Soft Comput (2009) 13:1013–1025
DOI 10.1007/s00500-008-0375-z

ORIGINAL PAPER

Self-spawning neuro-fuzzy system for rule extraction

Zhi-Qiang Liu · Tao Guan · Ya-Jun Zhang

Published online: 23 May 2009

Abstract In this paper we propose self-spawning neuro-fuzzy system (SSNFS), a new neuro-fuzzy system to derive fuzzy rules from data. The SSNFS model is based on a generic definition of incremental perceptron and a new learning algorithm that is capable of both structural (rule) learning and parametric learning. It constructs the fuzzy system by detecting a suitable number of rule patches and their positions and shapes in the input space. Initially the rule base consists of one single fuzzy rule; during the iterative learning process the rule base expands according to a supervised spawning validity measure. The rule induction process terminates when a given stop criterion is satisfied. SSNFS is very general since it does not require the prior knowledge about the input space and can be used in any application based on the scatter-partitioning fuzzy system. To demonstrate the effectiveness and applicability of our algorithm, we present a synthetic example and real-world modelling problems.

Keywords Incremental clustering · Gaussian membership function · Rule extraction · Self-spawning competitive learning (SSCL) · Self-spawning neuro-fuzzy system (SSNFS)

Z.-Q. Liu (✉) · T. Guan · Y.-J. Zhang
School of Creative Media,
City University of Hong Kong,
Hong Kong, Hong Kong SAR
e-mail: zq.liu@cityu.edu.hk

T. Guan
e-mail: taoguan@cityu.edu.hk

Y.-J. Zhang
Department of Computer Science and Software Engineering,
The University of Melbourne, Melbourne, VIC 3010, Australia

1 Introduction

Fuzzy inference systems are able to model the continuous input/output relationships by means of fuzzy IF–THEN rules without employing precise quantitative analysis. This fuzzy modelling has been successfully used in many applications, e.g., to build function approximators (Pomares et al. 2000; Chen and Saif 2005), fuzzy controllers (Boukezzoula et al. 2006), fuzzy classifiers (Pal and Mitra 1992), and decision making. Researchers have developed many practical systems, such as prediction and inference (Kandel 1992), non-linear control (Sugeno 1985; Pedrycz 1989; Berenji and Khedkar 1992; Lin 1994; Jouffe 1998), signal processing and communication systems (Nie and Linkens 1994), type 2 fuzzy system model (Uncu and Türkşen 2007), just to mention a few.

The basic components in a fuzzy inference system are fuzzy rules. It is desirable that the rule base covers all the states of the system that are important to data understanding (Duch et al. 2001; Duch et al. 2004) and the intended application (Setnes 2000). In general, fuzzy rules can be obtained by either human experts or a data-driven extraction scheme from measured input–output data pairs. The latter case is currently a growing research topic, since in most cases people or decision makers in a data-mining project or industry applications are usually not trained statisticians, mathematicians, or AI experts. Moreover, in many commercial areas there is a huge number of unstructured data collections. Therefore, it is important to learn knowledge and derive fuzzy rules from the data itself (Cherkassky and Mulier 1998). Various methods have been proposed, e.g., fuzzy clustering in product space (Setnes 2000) or in augmented data set (Pal et al. 2002), rule generation using data condensation algorithm (Mitra et al. 2002), genetic algorithms (Lee and Takagi 1993; Ishibuchi

et al. 1995), entropy (Yager and Filev 1993), orthogonal transformation methods (Wang and Mendel 1992; Yen and Wang 1999), and a group of neuro-fuzzy approaches (Berenji and Khedkar 1992; Brown and Harris 1994; Gupta and Rao 1994; Jang 1993; Fritzke 1997), etc.

Over the last few years, research on combining neural networks and fuzzy systems, neuro-fuzzy systems, has gained considerable importance and attention (Mitra and Hayashi 2000; Jang and Sun 1995). The learning capabilities of neural networks give fuzzy systems the ability to tune the parameters and shapes of fuzzy membership functions (MFs). There are several paradigms to combine neural networks and fuzzy systems: concurrent neural/fuzzy models, cooperative neuro-fuzzy models, and hybrid neuro-fuzzy models (see Kruse and Nauck 1995 for details). Among these approaches, the hybrid neuro-fuzzy models are the most popular forms in the modern neuro-fuzzy systems. In general, a hybrid neuro-fuzzy system can be viewed as a special *n*-layer feed-forward neural network (Nauck 1997) with sampled fuzzy memberships as the input/output (Keller and Tahani 1992; Keller et al. 1992), or with parameterized MFs stored in the neurons (Jang 1993; Berenji and Khedkar 1992), or with fuzzy sets as the link weights (Nauck 1997). In fact, few neuro-fuzzy approaches actually employ true neural networks, although they are very often depicted in the form of some kind of neural network structure which exhibits some learning capability (Nauck 1997). With this learning capability, we are able to determine the fuzzy sets or fuzzy rules with an iterative process. Learning in a neuro-fuzzy system normally involves two phases: structural learning and parametric learning (Jouffe 1998; Rojas et al. 2000). The structure learning tunes a number of rules, thus it is also often referred to as *rule learning*, whereas the parametric learning tunes the positions of fuzzy sets. For some of the neural-fuzzy systems in the literature (e.g., Berenji and Khedkar 1992; Jang 1993), there is no rule learning procedure defined. Therefore, once the neural network architecture has been determined, the number of rules is assumed fixed during the learning process. Such a process is often not effective to yield good (i.e., small and interpretable) rule bases. In this case, we may consider pruning techniques to reduce the number of rules and variables in the neuro-fuzzy system (Nauck 1997; Zimmermann et al. 1996). This usually starts from a *large* number of rules, during the learning process some of the rules are *pruned* (rule deduction) resulting in a set of rules that are most relevant to the problem at hand. However, it is difficult in most cases to choose a reasonably *large* number of rules, because we simply do not have enough knowledge about the data. This can lead to poor system performance in real-world applications. In the last few years incremental learning (rule induction) is getting to be another solution. The adaptive nature of incremental learning requires it to be implemented in the on-line fashion (Fritzke 1997). A serious problem with such learning algorithms is that they do not have a criterion to terminate the growth process of the neural network structure: the termination is judged by the human satisfactory degree on the learning performance and a pre-defined maximum network size. Furthermore, the shape adaption of a MF is based on a simple calculation between two closest rule patches in the input space, e.g., half of their distance. This is not suitable for rule patches with different sizes, as in this case the memberships for smaller rule patches may overlap significantly with parts of larger rule patches, leading to misclassification in data analysis.

In this paper, we introduce a new incremental, hybrid neuro-fuzzy system using the supervised self-spawning competitive learning (SSSCL) paradigm, self-spawning neuro-fuzzy system (SSNFS). It is a scatter-partitioning fuzzy inference system that allows the IF-parts of the fuzzy rules to be positioned at arbitrary locations in the input space. A major problem with scatter partitions is to find a suitable number of rules, suitable positions and suitable width of the rule patches in the input space (Fritzke 1997). Our neuro-fuzzy model will focuses on this problem and is able to incrementally and adaptively build up a network structure during the training process. It requires only a single rule prototype randomly initialized in the input space; during the training process the rule base expands adaptively according to a split validity measure to discover more rule patches. The shape width of the rule patch is indicated by an on-line property vector. The rule induction process terminates when a stop criterion is satisfied. The output weights are trained in the on-line manner, this is more appropriate (computationally less complex) than to repeatedly solve the complete linear system with matrix manipulation as that in batched learning. To extract rules from data, we assume that a limited number of input–output pairs are provided on-line for single-pass processing.

The rest of this paper is organized as follows. We first give a brief analysis on the scatter-partitioning fuzzy inference system in Sect. 2. In Sect. 3 we describe the detailed neuro-fuzzy modelling and the self-spawning learning algorithm. Section 4 illustrates the experimental study and simulations, and Sect. 5 presents discussions and conclusions.

2 Scatter-partitioning fuzzy systems

2.1 Takagi–Sugeno fuzzy model

The problem we are going to solve is to design a partitioning fuzzy inference system from input–output data for

classification analysis. The basic component of a fuzzy inference system is the fuzzy rule which is expressed using linguistic labels, for instance,

$$\begin{aligned}&\text{IF } (x_1 \text{ is low}) \text{ AND } (x_2 \text{ is medium})\\ &\quad \text{THEN } (y_1 \text{ is } 0.2) \text{ AND } (y_2 \text{ is } 0.4),\end{aligned} \tag{1}$$

where $x_1,\ x_2 \in \mathcal{R}$ is the input variables (antecedent) and $y_1,\ y_2 \in \mathcal{R}$ is the output (consequent) of this rule. The linguistic labels (e.g., low) are usually modelled as parameterized MFs within a particular area of the input space. AND is the T-norm fuzzy operator and in some cases OR (T-conorm) is used in fuzzy rules. In the example shown above, the fuzzy sets involved only in the premise part (IF-part). Fuzzy systems consists of this kind of rules are referred to as *Takagi–Sugeno fuzzy systems*. Moreover, the THEN-parts consists of only crisp values, e.g., 0.2 and 0.4, this model is called as *zero-order Sugeno fuzzy model*. In an *n*th-order Sugeno fuzzy system the THEN-part of each rule consists of a polynomial of degree n in the input variables (Fritzke 1997).

In this paper we will concentrate on zero-order Sugeno fuzzy systems since they have the interesting property of being equivalent to radial basis function networks (RBFNs) and have been used for data analysis in many applications. Different shapes of the MFs have been proposed such as the Gaussian, triangular, or trapezoidal. In the following discussions we assume that the MFs have the form of the Gaussian function and that the fuzzy system consists of m fuzzy rules each with n input variables and is designed to classify data into k fuzzy or crisp classes. Thus, the fuzzy system can be described as $\{R_i\}_{i\in\{1,\ldots,m\}}$. For the ith rule R_i,

$$\begin{aligned}\mathcal{R}_i : &\text{IF } (x_1 \text{ is } g_{i_1}(x_1)) \text{ AND } (x_2 \text{ is } g_{i_2}(x_2))\\ &\quad \text{AND } \cdots \text{ AND } (x_n \text{ is } g_{in}(x_n))\\ &\text{THEN } (y_1 \text{ is } t_{i_1}) \text{ AND } (y_2 \text{ is } t_{i_2})\\ &\quad \text{AND } \cdots \text{ AND } (y_k \text{ is } t_{i_k}),\end{aligned} \tag{2}$$

where $g_{ij}(x_j)$ is the MF denoting the linguistic label associated with the jth input variable in the ith fuzzy rule,

$$g_{ij}(x_j) = \exp\left[-\frac{(x_j - p_{ij})^2}{2\sigma_{ij}^2}\right], \tag{3}$$

where the variable p_{ij} and σ_{ij} are the center and variance of the Gaussian MF g_{ij}, respectively. $\mathbf{T}_i = [t_{i_1}, \ldots, t_{i_k}]^{\mathrm{T}}$ is the output vector for the ith fuzzy rule, $t_{ij} \in \{0, 1\}$ for partitioning crisp clusters or function approximation and $t_{ij} \in [0, 1]$ for partitioning fuzzy clusters.

Usually only a moderate number of MFs is defined for each input variable. It is always desirable that the membership values for each input variable add to unity everywhere. This may be achieved by dividing the membership values $g_{ij}(x_j)$ by the sum of all memberships with respect to x_j, leading to *normalized* MFs,

$$\widehat{g}_{ij}(x_j) = g_{ij}(x_j) \Big/ \sum\nolimits_{l=1}^{m} g_{lj}(x_j). \tag{4}$$

We focus our attention on rules that combine their subexpressions by fuzzy AND. In the case of Gaussian MFs, the fuzzy AND can be realized by the arithmetic product. Let G_i denote the membership value for the IF-part of R_i in Eq. 2,

$$G_i = \prod_{j=1}^{n} g_{ij}(x_j). \tag{5}$$

In this case the IF-part of each fuzzy rule can be described by an n-dimensional Gaussian MF,

$$G_i(\mathbf{X}) = \exp\left(-\frac{1}{2}(\mathbf{X} - \mathbf{P}_i)^{\mathrm{T}}\Lambda^{-1}(\mathbf{X} - \mathbf{P}_i)\right), \tag{6}$$

where $\mathbf{X}$ is the multivariate input vector $[x_1, x_2, \ldots, x_n]^{\mathrm{T}}$, $\mathbf{P}_i = [p_{i_1}, p_{i_2}, \ldots, p_{i_n}]^{\mathrm{T}}$ is the n-dimensional Gaussian center that has the corresponding centers of the one-dimensional factor Gaussians as components. $\Lambda^{-1} = \mathrm{diag}(1/\sigma_{i_1}^2, 1/\sigma_{i_2}^2, \ldots, 1/\sigma_{i_n}^2)$ is the inverse of the covariance matrix Λ corresponding to the product of n one-dimensional Gaussian memberships. Similarly, we can normalize the multivariate Gaussian MFs as follows,

$$\widehat{G}_i(\mathbf{X}) = G_i(\mathbf{X}) \Big/ \sum\nolimits_{l=1}^{m} G_l(\mathbf{X}). \tag{7}$$

For the input pattern $\mathbf{X}$, the output of the ith component of a fuzzy system with normalization is given by

$$O^{(i)}(\mathbf{X}) = \sum_{j=1}^{m} t_{ji}\widehat{G}_j(\mathbf{X}) = \sum\nolimits_{j=1}^{m} t_{ji}G_j(\mathbf{X}) \Big/ \sum\nolimits_{j=1}^{m} G_j(\mathbf{X}). \tag{8}$$

2.2 Scatter-partitioning fuzzy systems

Based on Eq. 6, we may simplify the fuzzy rule in Eq. 2 as

$$\mathcal{R}_i : \text{ if } \mathbf{X} \text{ is } G_i(\mathbf{X}) \text{ then } \mathbf{Y} \text{ is } \mathbf{T}_i, \tag{9}$$

where $\mathbf{Y} = \{y_1, y_2, \ldots, y_k\}$ is the multivariate output variable and T_i is the consequent output vector for ith rule.

Since each Gaussian MF $G_i,\ i \in \{1, \ldots, m\}$ covers a particular area in the input space, each rule is considerably activated only in this area. One can think of the input space as being covered by some small patches, each of them corresponding to the IF-part of one fuzzy rule (Fritzke 1997). We denote these small patches as *rule patches* in this paper. Building a meaningful fuzzy inference system to a large extent is to establish a series of fuzzy rules that cover all the rule patches in the input space. At the same time, we must keep the number of rules as low as possible in order to maintain the generalizing ability of the model and to ensure a compact and transparent model (Setnes 2000). Therefore,

our task is reduced to seeking an optimal number of rule patches, locating their positions, and computing their shape widths in the input space. This is known as *scatter-partitioning* distinguished from *grid-partitioning* approaches in which the rule patches are assumed on a regular predefined grid distribution. In a scatter-partitioning fuzzy system the fuzzy memberships are not confined to the corners of a rectangular grid. Rather, they can activate anywhere with any width in the input space as long as the $\mathbf{P}$ and Λ in Eq. 6 can be detected. If the covariance matrices Λ_i of the Gaussians are diagonal, then they can still be thought of being generated as a product of n one-dimensional Gaussian MFs. Figure 1 shows an example of grid-partitioning and scatter-partitioning fuzzy systems. Once the IF-parts are solved, the output vector $\mathbf{T}$ for each rule may be achieved by minimizing the summed squared error,

$$E = \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{m}\left[\mathbf{Y}_i - \widehat{G}_j(\mathbf{X}_i)\mathbf{T}_j\right]^2, \tag{10}$$

where $(\mathbf{X}_i, \mathbf{Y}_i)$ is the training pattern, and M is the total number of patterns presented for training. This approach is usually computationally complex. In fact, the output vector can be learned on-line concurrently in the output space to reduce the complexity, such as training by the delta-rule (Fritzke 1997; Nauck and Kruse 1995).

Now, the problem is how to correctly classify the rule clusters in the input space. The conventional learning algorithms using neural networks or fuzzy clustering are very sensitive to the number of prototypes initialized. They perform reasonably well when the number of prototypes has been chosen appropriately; the best case is when the number of prototypes equals to that of the natural clusters and with properly positioned initial prototypes. We, however, do not usually have an adequate prior knowledge about the data set. Therefore it is desirable to develop an algorithm that is able to adaptively detect the number of rule patches and their locations. In this paper we consider an incremental learning paradigm which first constructs a fuzzy system by seeking one fuzzy rule, and conducting rule induction in an iterative learning process according to a self-spawning validity measure. The rule growth is terminated when all the rule patches have been detected based on a stop criterion. The algorithm is implemented as an on-line competitive learning process. Therefore, we may consider the scatter-partitioning fuzzy systems as a neuro-fuzzy system.

3 The SSNFS neuro-fuzzy modeling

In general, neuro-fuzzy modelling includes two phases: building a neural network to interpret the fuzzy rules, and employing a learning algorithm to update fuzzy perceptrons or the structure itself. We consider neuro-fuzzy modelling as a technique to derive a fuzzy system from data, or to enhance it by learning from examples.

Many neuro-fuzzy systems use parameterized MFs that are stored within the "neurons" of a multilayered feed-forward architecture, e.g., GARIC (Berenji and Khedkar 1992) and ANFIS (Jang 1993). The links in these kinds of networks indicate only the data flow directions between nodes and no weights are associated with the links. The membership functions are usually parameterized Gaussians, trapezoidal, or triangular functions, or they are constructed by superposing sigmoids. NEFCLASS (Nauck 1997; Nauck and Kruse 1995) builds a different structure which uses sampled memberships (fuzzy sets) as link weights. For a quick review, Fig. 2a shows a typical ANFIS architecture and Fig. 2b shows an example of NEFCLASS structure. It can be seen that, either in Fig. 2a or b, each input variable is associated with a specific MF in its input area. This, however, brings difficulty to the structural learning. For instance, in Fig. 2a, a new linguistic label (membership function) will cause the network to learn the connection relationships from both the input to layer 1 and the input from layer 1 to layer 2.

It must be clear that the above mentioned neuro-fuzzy models (and in fact almost all neuro-fuzzy models in the literature) have little to do with fuzzy logic in the narrow sense (Kruse and Gebhardt 1994) but to do with the parameterized MFs in the fuzzy IF-THEN rules which are associated with linguistic labels.

3.1 SSNFS architecture

As discussed in Sect. 2, each fuzzy rule is associated with one parameterized Gaussian membership with multivariate inputs and one output vector, as seen in Eq. 9. The Gaussian MFs can be identified by detecting rule patches in the input space, whereas the output vectors can be updated on-line in the output space. In this perspective, we propose SSNFS based on a generic incremental perceptron and a new learning algorithm, *self-spawning competitive learning*, to incrementally search the rule patches.

Figure 3 shows an example of SSNFS architecture and its fuzzy reasoning. Similar to ANFIS, the MFs are stored in the hidden nodes to model continuous input relationships. However, the difference is, the new model uses a multivariate membership function for each rule, thus the node with this MF is fully connected to all the input variables. Also, in SSNFS, the parameters for each MF are modelled as the link weights between the input nodes and membership nodes, and the consequent vector of each rule is modelled as the weights between the input nodes and rule nodes. The benefit is, the weight vectors can be adaptively updated given each input training pattern. Third,

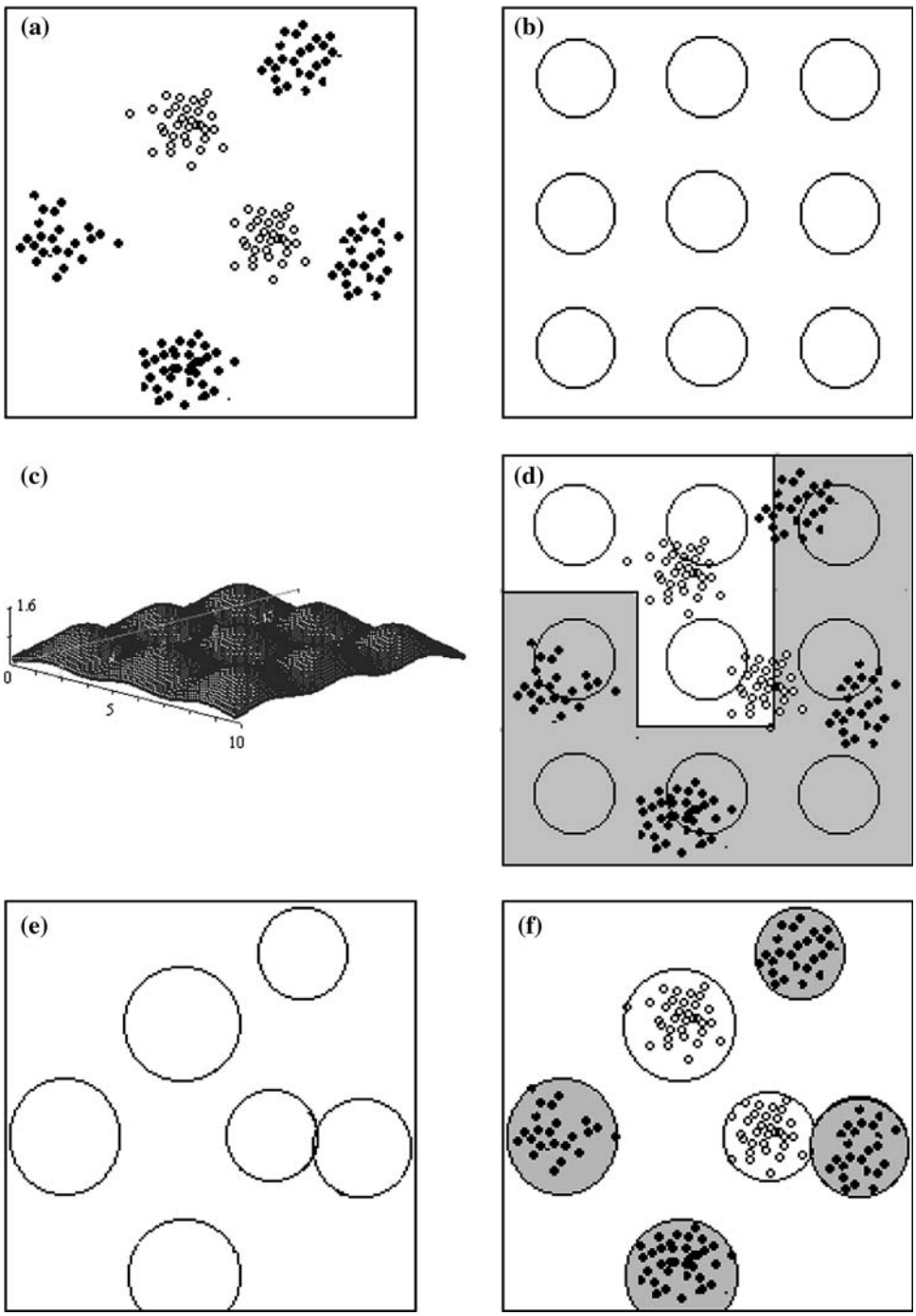

Fig. 1 TClassfication example with a grid-partitioning fuzzy system and a scatter-partitioning fuzzy system. **a** Two dimensional dataset in the unit square consisting of two classes (*white* and *black*). *b* The grid-partitioning system initializes 3 × 3 grid to arrange nine fuzzy rules. **c** The non-normalized Gaussian view in the input space for this grid-partitioning system. **d** The classification result by the grid-partitioning system. **e** The scatter-partitioning system captured six rule patches in the input space. **f** The classification result by the scatter-partitioning system

different from network models with pre-specified structures, the SSNFS is not a static model, but dynamically adjusted according to the distribution of patterns. This means that the number of fuzzy rules will change with the that of the nodes in layer 4. The dynamical learning scheme, called structure learning, makes SSNFS have better adaptivity than models with fixed structure.

To build a neuro-fuzzy system for extracting fuzzy linguistic rules, we present a *generic five-layer incremental perceptron* that will provide a basis for SSNFS architecture and self-spawning learning algorithm.

Definition 1 A generic five-layer incremental perceptron is a five-layer feed-forward neural network (U, W, P, T, NET, A, O, ex, Inc) with the following specifications:

1. $U = \bigcup_{i \in I} U_i$ is a non-empty set of neurons and $I = \{1, \ldots, 5\}$ is the index set of U. For all $i, j \in I$, $U_i \neq \emptyset$ and $U_i \cap U_j = \emptyset$ for $i \neq j$. $U_1 = U_1^{(\mathcal{X})} \bigcup U_1^{(\mathbf{X})} \bigcup U_1^{(\mathbf{Y})}$ is

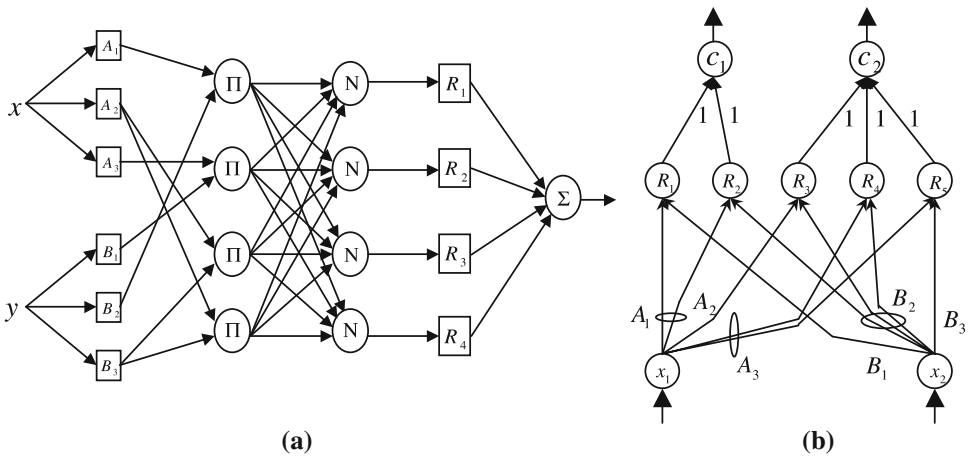

Fig. 2 An example of the ANFIS and NEFCLASS architectures. **a** The ANFIS structure with two antecedent variables, four fuzzy rules and one consequent output; the membership functions (*MFs*) are stored in the neurons and the link connections only indicate the data flow. **b** A NEFCLASS system with two inputs, five rules and two output classes; the MFs are modeled as the link weights. In both cases, each one-dimensional antecedent variable is associated with one MF in its input area

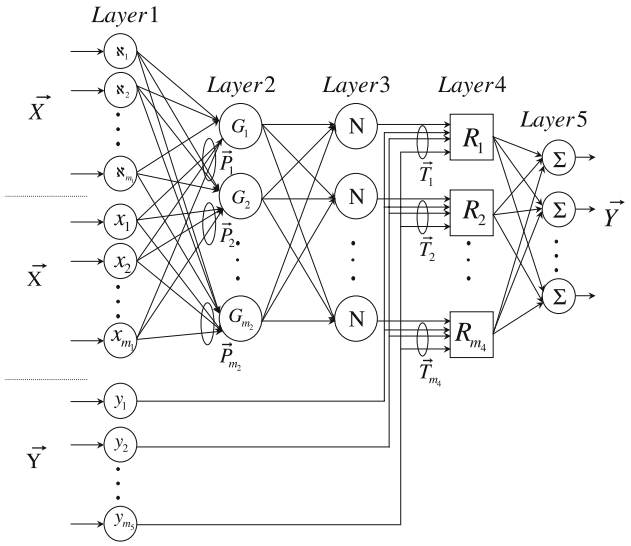

Fig. 3 An example of the SSNFS architecture. In this case, the MFs are generalized as multivariate parameterized functions stored in the hidden neurons. Either their parameters or the consequent output vectors are modelled as the link weights

the input layer, U_2, U_3 and U_4 are the hidden layers, and U_5 is the output layer.

2. Let $M = \{2m_1 + m_5,\ m_2,\ m_3,\ m_4,\ m_5\}$ define the number of neurons for layer 1 to layer 5, respectively; m_1 is the number of neurons for either $U_1^{(\mathcal{X})}$ or $U_1^{(\mathbf{X})}$; $m_2 = m_3 = m_4$ is the number of neurons for each hidden layer; m_5 is the number of neurons for $U_1^{(\mathbf{Y})}$ or U_5.
3. W defines the link connectedness in the perceptron as follows:
   (a) for $u_i \in U_1^{(\mathcal{X})} \bigcup U_1^{(\mathbf{X})}$ and $v_j \in U_2$, $W(u_i, v_j) = 1$;
   (b) for $u_i \in U_1^{(\mathbf{Y})}$ and $v_j \in U_4$, $W(u_i, v_j) = 1$;
   (c) for $u_i \in U_2$ and $v_j \in U_3$, $W(u_i, v_j) = 1$;
   (d) for $u_i \in U_3$ and $v_j \in U_4$ and $i = j$, $W(u_i, v_j) = 1$;
   (e) for $u_i \in U_4$ and $v_j \in U_5$, $W(u_i, v_j) = 1$;

   otherwise, $W(u_i, v_j) = 0$.
4. $P = \{\mathbf{P}_1, \ldots, \mathbf{P}_{m_2}\}$ defines the weight vectors (prototypes) on the connections $W(U_1^{(\mathbf{X})},\ U_2)$. The ith prototype can be given by $\mathbf{P}_i = [p_{i_1}, \ldots, p_{i_{m_1}}]^{\mathrm{T}}$ where p_{ij} is the weight on the connection $W(u_j, v_i)$ with $u \in U_1^{(\mathbf{X})}$ and $v \in U_2$. $T = \{\mathbf{T}_1, \ldots, \mathbf{T}_{m_2}\}$ defines the weight vectors on the connections $W(U_1^{(\mathbf{Y})},\ U_4)$. $\mathbf{T}_i = [t_{i_1}, \ldots, t_{i_{m_5}}]^{\mathrm{T}}$ where t_{ij} is the weight on the connection $W(u_j, v_i)$ with $u \in U_1^{(\mathbf{Y})}$ and $v \in U_4$.
5. NET defines the propagation function for each unit $u \in U$ to calculate the net input net_u.
   (a) for $u \in U_1$:
   $$\mathrm{NET}_{u_i} : \mathcal{R} \mapsto \mathcal{R},\ \mathrm{net}_{u_i} = ex_{u_i};$$
   (b) for $u \in U_2$:
   $$\mathrm{NET}_{u_i} : \mathcal{R}^{m_1} \mapsto \mathcal{R}^{m_1},\ \mathrm{net}_{u_i} = O(U_1^{(\mathcal{X})}) - [o_{v_1} p_{i_1}, o_{v_2} p_{i_2}, \ldots, o_{v_{m_1}} p_{i_{m_1}}]^{\mathrm{T}},\quad v \in U_1^{(\mathbf{X})};$$
   (c) for $u \in U_3$:
   $$\mathrm{NET}_{u_i} : \mathcal{R}^{m_2} \mapsto \mathcal{R},\ \mathrm{net}_{u_i} = \frac{o_{v_j}}{\sum_{j=1}^{m_2} o_{v_j}},\quad v \in U_2;$$
   (d) for $u \in U_4$:
   $$\mathrm{NET}_{u_i} : \mathcal{R} \times \mathcal{R}^{m_5} \mapsto \mathcal{R}^{m_5},\ \mathrm{net}_{u_i} = [o_{v_i} t_{i_1}, o_{v_i} t_{i_2}, \ldots, o_{v_i} t_{i_{m_5}}]^{\mathrm{T}},\quad v \in U_3;$$
   (e) for $u \in U_5$:
   $$\mathrm{NET}_{u_i} : \mathcal{R}^{m_4} \mapsto \mathcal{R},\ \mathrm{net}_{u_i} = \sum_{j=1}^{m_4} o_{v_j},\quad v \in U_4.$$
6. A defines the activation function for each $u \in U$ to calculate the activation a_u.
   (a) for $u \in U_1 \bigcup U_3 \bigcup U_4 \bigcup U_5$:
   $$A_{u_i} : a_{u_i} = A_{u_i}(\mathrm{net}_{u_i}) = \mathrm{net}_{u_i};$$
   (b) for $u \in U_2$:

$$A_u : \mathcal{R}^{m_1} \mapsto \mathcal{R}, \quad a_u = A_u(\text{net}_u),$$

where A_u is the parameterized activation function.

7. O defines for each $u \in U$ an output function O_u to calculate the output o_u.
   (a) for $u \in U_1 \bigcup U_2 \bigcup U_3 \bigcup U_4$:
   $$o_{u_i} = O_{u_i}(a_{u_i}) = a_{u_i};$$
   (b) for $u \in U_5$:
   $$O_{u_i} : \mathcal{R} \mapsto \{0, 1\} \text{ (for fuzzy partitioning or function approximation)},$$
   $$o_u = O_u(a_u) = a_u;$$
   $$O_{u_i} : \mathcal{R} \mapsto [0, 1] \text{ (for crisp partitioning)},$$
   $$o_u = O_u(a_u) = DF(a_u),$$
   where DF is a suitable defuzzification function.
8. ex defines for each input unit $u \in U_1$ its external input $\text{ex}(u) = \text{ex}_u$. For all other units ex is not defined.
9. *Inc* defines the policy for the perceptron increment. A spawning request will be carried out by taking the following actions:
   (a) One neuron is spawned for each hidden layer, while the input and output layers remain unchanged. Let u, v and w indicate the new added neuron for layer 2, 3, and 4, respectively.
   $$U_2 := U_2 \bigcup \{u\}, \quad m_2 := m_2 + 1;$$
   $$U_3 := U_3 \bigcup \{v\}, \quad m_3 := m_3 + 1;$$
   $$U_4 := U_4 \bigcup \{w\}, \quad m_4 := m_4 + 1;$$
   $m_2 = m_3 = m_4$ holds anytime.
   (b) for all U_1, U_2, U_3, U_4, and U_5, the structure are reconstructed following the definition of (U, W, P, T, NET, A, O, ex).

Given the definition of the generic perceptron, we can describe the SSNFS as follows.

Definition 2 A SSNFS system is a generic five-layer incremental perceptron employing supervised SSCL algorithm with the following specifications:

1. In SSNFS, $U_1^{(\mathcal{X})}$ is the input layer for data classification; $U_1^{(\mathbf{X})}$ and $U_1^{(\mathbf{Y})}$ are the input layers for training; U_2 is the membership layer; U_3 is the normalized membership layer; U_4 is the rule layer and U_5 is the output layer.
2. SSNFS starts from a single neuron for each hidden layer, $m_2 = m_3 = m_4 = 1$ holds initially.
3. SSNFS assigns each prototype $\mathbf{P}_i$ with three property vectors ($\mathbf{A}_i$, $\mathbf{C}_i$, $\mathbf{R}_i$), called asymptotic property vector (APV), center property vector (CPV), and distant property vector (DPV), respectively. In a simulated neural network, it is achieved by assigning each p_{ij} three other weights, while for a real neural network we can simply set $W(u_i, v_j) = 4$ where $u \in U_1^{(\mathbf{X})}$ and $v \in U_2$.
4. SSNFS applies Gaussian MFs or triangular MFs as the activation functions for layer 2; **P** and its property vectors are the parameter vectors for MFs and **T** is the consequent output vector for a rule.
5. SSNFS switches between the training process and classification task as following:
   (a) for training process, $\text{ex}_u = \text{N/A}$, for $u \in U_1^{(\mathcal{X})}$; $U_1^{(\mathbf{X})} \bigcup U_1^{(\mathbf{Y})}$ is the active input layer;
   (b) for classification task, $ex_v = 1$, for $v \in U_1^{(\mathbf{X})} \bigcup U_1^{(\mathbf{Y})}$; $U_1^{(\mathcal{X})}$ is the active input layer.

In the SSNFS model, we model fuzzy rule extraction as a task to spawn the structure and to establish the premise and consequent parameters in the training process.

3.2 Supervised self-spawning competitive learning

As defined above, SSNFS employs SSSCL algorithm to tune the structure and parameters. Compared to pruning techniques, it is an incremental learning process. Thus it has to face with the one prototype takes multi-clusters (OPTMC) problem. That is, when the number of initial prototypes is less than that of the actual clusters in the data set, the conventional competitive learning process ensures there must be at least one prototype that wins patterns from more than one cluster after the learning process. Unfortunately, this behavior is not desirable in a neuro-fuzzy system as the rules extracted will give missing representation of the input space. As shown in Fig. 4a, suppose there are three rule patches in the input space and one prototype is initialized to detect them, by the conventional competitive learning paradigm, this single prototype moves to the center of the training patterns. As a result, the extracted fuzzy rule covers a wrong rule patch leading to misclassification. In fuzzy systems, or for that matter any

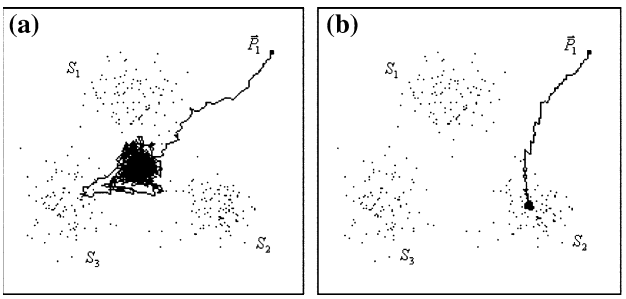

Fig. 4 Two learning behaviors: OPTMC versus OPTOC. **a** OPTMC: one prototype $\mathbf{P}_1$ is trying to take all three patches $\{S_1, S_2, S_3\}$, resulting in oscillation phenomenon. **b** OPTOC: this prototype detects only one rule patch S_2 and ignores the other two

rule-based systems, it is desirable that the extracted fuzzy rules represent the true rule patches.

The SSSCL tackles this problem by designing a new learning scheme in which one prototype takes one cluster (OPTOC) and ignores the others when the number of prototypes is less than that of the natural clusters, or in this case, the true rule patches. Figure 4b shows the same example as that in (a) but applying the *OPTOC* learning paradigm. In this case, one rule patch (cluster) is correctly detected, and the rest rule patches can be detected by spawning new prototypes in subsequent learning rounds. In this way we will be able to extract a correct set of rules from the data set.

In the neural-fuzzy network, the spawning of new prototypes is accomplished by expanding the SSNFS structure according to a spawning validity measure and the increment policy. The newly spawned prototype is hence to extract one more fuzzy rule in the next iterative learning process. This growth process terminates when a stop criterion is satisfied. The process is *supervised* in that the split validity measure is based on the desired output patterns for the given training patterns.

Let $\mathbf{P}_i(\mathbf{A}_i, \mathbf{C}_i, \mathbf{R}_i)$ denote the ith prototype in terms of its three property vectors; $(\mathbf{X}, \mathbf{Y})$ denote the pair of training patterns for input layer, where $\mathbf{X}$ is a m_1-dimensional pattern in the input space, and $\mathbf{Y}$ is a desired m_5-dimensional pattern in the output space; E_i denote the regression error and $\mathbf{T}_i$ the desired output vector for the fuzzy rule extracted by $\mathbf{P}_i$. Each time when a pair of patterns, $(\mathbf{X}, \mathbf{Y})$, is randomly picked from the training set, the competition occurs among all the current prototype vectors. The winning prototype is judged by the nearest neighbor criterion.

3.2.1 The SSSCL OPTOC learning algorithm

1. SSNFS starts from one fuzzy rule, therefore, $m_2 = m_3 = m_4 = 1$ holds initially. The only single prototype vector, $\mathbf{P}_1$, is initialized randomly in the input space. Its APV ($\mathbf{A}_1$) and CPV ($\mathbf{C}_1$) are initialized at a random location far from $\mathbf{P}_1$, whereas its DPV ($\mathbf{R}_1$) is set at the same place as $\mathbf{P}_1$. $\mathbf{T}_i$ is randomly initialized in the output space and $\mathbf{E}_i$ is initialized to 0.
2. For the input $(\mathbf{X}, \mathbf{Y})$, if $\mathbf{P}_i$ is the winning prototype for $\mathbf{X}$, its APV ($\mathbf{A}_i$) is updated in the input space by

$$\mathbf{A}_i^* = \mathbf{A}_i + \frac{1}{n_{\mathbf{A}_i}} \cdot \delta_i \cdot (\mathbf{X} - \mathbf{A}_i) \cdot \Theta(\mathbf{P}_i, \mathbf{A}_i, \mathbf{X}). \tag{11}$$

where

$$\Theta(\boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{\omega}) = \begin{cases} 1 & \text{if } |\boldsymbol{\mu}\boldsymbol{\nu}| \geq |\boldsymbol{\mu}\boldsymbol{\omega}|, \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

and

$$\delta_i = \left(\frac{|\mathbf{P}_i\mathbf{A}_i|}{|\mathbf{P}_i\mathbf{X}| + |\mathbf{P}_i\mathbf{A}_i|} \right)^2. \tag{13}$$

$n_{\mathbf{A}i}$ is the winning counter and $|\mathbf{uv}|$ is the Euclidean distance between a vector $\mathbf{u}$ and a vector $\mathbf{v}$. δ is the adaptively updated learning rate which satisfies $0 < \delta_i \leq 1$.

3. The DPV ($\mathbf{R}_i$) always follows the farthest pattern for which $\mathbf{P}_i$ has been the winner in the input space so far. For the input $(\mathbf{X}, \mathbf{Y})$, if $\mathbf{P}_i$ is the winning prototype for $\mathbf{X}$, $\mathbf{R}_i$ is updated by

$$\mathbf{R}_i^* = \mathbf{R}_i \cdot (1 - \Theta(\mathbf{P}_i, \mathbf{X}, \mathbf{R}_i)) + \mathbf{X} \cdot \Theta(\mathbf{P}_i, \mathbf{X}, \mathbf{R}_i). \tag{14}$$

4. For the input $(\mathbf{X}, \mathbf{Y})$, if $\mathbf{P}_i$ is the winning prototype for $\mathbf{X}$, the following update scheme guarantees $\mathbf{P}_i$ to cluster one rule patch and ignore the others in the input space.

$$\mathbf{P}_i^* = \mathbf{P}_i + \alpha_i \cdot (\mathbf{X} - \mathbf{P}_i), \tag{15}$$

where α_i is computed with

$$\alpha_i = \left(\frac{|\mathbf{P}_i\mathbf{A}_i|}{|\mathbf{P}_i\mathbf{X}| + |\mathbf{P}_i\mathbf{A}_i|} \right)^2 \quad (0 < \alpha_i \leq 1). \tag{16}$$

The OPTOC scheme enables each prototype to find only one natural rule patch in the input space when the patches are more than the prototypes. This in itself is a major improvement over other competitive learning algorithms in the literature. However, at this stage we are still not sure whether there are other rule patches that have not been detected yet. For this we introduce a spawning validity measure to judge if all the rule patches have been properly discovered. If not yet, SSNFS expands its structure by spawning and appending one neuron for each hidden layer, then reconstruct the architecture. The prototype vector of the newly spawned neuron in U_2 is hence to join the competition in the next iterative learning process. Based on the *OPTOC* learning scheme, it is therefore able to extract one more fuzzy rule.

Definition 3 (The SSSCL spawning validity measure and stop criteria)

1. For the input $(\mathbf{X}, \mathbf{Y})$, if $\mathbf{P}_i$ is the winning prototype for $\mathbf{X}$, $\mathbf{C}_i$ and $\mathbf{T}_i$ are updated on-line by the k-Means learning scheme (MacQueen 1967),

$$\mathbf{C}_i^* = \mathbf{C}_i + \frac{1}{n_{\mathbf{C}_i}}(\mathbf{X} - \mathbf{C}_i); \quad \mathbf{T}_i^* = \mathbf{T}_i + \frac{1}{n_{\mathbf{T}_i}}(\mathbf{X} - \mathbf{T}_i). \tag{17}$$

Just as the name k-Means indicates, the CPV ($\mathbf{C}_i$) indicates the arithmetic means (centroid) of all the X's

and $\mathbf{T}_i$ indicates the arithmetic centroid of all the Y's, where ($\mathbf{X}$, $\mathbf{Y}$)'s are the presented patterns for which $\mathbf{P}_i$ has been the winner.

2. E_i denotes the regression error for the fuzzy rule extracted by $\mathbf{P}_i$. Therefore, it is given by

$$E_i^* = E_i + (O_i^{(\mathbf{X})} - \mathbf{Y})^2, \tag{18}$$

where O_i represents the output vector of the fuzzy rule extracted by $\mathbf{P}_i$ as the if-parts and $\mathbf{T}_i$ as the then-parts.

3. For all $i \in \{1, ..., m_2\}$, if $|\mathbf{P}_i\mathbf{A}_i| < \varepsilon$ and there is at least one prototype $\mathbf{P}_j$ satisfies $|\mathbf{P}_j\mathbf{C}_j| > \varepsilon$, then SSNFS is suitable for spawning. ε is a small positive constant which is theoretically 0. The self-spawning process is carried by:

   (a) SSNFS expands its structure in terms of the policy *Inc* defined in the Definition 1.

   (b) Let $\mathbf{P}_{m_2}$ denote the prototype vector for the newly spawned neuron in U_2. It is initialized with

   $$\mathbf{P}_{m_2} = \mathbf{R}_s$$

   where the index s satisfies

   $$s = \arg\min_j E_j|_{|\mathbf{P}_j\mathbf{C}_j| > \epsilon}$$

   set $\mathbf{R}_{m_2} = \mathbf{P}_{m_2}, \mathbf{A}_{m_2} = \mathbf{C}_{m_2}$ at a random location in the input space, $E_{m_2} = 0$, and T_{m_2} at a random location in the output space.

4. For all $i \in \{1, ..., m_2\}$, if $|\mathbf{P}_i\mathbf{A}_i| < \varepsilon$ and $|\mathbf{P}_i\mathbf{C}_i| < \varepsilon$, SSNFS finishes its structural and parametric learning and can be interpreted as a set of established fuzzy rules.

SSNFS adaptively updates its structure and weight parameters to extract fuzzy rules. In the next section, we present our experimental results to show the effectiveness of our rule-extracting neuro-fuzzy model for data classification.

4 Experimental results

To demonstrate our system and its applicability, in this section we present the experimental results for three classical problems. First, we consider the reconstruction of a known rule base from data to verify the ground truth. The purpose is to identify the partition in the data from the rule base. Second, we describe an application in pattern recognition using a well-known benchmark data, namely, the Iris data set. Finally, we consider a chaotic time series given by the Mackey–Glass differential equation and use SSNFS to approximate this function.

4.1 Modeling a known rule-base

We consider an existing fuzzy system which partitions a two-dimensional data into three classes: A, B, and C. Let Tr(x, a, b, c) denote the triangular MF specified by three parameters (a, b, c),

$$\mathrm{Tr}(x, a, b, c) = \begin{cases} \dfrac{x-a}{b-a}, & \text{if } a \le x \le b \\ \dfrac{c-x}{c-b}, & \text{if } b < x \le c \\ 0, & \text{if } x < a \text{ or } x > c. \end{cases} \tag{19}$$

The rule base consists of seven fuzzy rules as shown in Table 1. The input variables take values in the domain [0, 10]. The output vector y indicates the class an input pattern belongs to. The three classes are labelled as A = [100], B = [010], and C = [001].

Figure 5a shows the training data set obtained by the data-generating rule base. It contains 200 data points partitioned into three classes. We know that the data-generating system consists of seven rules, but it is not straightforward to tell how many clusters are actually present in the input space (Setnes 2000). Given this data set, we apply the SSNFS model to extract the fuzzy rules with two-dimensional Gaussian MFs. Our objective is to verify, from the given data set, whether the SSNFS is able to extract a set of rules that are similar to those in the original, known rule base. It is a network of two input neurons, one neuron for each hidden layer and three output neurons. The constant parameter ε in the spawning validity measure was initialized to 2% of the domain scale. Initially, as expected, the only single prototype extracted one fuzzy rule by detecting one rule patch. During the self-spawning learning process, the spawning action occurred seven times. Consequently the rule base expanded to eight rules finally. In Fig. 5b the learning trajectories and spawning actions were drawn in detail in the input space to

Table 1 Rule base consists of seven fuzzy rules for generating the data set

R_1	If x_1 is Tr(x_1, 0.0, 2.7, 5.2) and x_2 is Tr(x_2, 2.5, 4.2, 5.4) then y is [1, 0, 0]
R_2	If x_1 is Tr(x_1, 0.0, 2.5, 5.0) and x_2 is Tr(x_2, 4.8, 5.6, 7.5) then y is [0, 1, 0]
R_3	If x_1 is Tr(x_1, 2.5, 5.2, 7.3) and x_2 is Tr(x_2, 0.0, 1.5, 3.4) then y is [0, 0, 1]
R_4	If x_1 is Tr(x_1, 2.5, 5.2, 7.3) and x_2 is Tr(x_2, 2.5, 4.2, 5.4) then y is [0, 1, 0]
R_5	If x_1 is Tr(x_1, 5.8, 7.5, 9.4) and x_2 is Tr(x_2, 2.5, 4.2, 5.4) then y is [0, 1, 0]
R_6	If x_1 is Tr(x_1, 5.8, 7.5, 9.4) and x_2 is Tr(x_2, 5.6, 7.5, 9.2) then y is [1, 0, 0]
R_7	If x_1 is Tr(x_1, 2.5, 5.2, 7.3) and x_2 is Tr(x_2, 5.6, 7.5, 9.2) then y is [0, 0, 1]

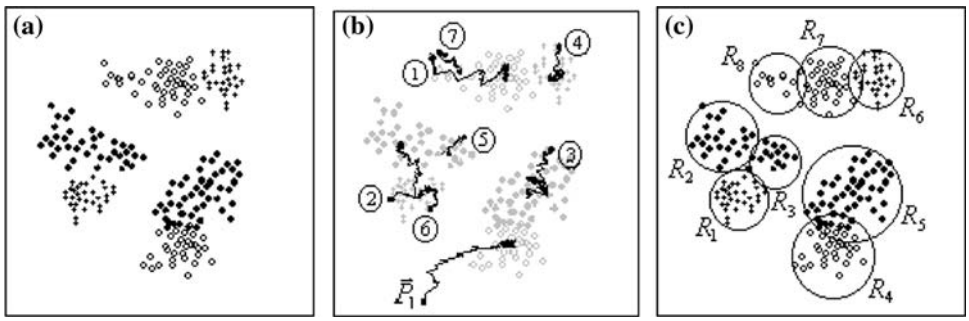

Fig. 5 **a** The data set contains 200 data points obtained by the data-generating rule base; they were partitioned into 3 classes: A (*plus signs*), B (*filled circles*), and C (*hollow circles*). **b** The OPTOC learning trajectories and spawning actions of SSNFS; the spawning occurred seven times and finally eight fuzzy rules were extracted. **c** The classification result obtained by our neural-fuzzy system

Table 2 The rule parameters extracted by SSNFS based on the data set in Fig. 5

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
$\mathbf{P}_i$	(2.71, 3.75)	(2.08, 5.83)	(3.85, 5.00)	(5.83, 2.08)	(6.46, 3.96)	(7.08, 7.71)	(5.52, 7.50)	(2.81, 7.60)
$\mathbf{R}_i$	(3.54, 4.17)	(1.46, 6.88)	(3.02, 4.79)	(4.38, 1.67)	(7.50, 5.00)	(7.19, 6.67)	(5.52, 6.46)	(3.23, 8.23)
$\mathbf{T}_i$	(0.99, 0.01, 0)	(0, 1, 0)	(0.01, 0.99, 0)	(0, 0.03, 0.97)	(0, 0.97, 0.03)	(1, 0, 0)	(0, 0, 1)	(0, 0, 1)

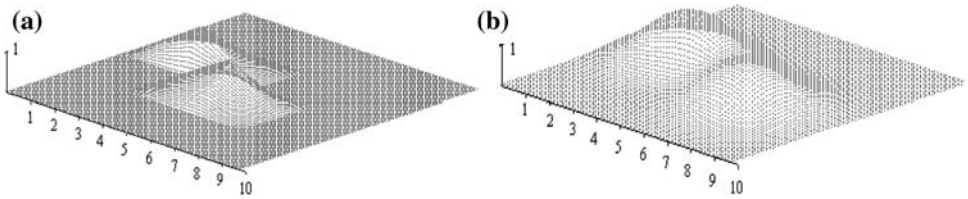

Fig. 6 **a** The class B output surface of the data generating rule base. **b** The corresponding output surface of the simulated fuzzy rules based on a limited data set

show the effectiveness of the SSNFS. The number in each circle indicates the spawning order and the initial position of the newly spawned prototype; the thin curves are the learning trajectories of prototype vectors. Figure 5c shows the partition of the data after the rules have been extracted by SSNFS. Table 2 lists the extracted rules with their parameters using SSNFS, where $\mathbf{P}_i$, $\mathbf{R}_i$, and $\mathbf{T}_i$ are defined in Definition 3.

Figure 6a shows the classification map with respect to class B data (filled circles) obtained by the original, known rule base. Figure 6b shows the corresponding classification map obtained by using the rules extracted by the SSNFS from the input data set (the 200 data points). Since the training data set does not reflect the global distribution of the data-generating rules, the extracted rules mostly perform well on these training data points and thereabouts. One can easily extract the global optimal fuzzy rules by uniformly sampling the input space with small step. Thus this is not a problem of technique but of computation complexity.

From the known rule set in Table 1, the extract rules in Table 2, and classification maps in Fig. 6, we can see that the SSNFS is able to reconstruct the existing rule base very accurately.

4.2 Rule extraction from Iris data

The *Iris* data is one of the best known databases found in pattern recognition.[1] The data set contains three classes, where each class refers to a type of iris plant and contains 50 data points. The three types are, Iris Setosa, Iris Versicolour, and Iris Virginica. For each data point, there are four real valued numerical attributes in turn: (1) sepal length in cm; (2) sepal width in cm; (3) petal length in cm; (4) petal width in cm. The objectives in this experiment are: (1) extract a set of fuzzy rules; (2) classify the data set using the extracted rules; (3) compare the class centroids with the actual centroids.

We applied the SSNFS to the Iris data set to extract a set of fuzzy rules. The SSNFS network consists of four input neurons, one hidden neuron for each hidden layer initially, and three output nodes. Throughout the test, we set $\varepsilon = 0.02$. During the learning process, the spawning action occurred four times, therefore five rules were extracted by SSNFS. Table 3 lists the final rules with their parameters.

[1] The Iris data was obtained from http://www.ics.uci.edu/mlearn/MLRepository.html via a free, public ftp site.

Table 3 The rule parameters extracted by SSNFS on the Iris data

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
$\mathbf{P}_i$	(5.19, 3.54, 1.42, 0.24)	(6.60, 2.64, 3.49, 1.23)	(6.60, 2.64, 4.72, 1.23)	(7.64, 2.64, 5.75, 2.08)	(7.64, 2.64, 6.60, 2.08)
$\mathbf{R}_i$	(4.50, 2.30, 1.30, 0.30)	(4.90, 2.40, 3.30, 1)	(4.90, 2.50, 4.50, 1.70)	(6.30, 3.40, 5.60, 2.40)	(7.90, 3.80, 6.40, 2.00)
$\mathbf{T}_i$	(1, 0, 0)	(0, 1, 0)	(0, 0.58, 0.42)	(0, 0, 1)	(0, 0, 1)

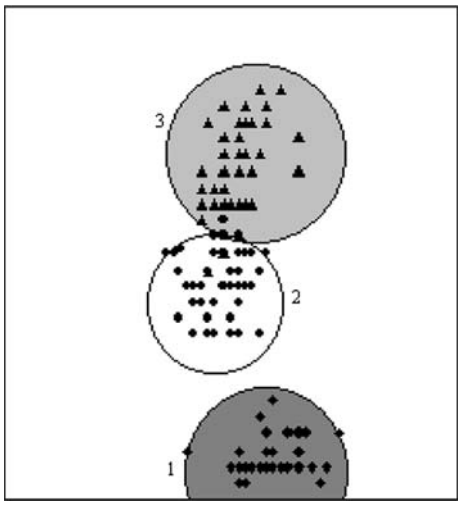

Fig. 7 An example: attribute 2 versus attribute 4 for Iris types 1, 2, and 3 classified by the SSNFS rules

To test the robustness of the proposed model, we carried out Monte Carlo tests and run the SSNFS ten times. Each time, SSNFS was able to extract a very consistent set of rules; in fact, the rules were almost identical among the tests. Figure 7 shows a classification result for attribute 2 against attribute 4 by using the extracted fuzzy rules to classify data points as being Iris Setosa, Versicolour, or Virginica.

The spawning constant ε is a constant that is used to tune the quality of rule extraction. The smaller the ε, the more rule patches in the input space, therefore the better accuracy of the rule extraction. Typically it is sufficient that $\varepsilon < 0.05$. As shown in Table 4, by setting $\varepsilon = 0.02$ or $\varepsilon = 0.05$, we can obtain the centroids of classes that are very close to the actual centroids of the three classes.

4.3 Function approximation

To show the learning capability of the SSNFS, in this section we consider a chaotic time series given by the Mackey–Glass differential equation (Mackey and Glass 1977). The prediction of future values of this time series is a benchmark problem which has been considered by a number of connectionist researchers (see Jang 1993, for more details). The equation can be described as

$$\dot{x}(t) = \frac{0.2x(t-T)}{1+x^{10}(t-T)} - 0.1x(t). \tag{20}$$

We use the values, $x(t-18)$, $x(t-12)$, $x(t-6)$, and $x(t)$, to predict $x(t+6)$. The training data were created using the fourth-order Runge–Kutta procedure with a unity step size. Assume initially $x(0) = 1.2$, $T = 17$, $x(t)$ can thus be derived for $0 \le t \le 2{,}000$. We created 1,000 values between $x = 118$ and 1,117, where the first 500 samples were used for training and the second half was used as a validation set. To give a sense of each four-dimensional data $[x(t-18), x(t-12), x(t-6), x(t)]^{\mathrm{T}}$, we displayed it as two-dimensional data points. Figure 8a shows a plot of variable $x(t-18)$ against $x(t-6)$, whereas Fig. 8b shows a plot of variable $x(t-6)$ against $x(t)$. The SSNFS has four input nodes, one output node, and one node for each hidden layer. We set ε to 0.07. During the learning process, the spawning action occurred 46 times. Therefore, 47 fuzzy rules were extracted to approximate this function. The final positions of each rule patch were shown in Fig. 8c and d, where we mapped them into the space of $x(t-18)$ against $x(t-6)$, and the space of $x(t-6)$ against $x(t)$, respectively. It must be noted that some data points in the mapped space could come from other dimensions, therefore, not all data points are covered by the rule prototypes (Cherkassky and Mulier 1998; Vapnik 1998).

Figure 9 shows the original Mackey–Glass time series (solid curve) and the simulated predictions (dashed curve) based on the first 500 samples. Since this is a local linear mapping problem, it would correspond to a first-order Sugeno fuzzy system; consequently, the SSNFS adaptively generates a large number of small systems to approximate the given data set.

Table 4 The class centroids obtained by $\varepsilon = 0.02$ and $\varepsilon = 0.05$

	Iris setosa	Iris versicolour	Iris virginica
Actual	(5.006, 3.428, 1.462, 0.246)	(5.936, 2.770, 4.260, 1.326)	(6.588, 2.974, 5.552, 2.026)
$\varepsilon = 0.05$	(5.012, 3.536, 1.460, 0.245)	(5.836, 2.731, 4.389, 1.332)	(6.848, 3.078, 5.632, 2.058)
$\varepsilon = 0.02$	(5.007, 3.425, 1.473, 0.250)	(5.882, 2.752, 4.353, 1.421)	(6.832, 3.061, 5.611, 2.052)

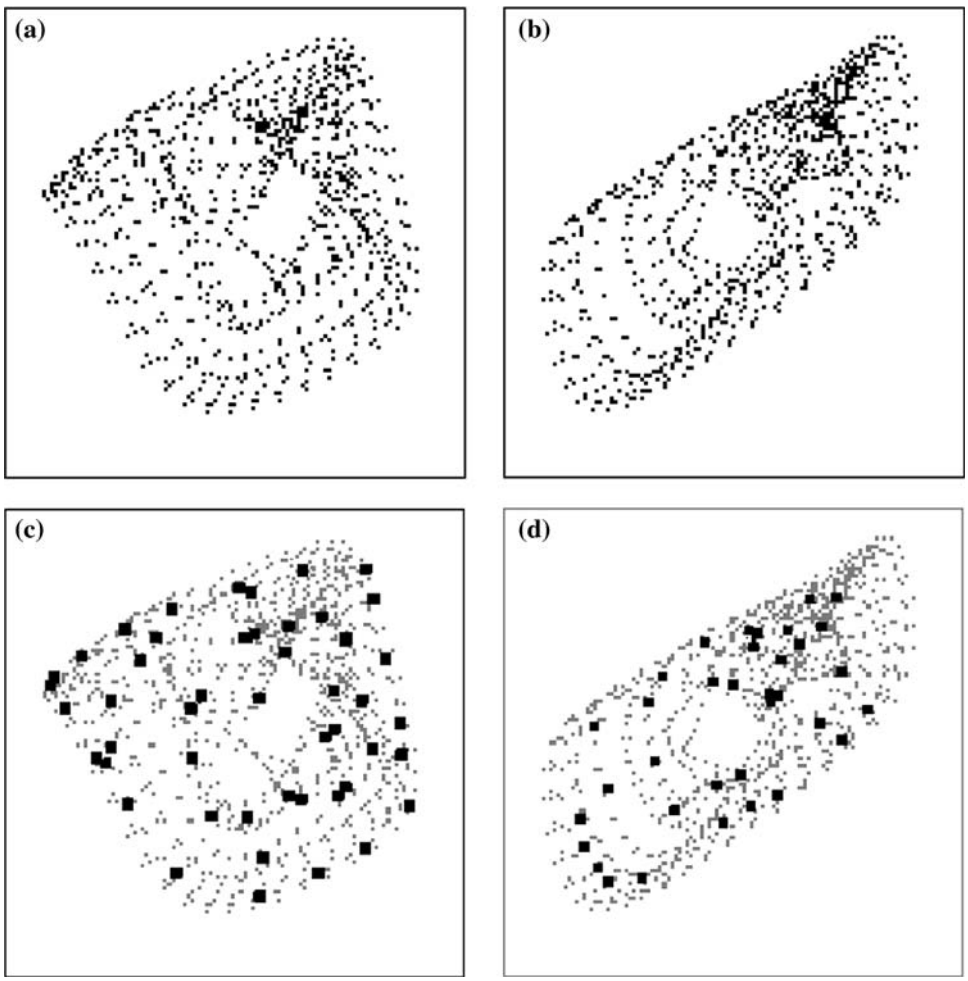

Fig. 8 **a** The view of four-dimensional data points in the two-dimensional space of $x(t-18)$ versus $x(t-6)$. **b** The view of four-dimensional data points in the two-dimensional space of $x(t-6)$ versus $x(t)$. **c** The SSNFS rule prototypes mapped in the space of $x(t-18)$ versus $x(t-6)$. **d** The SSNFS rule prototypes mapped in the space of $x(t-6)$ versus $x(t)$

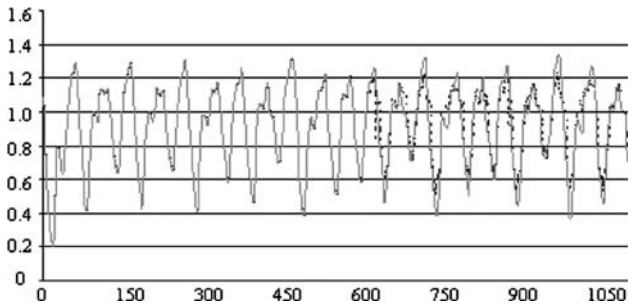

Fig. 9 Approximation of the Mackey–Glass time series by SSNFS. The *solid curve* shows the Mackey–Glass time series while the *dashed* one shows the prediction curve by SSNFS

5 Conclusions and discussions

In this paper, we first discussed the scatter-partitioning fuzzy system, based on which we then presented the SSNFS model, a neuro-fuzzy system for rule extraction. It is derived from a generic five-layer incremental perceptron model and employs the SSSCL algorithm. When initialized with a single rule prototype, the SSNFS is able to adapt its structure to reach a suitable number of rules by the self-spawning learning algorithm. The considered synthetic and real-world examples demonstrated the effectiveness and applicability of the new rule generating system.

In most real-world applications, rarely do we have an adequate prior knowledge to specify the shapes, locations, and the number of rules. Therefore, extracting rules using the SSCL algorithm is effective and robust. However, there are many challenging theoretical problems open to further research. For instance, rule generalization in SSNFS, convergence property, rule-patch validation in the context of soft computing, etc. The error based insertion strategy is a popular approach to guide the growth of a neuro-fuzzy system. This is effective in many ways, however, it has a disadvantage in that a well partitioned large cluster could still own more distortions than small cluster do. Thus, it can be always tricked to generate new, redundant prototypes to share the large cluster while ignore the small ones. To avoid this problem, in SSNFS model, we use a new spawning validity measure in which the error is part of the factors to be considered for rule growth. The rule induction and deduction strategies are always the active areas of investigation.

Acknowledgments This research has been supported by a grant from Hong Kong RGC CERG 9041020 (CityU 118205).

References

Pomares H, Rojas I, Ortega J, Gonzalez J, and Prieto A (2000) A systematic approach to a self-generating fuzzy rule-table for function approximation. IEEE Trans Syst Man Cybern Part B Cybern 30(3):431–447

Chen W, Saif M (2005) A novel fuzzy system with dynamic rule base. IEEE Trans Fuzzy Syst 13(5):569–582

Boukezzoula R, Foulloy L, Galichet S (2006) Inverse controller design for fuzzy interval systems. IEEE Trans Fuzzy Syst 14(1):569–582

Pal SK, Mitra S (1992) Multi-layer perceptron, fuzzy sets and classification. IEEE Trans Neural Netw 3(5):683–697

Kandel A (1992) Fuzzy expert systems. CRC Press, Boca Raton

Sugeno M (ed) (1985) Industrial applications of fuzzy control. Elsevier, New York

Pedrycz W (1989) Fuzzy control and fuzzy systems. Wiley, New York

Berenji HR, Khedkar P (1992) Learning and tuning fuzzy logic controllers through reinforcements. IEEE Trans Neural Netw 3(5):724–740

Lin CT (1994) Neural fuzzy control systems with structure and parameter learning. World Scientific, Singpore

Jouffe L (1998) Fuzzy inference system learning by reiforcement methods. IEEE Trans Syst Man Cybern Part C 28(3):338–355

Rojas I, Pomares H, Ortega J, Prieto A (2000) Self-organized fuzzy system generation from training examples. IEEE Trans Fuzzy Syst 8(1):23–36

Nie J, Linkens DA (1994) Fuzzy neural control: principles, algorithms, and applications. Prentice-Hall, Englewood Cliffs

Uncu ö, Türkşen İB (2007) Discrete interval type 2 fuzzy system models using uncertainty in learning parameters. IEEE Trans Fuzzy Syst 15(1):90–106

Duch W, Adamczak R, Grabczewski K (2001) A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. IEEE Trans Neural Netw 12(2):277–306

Duch W, Setiono R, Żurada JM (2004) Computational intelligence methods for rule-based data understanding. Proc IEEE 92(5):771–805

Setnes M (2000) Supervised fuzzy clustering for rule extraction. IEEE Trans Fuzzy Syst 8(4):416–424

Pal NR, Eluri VK, Mandal GK (2002) Fuzzy logic approaches to structure preserving dimensionality reduction. IEEE Trans Fuzzy Syst 10(3):277–286

Mitra P, Murthy CA, Pal SK (2002) Density-based multiscale data condensation. IEEE Trans Pattern Anal Mach Intell 24(6):734–747

Cherkassky V, Mulier P (1998) Learning from data. Wiley, New York

Lee MA, Takagi H (1993) Integrating design stages of fuzzy systems using genetic algorithms. In: Proceedings of FUZZY-IEEE/IFES'93, San Franciso, CA, pp 612–617

Ishibuchi H, Nozaki K, Yamamoto N, Tanaka H (1995) Selecting fuzzy if-then rules for classification problems using genetic algorithms. IEEE Trans Fuzzy Syst 3(2):260–270

Yager RR, Filev DP (1993) Unified structure and parameter identification of fuzzy models. IEEE Trans Syst Man Cybern 23(4):1198–1205

Wang LX, Mendel JM (1992) Fuzzy basis functions, universal approximation, and orthogonal least-squares learning. IEEE Trans Neural Netw 3(5):807–813

Yen J, Wang L (1999) Simplifying fuzzy rule-based models using orthogonal transformation methods. IEEE Trans Syst Man Cybern Part B 29:13–24

Brown M, Harris C (1994) Neurofuzzy adaptive modeling and control. Prentice-Hall, Englewood Cliffs

Gupta MM, Rao DH (1994) On the principles of fuzzy neural networks. Fuzzy Sets Syst 61(1):1–18

Jang JR (1993) ANFIS: adaptive-network-based fuzzy inference system. IEEE Trans Syst Man Cybern 23(3):665–685

Fritzke B (1997) Incremental neuro-fuzzy systems. In: Proceedings of application of soft computing, SPIE international symposium on optical science, engineering and instrumentation, San Diego

Mitra S, Hayashi Y (2000) Neuro-fuzzy rule generation: survey in soft computing framework. IEEE Trans Neural Netw 11(3):748–768

Jang J-SR, Sun C-T (1995) Neuro-fuzzy modeling and control. Proc IEEE 83(3):378–406

Kruse R, Nauck D (1995) Learning methods for fuzzy systems. In: Proceedings of the 3rd German GI-workshop "Neuro-Fuzzy-Systeme", Darmstadt, Germany, vol 3, pp 683–697

Nauck D (1997) Neuro-fuzzy systems: review and prospects. In: Procedings of the 5th European congress on intelligent techniques and soft computing (EUFIT'97), Aachen, pp 1044–1053

Keller JM, Tahani H (1992) Implementation of conjunctive and disconjunctive fuzzy logic rules with neural networks. Int J Approx Reason 6:221–240

Keller JM, Yager RR, Tahani H (1992) Neural network implementation of fuzzy logic. Fuzzy Sets Syst 45(1):1–12

Zimmermann HG, Neuneier R, Dichtl H, Siekmann S (1996) Modeling the german stock index DAX with neuo-fuzzy. In: Proceedings of fourth European congress on intelligent techniques and soft computing (EUFIT96), Aachen

Nauck D, Kruse R (1995) NEFCLASS-A neuro-fuzzy approach for the classification of data. Applied computing. In: George KM, Carrol JH, Deaton E, Oppenheim D, Hightower J (eds) Proceedings of the 1995 ACM symposium on applied computing, Nashville, February 26–28. ACM Press, New York, pp 461–465

Kruse R, Gebhardt J (1994) Foundations of fuzzy systems. Wiley, Chichester

MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of 5th Berkeley symposium on mathematical statistics and probability. University of California Press, Berkeley, pp 281–297

Mackey MC, Glass L (1977) Oscillation and chaos in physiological control systems. Science 197(4300):287–289

Vapnik VN (1998) Statistical learning theory. Wiley, New York