elapsed time from the start of the algorithm is considered since the ALS algorithm increases the number of fuzzy rules one by one during the program run. It should be noted that the computing time for generating $M$ fuzzy rules in sequence corresponds to $M$ separate GA runs with both algorithms.

Both algorithms were programmed in MATLAB language and run on an IBM-compatible PC with a 400 MHz celeron processor and 96 megabytes’ memory. Fig. 11 compares cumulative computing times for the conventional-GA based algorithm and the ALS algorithm as the number of fuzzy rules increases. It can be seen that the ALS algorithm takes shorter time than the conventional GA-based algorithm for all numbers of fuzzy rules.

Fig. 12 shows the function approximation of the FBFNs obtained from the ALS algorithm, the backpropagation algorithm, and the genetic algorithm compared with the original function. As it can be expected from the error curves in Fig. 10, the FBFNs trained by the backpropagation algorithm and the conventional-GA based algorithm did not approximate the original function accurately with 40 fuzzy rules, while the ALS algorithm yielded a very close approximation of the function with the same number of rules.
The simulation results of Example 4 show that the ALS algorithm provides superior performance over the conventional GA-based algorithm in terms of approximation capability and computation time for problems with a large number of training data and fuzzy rules.

4.2.2. Modeling of the surface grinding process

In order to demonstrate the benefits of FBFN, the proposed algorithm is applied to construct a four-input-one-output FBFN model for surface roughness in a surface grinding process using 25 sets of experimental data listed in Table 1. Modeling of the manufacturing process using a neuro-fuzzy networks can be useful when accurate analytical models are not available. Training of the neuro-fuzzy networks, however, tends to require a large amount of experimental data, which can be costly in actual operation. It can be easily expected that the FBFN, trained using the limited amount of experimental data, would have inadequate learning of the process since 25 sets of training data are, in most cases, insufficient for construction of a four-input-one-output FBFN.

The basic idea behind the approach taken in this case study is to make use of the ALS algorithm to extract fuzzy rules in rough form from the available knowledge, i.e., a generic form of the analytical model for initial construction of the network structure, and then tune the parameters of the constructed FBFN using a small number of available experimental training data. It is considered
Table 1
Training data for surface roughness model

<table>
<thead>
<tr>
<th>No.</th>
<th>Input variables</th>
<th>Crossfeed, ( s_t ) (mm)</th>
<th>Dressing depth, ( a_d ) (( \mu )m)</th>
<th>Dressing lead, ( s_d ) (mm)</th>
<th>Output variable, ( R_a ) (( \mu )m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.10</td>
<td>0.61</td>
<td>20</td>
<td>0.12</td>
<td>0.26</td>
</tr>
<tr>
<td>2</td>
<td>0.20</td>
<td>0.56</td>
<td>20</td>
<td>0.12</td>
<td>0.32</td>
</tr>
<tr>
<td>3</td>
<td>0.30</td>
<td>0.48</td>
<td>20</td>
<td>0.12</td>
<td>0.36</td>
</tr>
<tr>
<td>4</td>
<td>0.30</td>
<td>0.50</td>
<td>20</td>
<td>0.12</td>
<td>0.36</td>
</tr>
<tr>
<td>5</td>
<td>0.20</td>
<td>0.56</td>
<td>20</td>
<td>0.12</td>
<td>0.33</td>
</tr>
<tr>
<td>6</td>
<td>0.10</td>
<td>0.61</td>
<td>20</td>
<td>0.12</td>
<td>0.25</td>
</tr>
<tr>
<td>7</td>
<td>0.10</td>
<td>1.08</td>
<td>20</td>
<td>0.12</td>
<td>0.31</td>
</tr>
<tr>
<td>8</td>
<td>0.20</td>
<td>0.96</td>
<td>20</td>
<td>0.12</td>
<td>0.41</td>
</tr>
<tr>
<td>9</td>
<td>0.30</td>
<td>1.00</td>
<td>20</td>
<td>0.12</td>
<td>0.44</td>
</tr>
<tr>
<td>10</td>
<td>0.30</td>
<td>1.00</td>
<td>20</td>
<td>0.12</td>
<td>0.47</td>
</tr>
<tr>
<td>11</td>
<td>0.20</td>
<td>0.92</td>
<td>20</td>
<td>0.12</td>
<td>0.38</td>
</tr>
<tr>
<td>12</td>
<td>0.10</td>
<td>1.06</td>
<td>20</td>
<td>0.12</td>
<td>0.33</td>
</tr>
<tr>
<td>13</td>
<td>0.10</td>
<td>1.54</td>
<td>20</td>
<td>0.12</td>
<td>0.38</td>
</tr>
<tr>
<td>14</td>
<td>0.20</td>
<td>1.45</td>
<td>20</td>
<td>0.12</td>
<td>0.51</td>
</tr>
<tr>
<td>15</td>
<td>0.30</td>
<td>1.54</td>
<td>20</td>
<td>0.12</td>
<td>0.56</td>
</tr>
<tr>
<td>16</td>
<td>0.30</td>
<td>1.54</td>
<td>20</td>
<td>0.12</td>
<td>0.61</td>
</tr>
<tr>
<td>17</td>
<td>0.20</td>
<td>1.49</td>
<td>20</td>
<td>0.12</td>
<td>0.52</td>
</tr>
<tr>
<td>18</td>
<td>0.10</td>
<td>1.49</td>
<td>20</td>
<td>0.12</td>
<td>0.39</td>
</tr>
<tr>
<td>19</td>
<td>0.20</td>
<td>1.02</td>
<td>8</td>
<td>0.12</td>
<td>0.26</td>
</tr>
<tr>
<td>20</td>
<td>0.20</td>
<td>1.02</td>
<td>13</td>
<td>0.12</td>
<td>0.33</td>
</tr>
<tr>
<td>21</td>
<td>0.20</td>
<td>1.02</td>
<td>20</td>
<td>0.12</td>
<td>0.38</td>
</tr>
<tr>
<td>22</td>
<td>0.20</td>
<td>1.02</td>
<td>30</td>
<td>0.12</td>
<td>0.43</td>
</tr>
<tr>
<td>23</td>
<td>0.20</td>
<td>1.02</td>
<td>51</td>
<td>0.12</td>
<td>0.50</td>
</tr>
<tr>
<td>24</td>
<td>0.20</td>
<td>1.02</td>
<td>20</td>
<td>0.06</td>
<td>0.29</td>
</tr>
<tr>
<td>25</td>
<td>0.20</td>
<td>1.02</td>
<td>20</td>
<td>0.24</td>
<td>0.60</td>
</tr>
</tbody>
</table>

that the ALS algorithm combined with an analytical model would replace the function of an expert in providing heuristic rules regarding a grinding process.

The reference surface roughness model for a surface grinding process has the following generalized functional form [12]:

\[
R_a = R_0 x_d^{-\frac{v_w}{v_s}} \left( \frac{v_w}{v_s} \right) \left( \frac{s_t}{b_s} \right)^\gamma ,
\]

where \( R_a \) is the surface roughness after dressing (\( \mu \)m), \( s_t \) is the dressing lead (mm), \( a_d \) is the dressing depth (\( \mu \)m), \( v_w \) is the work speed (m/s), \( v_s \) is the wheel speed (m/s), and \( s_t \) and \( b_s \) are the crossfeed (mm) and the wheel width (mm), respectively. \( R_0, x, y, z, \) and \( \gamma \) are the model coefficients whose typical values can be found from [15] as listed in Table 2. The actual values of these coefficients depend on the specific grinding process and must be determined by a large number of careful experiments. Since the objective of using the analytical model is to provide a rough functional form of the process input–output relationship to the initial construction of FBFN,
Table 2
Coefficients of the reference model for surface grinding processes

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Typical values</th>
<th>Values used for training</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_0$</td>
<td>—</td>
<td>45</td>
</tr>
<tr>
<td>$x$</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>$y$</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>$z$</td>
<td>$0.15 &lt; z &lt; 0.6$</td>
<td>0.375</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>$0.5 &lt; \gamma &lt; 1$</td>
<td>0.75</td>
</tr>
</tbody>
</table>

![Fig. 13. Results of training.](image)

The coefficients of the analytical model were determined by simply choosing median values in the ranges of typical values listed in Table 2. From the analytical equation, while $v_x$ and $b_y$ were fixed at 33 m/s and 25.4 mm, respectively, 625 uniformly distributed samples were generated within the following ranges of operating conditions, and used for training:

\[
0.06 \leq s_d \leq 0.24,
8 \leq a_d \leq 51,
0.1 \leq v_w \leq 0.3,
0.5 \leq s_t \leq 1.5.
\]

It should be noted that the above 4 variables correspond to input variables of the FBFN. The ALS algorithm generated six fuzzy rules, whose linear weights were then tuned by applying a linear least-squares algorithm to the FBFN for 25 sets of experimental data listed in Table 1.
Table 3
Experimental conditions for testing

<table>
<thead>
<tr>
<th>Test no.</th>
<th>Work speed, ( v_w ) (m/s)</th>
<th>Crossfeed, ( s_t ) (mm)</th>
<th>Dressing depth, ( a_d ) (µm)</th>
<th>Dressing lead, ( s_d ) (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.30</td>
<td>2.3</td>
<td>16</td>
<td>0.12</td>
</tr>
<tr>
<td>2</td>
<td>0.30</td>
<td>2.3</td>
<td>16</td>
<td>0.12</td>
</tr>
<tr>
<td>3</td>
<td>0.29</td>
<td>1.3</td>
<td>25</td>
<td>0.12</td>
</tr>
<tr>
<td>4</td>
<td>0.30</td>
<td>1.2</td>
<td>24</td>
<td>0.12</td>
</tr>
<tr>
<td>5</td>
<td>0.30</td>
<td>2.3</td>
<td>51</td>
<td>0.12</td>
</tr>
<tr>
<td>6</td>
<td>0.30</td>
<td>2.3</td>
<td>16</td>
<td>0.12</td>
</tr>
<tr>
<td>7</td>
<td>0.25</td>
<td>1.4</td>
<td>13.8</td>
<td>0.12</td>
</tr>
<tr>
<td>8</td>
<td>0.30</td>
<td>2.0</td>
<td>25.8</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Fig. 14. Comparison between measured surface roughness values and predicted outputs from the FBFN for 8 sets of test grinding conditions.

A comparison between measured surface roughness and outputs from the FBFN has been made for the 25 sets of training data as shown in Fig. 13, from which a good model fitting can be observed. In order to test prediction performance of the FBFN model, 8 sets of experiments were designed as listed in Table 3. The experiments were carried out twice, and the average and deviation of their measured values are shown in Fig. 14 along with the model outputs from the FBFN. It can be seen that prediction performance of the FBFN is excellent for all the tested cases. In order to show that the analytical model alone cannot be used to predict the surface roughness, the ranges of model outputs that can be expected from the analytical equation are calculated and listed in Table 4 based on the ranges of model coefficients listed in Table 2.
Table 4
Measured and predicted surface roughness values for operating conditions listed in Table 3

<table>
<thead>
<tr>
<th>Test no.</th>
<th>Measured value (µm)</th>
<th>Analytical model (µm)</th>
<th>FBFN (µm)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minimum</td>
<td>Maximum</td>
<td>Minimum</td>
</tr>
<tr>
<td>1</td>
<td>0.62</td>
<td>0.64</td>
<td>0.17</td>
</tr>
<tr>
<td>2</td>
<td>0.60</td>
<td>0.64</td>
<td>0.17</td>
</tr>
<tr>
<td>3</td>
<td>0.43</td>
<td>0.57</td>
<td>0.11</td>
</tr>
<tr>
<td>4</td>
<td>0.50</td>
<td>0.56</td>
<td>0.10</td>
</tr>
<tr>
<td>5</td>
<td>0.85</td>
<td>0.89</td>
<td>0.23</td>
</tr>
<tr>
<td>6</td>
<td>0.63</td>
<td>0.63</td>
<td>0.17</td>
</tr>
<tr>
<td>7</td>
<td>0.45</td>
<td>0.49</td>
<td>0.09</td>
</tr>
<tr>
<td>8</td>
<td>0.71</td>
<td>0.71</td>
<td>0.17</td>
</tr>
</tbody>
</table>

5. Conclusions

Neuro-fuzzy networks have attracted significant attention due to their learning abilities to construct fuzzy inference systems. Two novel algorithms for training of FBFNs have been proposed in this study. The LS algorithm provides a way of constructing a true fuzzy inference system in modeling nonlinear relationships with efficient computation, while the ALS algorithm provides true hybrid structure-parameter learning without human assistance by globally searching for an optimal fuzzy rule at each iteration using GA. Hence, either algorithm can be chosen depending on applications and abilities of computing hardware.

Simulation studies have shown that new algorithms generate superior results over conventional algorithms such as the orthogonal least-squares algorithm, backpropagation algorithm, and conventional GA-based algorithm. A comparison of computation time and approximation capability between the ALS algorithm and the conventional-GA based algorithm showed that the proposed ALS algorithm provides valuable advantages when the GA is applied to large-scale problems. The example of modeling a surface grinding process using a small amount of experimental data demonstrated the potential of the ALS algorithm in constructing fuzzy models of complex manufacturing processes efficiently.

It is noted that any evolutionary algorithm can work for the proposed ALS algorithm in place of the GA. The GA has been adopted in this study as an example of global search methods. It is expected that application of an evolutionary algorithm with a floating-point representation such as evolution strategies will lead to even better performance of the proposed methodology.

Acknowledgements

The financial support of this work provided by NSF MT-AMRI, Seagate and Purdue Research Foundation is gratefully acknowledged.

References