國 立 交 通 大 學

電機與控制工程學系

博 士 論 文

# 具有相互影響之遞迴式自我演化類神經模糊系統及其應用

# A Novel Recurrent Self-evolving Neural Fuzzy System and Its Applications

研 究 生：林洋印

指導教授：張志永　教授

中 華 民 國 一 百 零 二 年 四 月

# 具有相互影響之遞迴式自我演化類神經模糊系統及其應用

# A novel Recurrent Self-evolving Neural Fuzzy System and Its Applications

研 究 生：林洋印　　Student：Yang-Yin Lin

指導教授：張志永 博士　　Advisor：Dr. Jyh-Yeong Chang

國 立 交 通 大 學

電 機 與 控 制 工 程 學 系

博 士 論 文

A Dissertation

Submitted to Department of Electrical and Control Engineer

College of Electrical Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Electrical and Control Engineering

April 2013

Hsinchu, Taiwan, Republic of China

中華民國一百零二年四月

# 具有相互影響之遞迴式自我演化類神經模糊系統及其應用

研究生：林洋印　　指導教授：張志永 博士

國立交通大學電機與控制工程學系（研究所）博士班

## 摘　　要

本篇論文提出以相互影響地遞迴式架構為基礎之類神經模糊系統及其應用於動態系統辨識。而此論文主要分成四大部份，第二部份詳細介紹相互影響地遞迴式架構與類神經模糊系統作結合，並且在後鑑部份使用輸入變數的非線性組合，不同於傳統Takagi-Sugeno-Kang （TSK）架構，它是利用函數展開的方式，能在高維度的輸入空間中提供良好的非線性決策能力，因此，可使網路輸出更具體且更逼近目標輸出。在架構學習上，使用線上學習模糊分群法，而此演算法能有效地處理時變特徵問題。在參數學習上，後鑑部參數的更新是由可變動維度之卡爾曼濾波器演算法調整,可具有高精密學習的性能。前鑑部及遞迴式參數則利用梯度下降法去做參數更新的動作。在第三部份中，我們提出區間第二類型模糊邏輯系統結合發展出的遞迴式網路，即相互影響地遞迴式區間第二類型類神經模糊系統。區間第二類型模糊邏輯系統具有良好的雜訊容忍度，能直接處理規則的不確定性，這是第一類型模糊邏輯系統所不能達到的。在架構學習上，相互影響之遞迴式區間第二類型類神經模糊系統最初不包含任何規則，所有規則的產生是由線上第二類型模糊分群所取得。在參數學習上，後鑑部參數的更新是由排序後規則之卡爾曼濾波器演算法調整以改善系統的性能。前鑑部及遞迴參數的更新由梯度下降法做調整。在此，我們提出參數消除的方法針對無效的遞迴參數，因規則數太大，會產生許多遞迴參數，我們將冗餘的遞迴參數刪除，來減少網路的計算量。模擬結果顯示出針對

動態系統在無喧雜的狀況下，所提出的相互影響之遞迴式模糊類神經網路具有優越的性能。最後，我們將與其他方法做比較，證實所提出的架構是卓越的。

# A Novel Recurrent Self-evolving Neural Fuzzy System and Its Applications

Student：Yang-Yin Lin　　　　　Advisor：Dr. Jyh-Yeong Chang

Department of Electrical and Control Engineering

National Chiao-Tung University

## ABSTRACT

This dissertation mainly describes two different kinds of recurrent neural fuzzy systems, involving a novel recurrent self-evolving fuzzy neural network for identification and prediction of time-varying plants and a novel recurrent interval type-2 neural fuzzy system with self-evolving structure and parameter for dynamic system processing under noise-free and noise environment. For the first kind, we describe a novel recurrent self-evolving neural fuzzy system, namely an interactively recurrent self-evolving fuzzy neural network (IRSFNN). The recurrent structure in an IRSFNN is formed as an external loops and internal feedback by feeding the rule firing strength of each rule to others rules and itself. The consequent part in the IRSFNN can be chosen by a Takagi-Sugeno-Kang (TSK) or functional-link-based type. The proposed IRSFNN employs a functional link neural network (FLNN) to the consequent part of fuzzy rules for promoting the mapping ability. Unlike a TSK-type fuzzy neural network, the FLNN in the consequent part is a nonlinear function of input variables. An IRSFNN's learning starts with an empty rule base and all of rules are generated and learned online through a simultaneous structure and parameter learning. The consequent update parameters are derived by a variable-dimensional Kalman filter algorithm. The premise and recurrent parameters are learned through a gradient descent algorithm. We test the IRSFNN for the prediction and identification of dynamic plants and compare it to other well-known recurrent FNNs. The proposed model obtains enhanced performance results. For the second kind, we introduce a mutually recurrent interval type-2 neural fuzzy system (MRIT2NFS) with self-evolving structure and parameters for system identification under both noise-free and noisy environments. The MRIT2NFS employs interval type-2 set in the premise clause in order to enhance noise tolerance of system. The consequent part of each recurrent fuzzy rule is defined using the Takagi-Sugeno-Kang (TSK) type with interval weights. The structure learning of MRIT2NFS uses on-line type-2 fuzzy clustering to determine number of fuzzy rules. For parameter learning, the consequent part parameters are tuned by rule-ordered Kalman filter algorithm to reinforce parameter learning ability. The type-2 fuzzy sets in the antecedent and weights representing the mutual feedback are learned by gradient descent algorithm. After the training, a weight-elimination scheme eliminates feedback connections that do not have much effect on the network behavior. This method can efficiently remove redundant recurrence weights. Simulation results show that the MRIT2NFS produces smaller root mean squared errors using the same number of iterations.

# Acknowledgment

在這研究期間裡，首先要感謝我的博士班指導教授 張志永博士和林進燈博士，在二位教授豐富的學識、殷勤的教導及嚴謹的督促下，使我學習到許多的寶貴知識及在面對事情中應有的處理態度、方法，並且在研究與投稿論文的過程中，二位教授有許多深入的見解及看法且對於斟酌字句、思慮周延，更是我該學習的目標。師恩浩蕩，指導提攜，銘感於心。

由衷感謝口試委員李祖聖教授、蘇順豐教授、洪宗貝教授、楊谷洋教授、及林正堅教授給予許多寶貴的建議與指正，使得這篇論文更加完整。同時要感謝交大腦科學中心全體同學，在研究的過程中不斷的互相砥礪及討論。感謝過去 97 級中興電機所的同學，在這過程中，一起分享不同的經驗。感謝一路走來陪伴身邊的所有朋友，使得我的研究生涯變得多采多姿。

特別要感謝我的父親、母親、姐姐，在這段日子中不斷的給予支持及鼓勵，讓我能夠專心於研究的工作並完成博士學位。最後誠摯地以本論文研究成果獻給我的師長、父母、家人及所有的朋友們。

林洋印

一百零二年四月二十三日

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Dynamic systems depend on past inputs, past outputs, or both, and identification and modeling of such systems is not as straightforward as that for static / algebraic systems. For dynamic system processing, practical problems are encountered in a variety of areas, such as control, pattern recognition, time series prediction, and signal processing. Recently, the combination of recurrent structures and fuzzy neural networks has become popular to identify and recognize temporal behaviors [1–16, 43–44]. Therefore, recurrent structures enable effectively address temporal sequences responding to memory information from prior system states.

In contrast with pure feed-forward FNN, we have to know the number of lagged inputs and outputs in advance, and feed these lagged values as feed-forward FNN input. The exact order of dynamic system is usually unknown, and thus, we do not know the number of lagged values to provide. Moreover, the lagged values increase input dimensions and result in a larger network size. Apparently, the use of a feed-forward FNN is unsuitable for constructing dynamic system. Therefore, some recurrent fuzzy neural networks (RFNNs) have already been proposed [1–16, 43–44] for solving the temporal characteristics of dynamic systems, and have been shown to outperform feed-forward FNNs and recurrent neural networks. Recently, a considerable research effort has being devoted toward developing recurrent neural-fuzzy models that are separated into two major categories. One category of recurrent FNNs uses feedback from the network output as the recurrence structure [2–4], [9]. In [2], recurrent self-organizing neural fuzzy inference network (RSONFIN) uses a global feedback structure, where the firing strengths of all rules are summed and fed back as internal network inputs. The other approach of recurrent FNNs uses feedback from internal state variables as its recurrence [10, 12, 14, 44]. In [14], the authors presented a recurrent self-evolving fuzzy neural network with local feedbacks for dynamic system identification, where the recurrent firing values are influenced by both prior and current values.

One import purpose is to design consequent part of FNN which is able to impact the performance on using different types. Researchers usually use two types of fuzzy if-then rules and fuzzy reasoning employed, i.e., Mamdani-type and TSK-type. For Mamdani-type fuzzy neural networks [2, 6, 17–19], the minimum fuzzy implication is adopted in fuzzy reasoning. For TSK-type fuzzy neural networks [5, 9, 14, 20, 21], the consequent part of each rule is a linear function of input variables. Several studies [14, 20, 21] indicate that the performance of a feedforward TSK-type fuzzy network in network size and learning accuracy is superior to those of Mamdani-type fuzzy networks. A feedforward TSK-type fuzzy network appears to have more free parameters to adjust input space mapping. However, each consequent part of each fuzzy rule in a standard TSK-type fuzzy neural network does not take full advantage of the mapping capabilities of local approximation by rule hyper-planes. Therefore, several studies [22–28] consider trigonometric functions to replace the traditional TSK-type fuzzy reasoning and also obtain the better performance.

In this view, the functional-link neural networks (FLANN) [22, 23] have been proposed using trigonometric functions to construct consequent part. The functional expansion increases the dimensionality of the input vector and thus, creation of nonlinear decision boundaries in the multidimensional space and identification of complex nonlinear functions become simple with this network.  It seems to be more efficient, based on these results, to include the functional-link fuzzy rules into the design of recurrent fuzzy network.

With above mentioned motivations, this study presents the combination of a novel recurrent structure and a FLANN to construct the consequent part, called an interactively recurrent self-evolving fuzzy neural network (IRSFNN), for dynamic system identification and prediction. The proposed IRSFNN contains four major contributions as follows:

(1) A recurrent structure with interaction feedback incorporates the advantages of local

feedback and global feedback. The global feedback in the proposed network means that the necessary information is obtained from the other fuzzy rules. Local source (a rule gets feedback from itself only) is not sufficient to represent the necessary information. Therefore, external (global) backward connection is aim to reimburse the shortcoming of local information and then achieve comprehensive information requirement.

(2) Many studies [1–13] have only considered the past states in recurrent structure, which is insufficient without referring to current states. Previous studies [14, 29] were provided the strong evidences that compatibly use past and current states to be more desirable. Therefore, the proposed recurrent structure depends on current states along with previous states in order to obtain excellent compromise with temporal.

(3) We use the FLNN to replace the traditional TSK-type fuzzy reasoning, and compare their performance. As has explained before, the functional expansion increases the dimensionality of the input pattern and thus, creation of nonlinear decision boundaries in the multidimensional space and identification of complex nonlinear functions become simple with this network.

(4) We use hybrid learning algorithms for parameter learning to reinforce the network performance.

For structure learning, all of the rules and fuzzy sets are generated on-line in an IRSFNN, which helps automate rule generation. We do not need to set any initial IRSFNN structure in advance. The antecedent part and recurrent parameters are learned by gradient descent algorithm. The consequent parameters in an IRSFNN are tuned using a variable-dimensional Kalman filter algorithm. This algorithm handles inputs with variable dimensions, a phenomenon caused by incremental rules during the structure learning process.

All of the recurrent FNNs that we have discussed so far use type-1 fuzzy sets. In recent years, studies on type-2 fuzzy logic systems (FLSs) have drawn much attention [45–49, 78–87]. Type-2 FLSs are extensions of type-1 FLSs, where the membership functions involved in the fuzzy rules are type-2 fuzzy sets. We shall refer to such rules as trype-2 fuzzy rules or type-2 rules. The membership values of a type-2 fuzzy set are type-1 fuzzy sets. Type-2 FLSs appear to be more promising than their type-1 counterparts in handling uncertainties, that allow researchers to model and minimize the effect of uncertainties associated with rule-base system, and have already been successfully applied in several areas [49–53, 75–77, 88–91]. The uncertainties in type 2 fuzzy sets can arise from different sources. Four types of uncertainties are described in [46] and [47]. One of them is related to the answers of experts to the same question in different manners. The second type of uncertainty is related to the estimation of the membership function of the same linguistic value by different experts, the third is connected with the noise of measurements that activate type-1 FLS, and the last one is related to the noisy data that are used to tune the parameters of type-1 FLSs. Type-1 fuzzy systems cannot directly model these types of uncertainties. Because the membership functions of type 2 fuzzy systems are themselves fuzzy, they provide a powerful framework to represent and handle such types of uncertainties.

Usually, the T2FNN is computationally more expensive than that of its Type-1 counterpart primarily due to the complexity of type reduction from Type-2 to Type-1. An Interval Type-2 Fuzzy set (IT2FS) is a special case of a general type-2 fuzzy set, which reduces the computational overhead of a general type-2 fuzzy system significantly. For an IT2FS, the membership associated with an element is a sub-interval of [0, 1]. In this dissertation we use the interval type-2 fuzzy modeling to simplify the computational efforts to some extent. In [54]–[59], some interval type-2 FNNs are proposed for designing of interval type-2 FLS. In [29, 60–69, 92], the authors have proposed automatic design of fuzzy rules, which are used in a variety of applications. A self-evolving interval type-2 fuzzy neural network (SEIT2FNN) is proposed in [62], which learns the structure and parameters in an online manner. The premise and consequent parameters in an SEIT2FNN are tuned by gradient descent and rule ordered Kalman filter algorithm, respectively. The performances of

SEIT2FNN are especially good for time varying systems. Several Interval type-2 FNNs [29, 67–69], which use feedback/recurrent structure are proposed for modeling of dynamic systems. In [67], a recurrent interval type-2 fuzzy neural network (RIT2FNN-A) that uses interval asymmetric type-2 fuzzy sets is proposed. This five-layer FNN uses a four-layer forward network and a feedback layer. In [68], the authors propose an internal/interconnection recurrent type-2 fuzzy neural network (IRT2FNN) structure that is suitable for dealing with time-varying systems. All free parameters of the IRT2FNN are updated via gradient descent algorithm. Moreover, recurrent interval type-2 FNNs with local feedbacks are proposed in [29, 69], where the recurrent property is achieved by locally feeding the firing strength of each rule back to itself. In [29], the consequent part of the recurrent self-evolving interval type-2 fuzzy neural network (RSEIT2FNN) is a linear function of current and past outputs and inputs. On the other hand, the consequent part in the RIFNN [69] is of Mamdani type, which is formulated as an interval-valued fuzzy set. In many papers, it has been seen that the Takagi-Sugeno-Kang (TSK) type modeling can do an excellent job of modeling dynamic systems [7, 14, 21, 62, 66, 68].

Here, we propose a Mutually Recurrent Interval Type-2 Neural Fuzzy System (MRIT2NFS) for dynamic system identification. The MRIT2NFS has a self-evolving ability such that it can automatically evolve to acquire the required network structure as well as its parameters based on the training data. Therefore, to start the learning process no pre-assigned network structure is necessary. In this proposed MRIT2NFS, we have three major contributions as follows:

(1) We propose a novel recurrent NFS structure utilizing interval type-2 fuzzy sets. Our network incorporates the advantage of local feedback and effective delivery of information through mutual feedbacks in order to achieve information completeness. In our network, the internal feedback and interaction loops in the antecedent part are formed by feeding the firing strength of each rule back to itself and to other rules.

Based on the view of networks, many studies employ external registers to memorize prior states that could cause network's complexity when number of rules is bigger. Therefore, a internal register is used for reducing network's complexity.

(2) An innovative learning algorithm for the structure and parameters of the system is suitable for handling time-varying systems, i.e., self-evolving structure and parameter mechanism.

(3) We also propose an interesting scheme to eliminate the less-useful recurrent weights. During the learning process, the MRIT2NFS may generate many recurrent weights when the rule base is bigger. As a result of elimination of the less-useful recurrent weights, our system achieves a significant reduction in both complexity and computational requirements.

The consequent parameters in the MRIT2NFS are tuned by a rule-ordered Kalman filter algorithm. The antecedent parameters and all of the rule recurrent weights are learned by a gradient descent learning algorithm. To demonstrate the performance of MRIT2NFS, several simulations have been conducted. The performance of MRIT2NFS is also compared with that of recurrent type-1 FNNs, feed-forward type-1 FNNs, and other type-2 FNNs.

# Chapter 2

# An Interactively Recurrent Self-evolving Fuzzy Neural Network (IRSFNN)

## 2.1 Brief Survey of Existing Methods

Recently, considerable research has been devoted toward these developing recurrent fuzzy neural networks, and these networks can be separated into two major categories. One category of recurrent FNNs in studies [1–9, 16], the recurrent structure uses global feedbacks. In [2], a recurrent self-organizing neural fuzzy inference network (RSONFIN) computes the values of the internal feedback variables using all rule firing strengths and the consequent parts are fuzzy sets. The recurrent structure in the RSONFIN just considers past state. For parameter learning, the RSONFIN uses gradient descent algorithm to tune free parameters. The authors in [3] and [4] proposed an output-recurrent fuzzy neural network where the output values are fed back as input values. In [7], the TSK-type recurrent fuzzy network's structure is similar to an RSONFIN. The recurrent neuron-fuzzy network in [9] feeds back the network output values not only globally to all the rule inputs, but also locally to the consequent part of each rule, in the form of the autoregressive moving average with exogenous inputs model. The recurrent high-order neural network (RHONN) [16] trained with an extended Kalman filter algorithm was proposed for optimal control of nonlinear systems.

The other approach of recurrent FNNs [10–14] uses feedback loops from internal state variables as its recurrence structure. The design of local recurrent structures seems to be simpler than that of global recurrent structures, and also obtains superior performance. In [10] and [11], the recurrent property is achieved by feeding the output of each membership function back to itself; thus each membership value is only influenced by its previous value. The recurrent property in study [14], a recurrent self-evolving fuzzy neural network with local feedback (RSEFNN-LF), is achieved by locally feeding the output of temporal firing strength back to itself; thus, temporal firing strength is influenced by current and past states.

As mentioned earlier, many researchers frequently use Mandani-type or TSK-type to construct consequent part of fuzzy rules. Many studies indicate that TSK-type fuzzy systems significantly outperform Mandani-type fuzzy systems. However, TSK-type fuzzy neural network does not take full advantage of the mapping capabilities of local approximation by rule hyper-planes. In order to overcome this problem, our proposed model employs the FLANN [22], [23] to strength the mapping ability of input space. Therefore, nonlinear function (trigonometric function) to the consequent part shall be able to effectively discriminate in mapping input space. Previous studies [22–28] indicated that the use of trigonometric function obtains better performances than the use of TSK-type. As a result, in this dissertation the marriage of a novel recurrent structure and functional-link-based NN is a significant research for addressing the temporal problems as demonstrated by every example.

## 2.2 IRSFNN Structure

This section introduces the structure of the functional link neural network and multiple-input-single-output IRSFNN. The recurrent structure in the IRSFNN uses interaction feedback that has the ability to capture critical information from other rules. The consequent part of each recurrent fuzzy rule is functional link and executes a nonlinear model. Next, we have described the structure of functional link neural network.

### *2.2.1 Structure of a Functional-link Artificial Neural Network*

The functional link artificial neural network (FLANN) is basically a single layer structure in which nonlinearity is introduced by enhancing the input pattern with nonlinear functional expansion. Therefore, the FLANN structure considers trigonometric functions. Fig 2.1 shows the structure of FLANN, where each of the input patterns is passed through a functional expansion block yielding a corresponding $N$-dimensional expanded vector. Suppose that for the input pattern $X$ is of a two-dimensional input $(x_1, x_2)$ the expanded inputs are using trigonometric functions to be taken. The expanded input variables can be denoted as $\vec{\varphi} = (1, x_1, \sin(\pi x_1), \cos(\pi x_1), x_2, \sin(\pi x_2), \cos(\pi x_2))$.

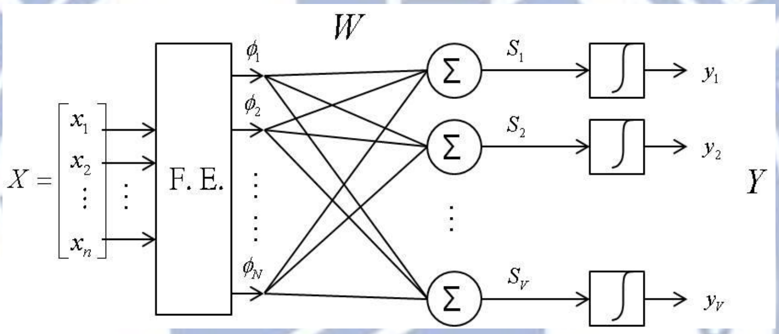

Fig. 2.1. Structure of FLANN.

The theory of the FLANN for multidimensional function approximation has been discussed and analyzed below [22, 23]. Let us consider a set of basis functions $B = \{\varphi_k \in \mathbf{\Phi}(A)\}_{k \in \mathbf{K}}$, with the following properties; (1) $\varphi_1 = 1$; (2) the subset $B_j = \{\varphi_k \in B\}_{k=1}^{N}$ is a linearly independent set, that is, if $\sum_{k=1}^{N} \varphi_k w_k = 0$, then $w_k = 0$ for $k = 1, \cdots, j$; and (3) $\sup_j \left[ \sum_{k=1}^{j} \|\varphi_k\|_A^2 \right]^{1/2} < \infty$. Next, $B_N = \{\varphi_k\}_{k=1}^{N}$ is a set of a set of basis functions to be considered, as shown in Fig. 2.1. Hence, output of functional expansion block is composed by $(\varphi_1, \varphi_2, \ldots, \varphi_N) \in B_N$ with the following input–output relationship for the $j$ th output.

$$\hat{y}_j = \rho(S_j); \quad S_j = \sum_{k=1}^{N} \varphi_{kj} w_k(X) \tag{2.1}$$

where $X \in A \subset \Re^n$, that is, $X = (x_1, x_2, \cdots, x_n)^T$ is the input dimension and $W = (w_{j1}, w_{j2}, ..., w_{jN})^T$ is the weight vector associated with the jth output of the FLANN. The vector $S$ is a matrix of linear outputs of the FLANN, and the output vector is $\hat{y} \in \Re^V$, that is, $\hat{y} = (\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_V)^T$. The nonlinear function can be denoted as

$$\rho(\cdot) = \tanh(\cdot) \tag{2.2}$$

In the IRSFNN model, the corresponding weights of functional link bases do not exist in the initial state, and the amount of the corresponding weights of functional link bases generated by the online learning algorithm is consistent with the number of fuzzy rules. Section 2.3 describes the self-evolving technology.

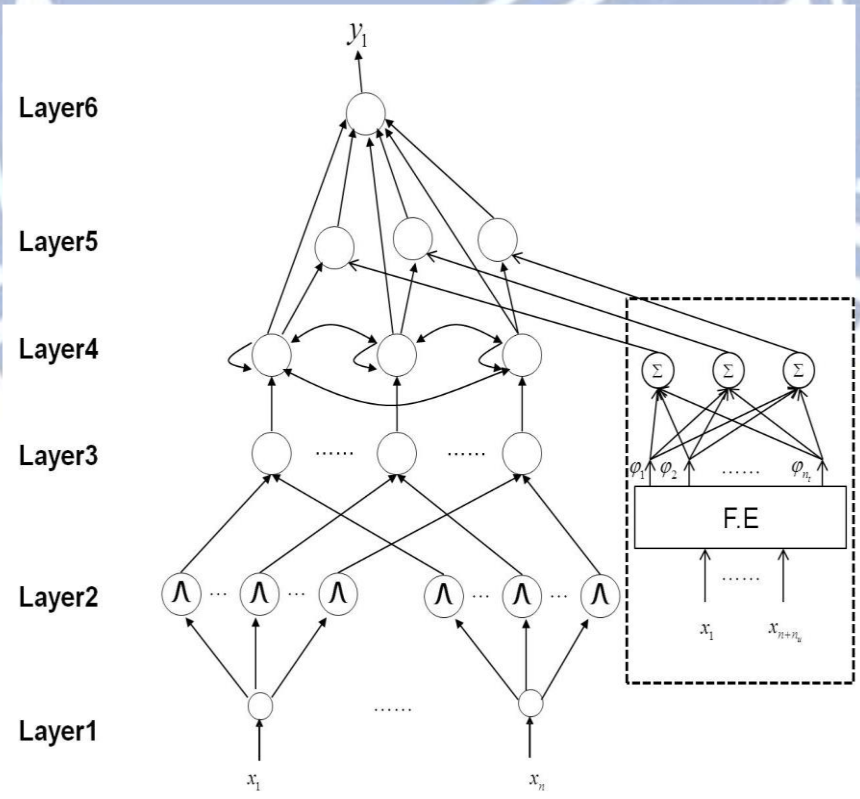

Fig. 2.2. Structure of the proposed IRSFNN model, where each recurrent fuzzy rule in layer 4 forms a locally and globally recurrent structure and each node in layer 5 combines functional-link-based.

## 2.2.2 Structure of IRSFNN

This sub-section describes the IRSFNN model that employs FLANN to the consequent part of the IRSFNN for enhancing network's performance. Fig 2.2 shows the proposed six-layered IRSFNN structure. The detailed function of each layer is discussed next.

For a clear understanding of the mathematical function of each node, we will describe function relationship between each layer. The net input to the *i*th node in layer *l* is represented as $u_i^{(l)}$ and the output value is represents as $O_i^{(l)}$

*Layer 1 (Input layer)*: The inputs are crisp values and $\vec{x} = (x_1, \cdots, x_n)$ are fed as inputs to this layer. This is in contrast to feed-forward FNNs where both current and past states are fed as inputs to input layer when such networks are used to model time-varying systems. Weight requiring adjustment in this layer is absent.

*Layer 2 (Fuzzification layer)*: Each node in this layer defines a Gaussian membership function (MF) and performs a fuzzification operation. For the $i$ th fuzzy set $A_j^i$ on the input variable $x_j$ , $j = 1, \ldots, n$, a Gaussian MF is computed by Eq. (1.3)

$$\mu_j^i(x_j) = O_i^{(2)} = \exp\left\{-\frac{1}{2}\left(\frac{u_j^{(2)} - m_j^i}{\sigma_j^i}\right)^2\right\}, \text{ and } u_j^{(2)} = O_j^{(1)} \tag{2.3}$$

*Layer 3 (Spatial firing layer)*: Each node in this layer represents one fuzzy rule that computes the firing strength. Because this layer does not depend on any temporal input, we call this layer "spatial" to distinguish it from the "temporal" firing strength computed in the next layer. For the obtained spatial firing strength $\phi^i$ , each node performs a fuzzy meet operation on inputs it receives from layer 2 using an algebraic product operation.
There are M nodes in this layer, and the spatial firing strength is computed as

$$\phi^i = O_i^{(3)} = \prod_{j=1}^{n} u_j^{(3)}, \text{ and } u_j^{(3)} = O_j^{(2)} \tag{2.4}$$

where $M$ is the total number of rules.

*Layer 4 (Temporal firing layer)*: Each node in this layer is a recurrent rule node, which formulates an internal feedback (self-loop) and external interaction feedback loop. The ideal of temporal firing strength is extended from the concept of Infinite Impulse Response (IIR) filter that formulates recursive function of prior states and current observation. The output of a recurrent rule node is a temporal firing strength that depends not only on current spatial firing strength but also on the previous temporal firing strength. The temporal firing strength is a linear combination function expressed as

$$O_i^{(4)} = \sum_{k=1} (\lambda_{ik}^q \cdot O_k^{(4)}(t-1)) + (1-\gamma_i^q) \cdot u_i^{(4)}, \text{ and } u_i^{(4)} = O_i^{(3)} \tag{2.5}$$

that is,

$$\psi_i^q(t) = \sum_{k=1} (\lambda_{ik}^q \cdot \psi_k^q(t-1)) + (1-\gamma_i^q) \cdot \phi^i(t), \qquad i = 1, \cdots, M \text{ and } q = 1, \cdots, n_O \tag{2.6}$$

where $\gamma_i^q = \sum_{k=1}^{M} \lambda_{ik}^q$ and $\lambda_{ik}^q = \dfrac{C_{ik}^q}{M}$ $(0 \le C_{ik}^q \le 1)$ is the rule interaction weight between itself and other rules. For the updated recurrent weights, the proposed approach uses a gradient descent algorithm to derive the optimal values. The recurrent weights $\lambda_{ik}^q$ determine the compromised ratio between the current and previous inputs to the network outputs.

*Layer 5 (Consequent layer)*: Each node in this layer is an optional node, called a consequent layer, and can be TSK-type or functional-link-based fuzzy rules. The weight of the link from a node in layer 4 to one in layer 5 is $a_{i0}^q$, for $q = 1, \cdots, n_o$ and $i = 1, \cdots, M$. For the TSK-type IRSFNN, the node output is a linear combination of current input states $x_1, \cdots, x_n$. The output of TSK-type $\overset{-q}{v_i}$ of the $i$ th rule node connecting to the $q$ th output variable is computed as follows:

$$\bar{v}_i^q = \bar{O}_i^{(5)} = \sum_{j=0}^{n} a_{ij}^q \cdot u_j^{(4)}, \text{ and } u_j^{(4)} = O_j^{(1)} \tag{2.7}$$

where $x_0 \equiv 1$.

For the functional-link-based IRSFNN, the output uses a functional expansion as given by the trigonometric polynomial basis function $[x_1 \ \sin(\pi x_1) \ \cos(\pi x_1) \ x_2 \ \sin(\pi x_2) \ \cos(\pi x_2)]$ for two-dimensional input variables. The output of functional-link-based $\tilde{v}_i^q(t)$ is expressed by

$$\tilde{v}_i^q = \tilde{O}_i^{(5)} = \sum_{k=0}^{n_t} a_{ik}^q \cdot \varphi_k, \quad n_t = 3\times(n+n_u) \tag{2.8}$$

where $\varphi_0 \equiv 1$.

The coefficient $n_u$ denotes lag numbers of system output or control input. If we do not use extra lagged values ( $n_u$ =0), $\varphi_k = (x_1, \sin(\pi x_1), \cos(\pi x_1), \cdots, \text{x}_n, \sin(\pi x_n), \cos(\pi x_n))$ and $k = 1, \cdots, n_t$. The constant $n_t$ is an amount of basis expansion according to input variables.

*Layer 6 (Output layer)*: Each node in this layer corresponds to one output variable. For defuzzification operations, the $q$th output layer node computes the network output variable $y_q$ by using the weighted average method.

For the TSK-type IRSFNN, the output can be expressed as

$$y_q = O^{(6)} = \frac{\sum_{i=1}^{M} O_i^{(4)} \bar{O}_i^{(5)}}{\sum_{i=1}^{M} O_i^{(4)}} = \frac{\sum_{i=1}^{M} \psi_i^q(t) \cdot \sum_{j=0}^{n} a_{ij}^q \cdot x_j}{\sum_{i=1}^{M} \psi_i^q(t)}, \quad q = 1, \cdots, n_o \tag{2.9}$$

where $\bar{v}$ denotes the consequent value and $a$ denotes the parameters.
For the functional-link-based IRSFNN, the output is

$$y_q = O^{(6)} = \frac{\sum_{i=1}^{M} O_i^{(4)} \tilde{O}_i^{(5)}}{\sum_{i=1}^{M} O_i^{(4)}} = \frac{\sum_{i=1}^{M} \psi_i^q(t) \cdot \sum_{k=0}^{n_t} a_{ik}^q \cdot \varphi_j}{\sum_{i=1}^{M} \psi_i^q(t)}, \quad q = 1, \cdots, n_o \qquad (2.10)$$

where $\tilde{v}$ denotes the consequent value and $a$ denotes the parameters.

## 2.3 IRSFNN Learning

In this section, two phase learning is used for constructing the IRSFNN. There are no rules in an IRSFNN. All of the recurrent fuzzy rules evolve from the simultaneous structure and parameter learning after receiving each piece of training data. Fig. 2.3 presents flowchart of the IRSFNN's learning scheme. The parameter learning phase describes the use of a gradient descent algorithm and a variable-dimensional Kalman filter algorithm.

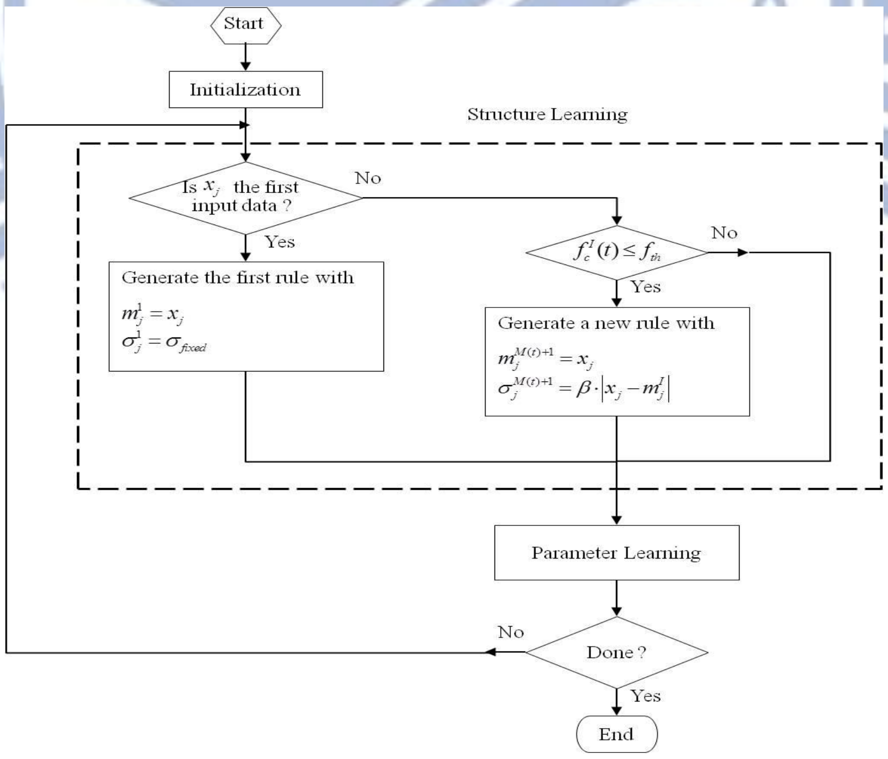

Fig. 2.3. Flowchart of the structure and parameter learning of the IRSFNN.

### 2.3.1 Structure Learning

The first task in structure learning is to determine whether a new rule should be extracted from the training data and to determine the number of fuzzy sets in the universe of discourse of each input variable because one cluster in the input space corresponds to one potential fuzzy rule, in which $m_j^i$ represents the mean and $\sigma_j^i$ represents the variance of that cluster. The spatial firing strength $\phi^i$ in (2.4) is used to determine whether a new rule should be generated. The first incoming data point **x** is used to generate the first fuzzy rule, and the mean and width of the fuzzy membership functions associated with this rule are set as:

$$m_j^1 = x_j \text{ and } \sigma_j^1 = \sigma_{fixed}\,,\ \ j = 1,\cdots,n \tag{2.11}$$

where $\sigma_{\text{fixed}}$ is a predefined value (we use $\sigma_{\text{fixed}}$ =0.3 in this paper) that determines the width of the memberships associated with a new rule. For subsequent new incoming data **x**(t) we find

$$I = \arg \max_{1 \le i \le M(t)} f_c^i(t) \tag{2.12}$$

where M(t) is the number of existing rules at time *t*. If $f_c^I(t) \le f_{th}$ ( $f_{th}$ is a pre-specified threshold), then a new fuzzy rule is generated and M(t+1)=M(t)+1. In this approach, if the present data do not match well according to the existing rules, then a new rule is evolved. We also use the same procedure to assign the mean of fuzzy sets as we have done for first rule. For a new rule, the mean and width of corresponding fuzzy sets are defined as

$$m_j^{M(t)+1} = x_j \text{ and } \sigma_j^{M(t)+1} = \beta \cdot \left| x_j - m_j^I \right|,\ \ j = 1,\cdots,n \tag{2.13}$$

where $\beta$ is an overlap coefficient. Eq. (1.13) indicates that the initial width is equal to the Euclidean distance between current input data $x$ and the center of the best matching rule for this data point times an overlapping parameter $\beta$. In this study $\beta$ is set to 0.5, so that the width of new fuzzy set is half of the Euclidean distance from the best matching center, and a suitable overlap between adjacent rules is realized.

## 2.3.2 Parameter Learning

In addition with the structure, all free parameters in an IRSFNN are also learned, including those newly generated and previously existing. For clarification, we consider the single-output case and define the objective to minimize the error function as

$$E = \frac{1}{2}\left[ y_q(t) - y_d(t) \right]^2 \tag{2.14}$$

where $y_q(t)$ represents the IRSFNN output and $y_d(t)$ represents the desired output. Parameters in the consequent part of the TSK-type IRSFNN are learned based on the variable-dimensional Kalman filter algorithm as discussed in [14]. According to [14], Eq. (2.10) of a functional-link-based IRSFNN can be re-written as

$$y_q = \vec{\psi}(t)_{FuL}^{T} \vec{a}_{FuL} \tag{2.15}$$

where

$$\vec{\psi}(t)_{FuL}^{T} = \left[ \overbrace{\frac{\psi_1^q(t)}{\sum_{i=1}^{M}\psi_i^q(t)}\varphi_0, \cdots, \frac{\psi_1^q(t)}{\sum_{i=1}^{M}\psi_i^q(t)}\varphi_{n_t}}^{n_t+1}, \cdots, \overbrace{\frac{\psi_M^q(t)}{\sum_{i=1}^{M}\psi_i^q(t)}\varphi_0, \cdots, \frac{\psi_M^q(t)}{\sum_{i=1}^{M}\psi_i^q(t)}\varphi_{n_t}}^{n_t+1} \right] \in \Re^{1\times(n_t+1)\times M} \tag{2.16}$$

and

$$\vec{a}_{FuL} = \left[ a_{10}^q, \cdots, a_{1n_t}^q, \cdots, a_{M0}^q, \cdots, a_{Mn_t}^q \right]^T \in \Re^{M\times(n_t+1)\times 1} \tag{2.17}$$

and

$$[\varphi_0, \varphi_1, \cdots, \varphi_{n_t}] = [1, x_1, \sin(\pi x_1), \cos(\pi x_1), \cdots, x_n, \sin(\pi x_n), \cos(\pi x_n)] \tag{2.18}$$

The consequent parameter vector $\vec{a}_{FuL}$ is updated by executing the following variable-dimensional Kalman filtering algorithm:

$$\vec{a}_{FuL}(t+1) = \vec{a}_{FuL}(t) + S(t+1)\vec{\psi}_{FuL}(t+1)(y_d(t+1) - \vec{\psi}_{FuL}(t+1)\vec{a}_{FuL}(t)),$$
$$S(t+1) = \frac{1}{\kappa}\left[S(t) - \frac{S(t)\vec{\psi}_{FuL}(t+1)\vec{\psi}^T_{FuL}(t+1)S(t)}{\kappa + \vec{\psi}^T_{FuL}(t+1)S(t)\vec{\psi}_{FuL}(t+1)}\right] \tag{2.19}$$

where $\kappa$ is a forgetting factor and lies in [0,1] ( $\kappa$ =0.99995 in this paper). Once a new rule is generated, the dimension of the vectors $\vec{a}_{FuL}$ , $\vec{\psi}_{FuL}$ , and the matrix $S$ increases accordingly. When a new rule evolves at time t+1, the new vector $\vec{\psi}_{FuL}(t+1)$ becomes

$$\vec{\psi}(t+1)^T_{FuL} = \left[\overbrace{\frac{\psi_1^q(t)}{\sum_{i=1}^{M}\psi_i^q(t)}\varphi_0, \cdots, \frac{\psi_1^q(t)}{\sum_{i=1}^{M}\psi_i^q(t)}\varphi_{n_t}}^{n_t+1}, \cdots, \overbrace{\frac{\psi_{M+1}^q(t)}{\sum_{i=1}^{M}\psi_i^q(t)}\varphi_0, \cdots, \frac{\psi_{M+1}^q(t)}{\sum_{i=1}^{M}\psi_i^q(t)}\varphi_{n_t}}^{n_t+1}\right] \in \Re^{1\times(n_t+1)\times(M+1)} \tag{2.20}$$

An IRSFNN augments $\vec{a}_{FuL}(t)$ and $S(t)$ on the right-hand side of Eq. (2.17) as follows:

$$\vec{a}_{FuL\_new} = \left[\vec{a}_{FuL},\ a^q_{(M+1)0}, \cdots, a^q_{(M+1)n_t}\right]^T \in \Re^{(M+1)\times(n_t+1)\times 1} \tag{2.21}$$

and

$$S_{new}(t) = block\ diag[S(t)\ C\cdot I] \in \Re^{(M+1)(n_t+1)\times(M+1)(n_t+1)} \tag{2.22}$$

where $C$ is a large positive constant (we use C=10).

This paper uses resetting operations to keep S bounded and to avoid divergence problems. After a period of training, the matrix S is re-set as $C \cdot I$ . Simulation results in Section 2.4 show that the learning of the IRSFNN achieves good training and test performance. A gradient descent algorithm tunes the antecedent parameters of the IRSFNN. This gradient descent algorithm is performed once for each piece of incoming datum.

By using a gradient descent algorithm for the updated recurrent weights, we have

$$\lambda^q_{ik}(t+1) = \lambda^q_{ik}(t) - \eta\frac{\partial E}{\partial \lambda^q_{ik}} \tag{2.23}$$

where $\eta$ is the learning rate and

$$\frac{\partial E}{\partial \lambda_{ik}^q} = \frac{\partial E}{\partial y_q}\frac{\partial y_q}{\partial \psi_i^q}\frac{\partial \psi_i^q}{\partial \lambda_{ik}^q}$$
$$= (y_q - y_d)\cdot(v_i^q - y_q)\cdot(\psi_k^q(t-1) - \phi^i(t)) / \sum_{i=1}^{M}\psi_i^q(t) \tag{2.24}$$

The antecedent part of parameter $m_j^i$ is updated as

$$m_j^i(t+1) = m_j^i(t) - \eta\frac{\partial E}{\partial m_j^i} \tag{2.25}$$

where

$$\frac{\partial E}{\partial m_j^i} = \frac{\partial E}{\partial y_q}\frac{\partial y_q}{\partial \psi_i^q}\frac{\partial \psi_i^q}{\partial \phi^i}\frac{\partial \phi^i}{\partial \mu_j^i}\frac{\partial \mu_j^i}{\partial m_j^i}$$
$$= (y_q - y_d)\cdot\frac{(v_i^q - y_q)}{\sum_{i=1}^{M}\psi_i^q(t)}\cdot(1-\gamma_i^q)\cdot\phi^i\cdot\frac{2(x_j - m_j^i)}{(\sigma_j^i)^2} \tag{2.26}$$

The antecedent part of parameter $\sigma_j^i$ is updated as

$$\sigma_j^i(t+1) = \sigma_j^i(t) - \eta\frac{\partial E}{\partial \sigma_j^i} \tag{2.27}$$

where

$$\frac{\partial E}{\partial \sigma_j^i} = \frac{\partial E}{\partial y_q}\frac{\partial y_q}{\partial \psi_i^q}\frac{\partial \psi_i^q}{\partial \phi^i}\frac{\partial \phi^i}{\partial \mu_j^i}\frac{\partial \mu_j^i}{\partial \sigma_j^i}$$
$$= (y_q - y_d)\cdot\frac{(v_i^q - y_q)}{\sum_{i=1}^{M}\psi_i^q(t)}\cdot(1-\gamma_i^q)\cdot\phi^i\cdot\frac{2(x_j - m_j^i)^2}{(\sigma_j^i)^3} \tag{2.28}$$

## 2.4 Simulation Results

This section presents five examples to assess the performance of IRSFNN and MRIT2NFS. These simulation studies include two types of dynamic system problems (Examples 1–2) and three types of prediction problems (Examples 3–5). These examples are also used to compare the performance of the IRSFNN with those of existing recurrent FNNs. For dynamic system identification, the recurrent structure in the proposed approach shows the advantages, as listed in Tables 2.1–2.5.

## 2.4.1. Example 1 (Dynamic System Identification)

This example uses an IRSFNN to identify a nonlinear dynamic system, which is a nonlinear plant with multiple time delays that has been studied in [7]. The dynamic system is described by the following difference equation

$$y_p(t+1) = f(y_p(t), y_p(t-1), y_p(t-2), u(t), u(t-1)) \tag{2.29}$$

where

$$f(x_1, x_2, x_3, x_4, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) x_4}{1 + x_2^2 + x_3^2} \tag{2.30}$$

The dynamic system output depends on three previous outputs and two previous inputs. In this study, only two current values, $u(t)$ and $y_p(t)$, are fed as input to the IRSFNN input layer. Here, we do not use extra lagged values ($n_u$=0) in the consequent part. The desired output of the IRSFNN is $y_d(t+1)$. In the training procedure of the IRSFNN, we follow the same computational protocols as in [7] and [14], i.e., we use only 10 epochs, with 900 time steps in each epoch. In each epoch, the first 350 inputs are random values uniformly distributed over [-2, 2] and the remaining 550 training inputs are generated from a sinusoid, $1.05\sin(\pi t/45)$.

We follow this strategy for an online training process because a similar procedure was followed in [14], where the total number of online training time steps is 9000. The structure learning threshold $f_{th}$ decides the number of rules to be generated. After training, three rules are generated when the structure learning threshold is set to 0.01. Table 2.1 shows the root–mean-squared error (RMSE) of training data. The testing input signal $u(t)$ is guided by

$$u(t)=\begin{cases}\sin(\frac{\pi t}{25}), & t<250\\ 1.0, & 250\le t<500\\ -1.0, & 500\le t<750\\ 0.3\sin(\frac{\pi t}{25})+0.1\sin(\frac{\pi t}{32}) & \\ \quad +0.6\sin(\frac{\pi t}{10}), & 750\le t<1000\end{cases} \tag{2.31}$$

Fig. 2.4 shows a comparison of the actual output with the output produced by the IRSFNN for the test input. Fig. 2.5 shows the error difference between the actual plant output and the IRSFNN. Figs. 2.4–2.5 show a very good match, suggesting that IRSFNN architecture combined with the proposed system identification scheme adequately identifies the dynamic system with feedback.

Table 2.1 shows the performance of the IRSFNN compared with the other recurrent networks, including a recurrent self-organizing neural fuzzy inference network (RSONFIN) [2], a wavelet recurrent fuzzy neural network (WRFNN) [11], a TSK-type recurrent fuzzy network (TRFN) [7], a HO-RNFS [6], and a recurrent self-evolving fuzzy neural network with local feedback (RSEFNN-LF) [14].

The consequent part in the RSEFNN-LF is composed by a first-order TSK-type that performs a linear combination of input variables. As in the IRSFNN, all these networks use the same information including the number of input variables, training data, test data, and training epochs. For a fair comparison, the total number of parameters of the IRSFNN is kept similar to that of the compared networks. The result indicates that the IRSFNN achieves better identification than the other recurrent networks.

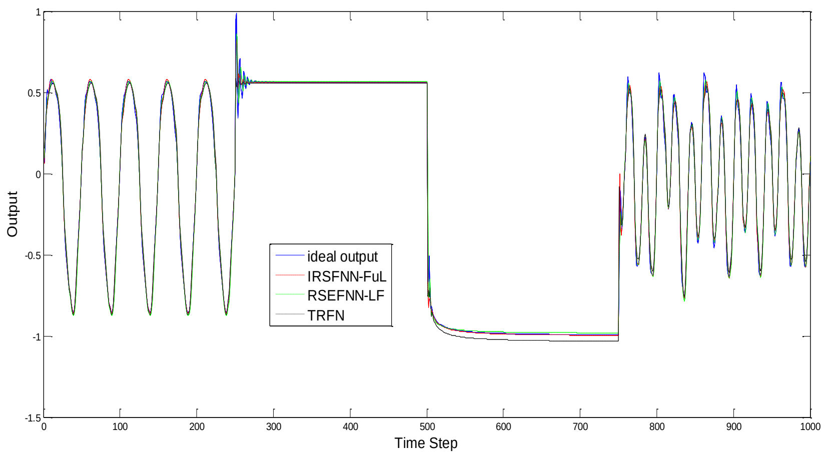

Fig. 2.4. Outputs of the dynamic plant (blue line), IRSFNN (red line), RSEFNN_LF (green line), and TRFN (black line) in Example 1.

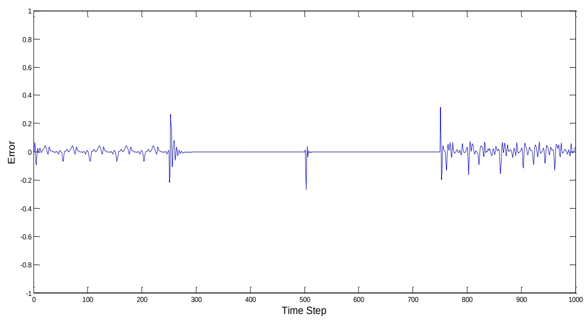

Fig. 2.5. Test errors between the MRIT2FNN and actual plant outputs.

TABLE 2.1 PERFORMANCE OF IRSFNN AND OTHER RECURRENT MODELS FOR DYNAMIC SYSTEM IDENTIFICATION IN EXAMPLE 1

| Models | RSONFIN [2] | WRFNN [11] | HO-RNFS [6] | TRFN [7] | RSEFNN-LF [14] | IRSFNN (TSK) | IRSFNN (FuL) |
|---|---|---|---|---|---|---|---|
| Rules | 4 | 5 | 3 | 3 | 4 | 3 | 3 |
| Number of parameters | 36 | 55 | 45 | 33 | 32 | 30 | 42 |
| Training RMSE | 0.025 | 0.064 | 0.054 | 0.032 | 0.020 | 0.015 | 0.011 |
| Test RMSE | 0.078 | 0.098 | 0.082 | 0.047 | 0.040 | 0.036 | 0.031 |

*FuL denotes Functional-Link-based

### 2.4.2. Example 2 (Dynamic System Identification)

This example considers the use of the IRSFNN for dynamic system identification with longer input delays that is described by

$$y_p(t+1)=0.72y_p(t)+0.025y_p(t-1)u_1(t-1)+0.01u_1^2(t-2)+0.2u_1(t-3) \tag{2.32}$$

This plant is the same as the one used in [7]. This plant output depends on four previous inputs and two previous outputs. As shown in Example 1, the current variables $u(t)$ and $y_p(t)$ are fed as inputs to the IRSFNN input layer. In this example, we do not use extra lagged values ($n_u$=0) in the consequent part. The training data and time steps are the same as those used in Example 1. When the structure learning threshold $f_{th}$ is set to 0.05, three rules are generated. The test signal used in Example 1 is also adopted here to assess the identified system. Fig. 2.6 shows the outputs of the plant and the IRSFNN for these test inputs. Fig. 2.7 shows the test error between the outputs of the IRSFNN and the desired plant. Table 1.2 shows the number of rules, total number of parameters, and training and test RMSEs of the IRSFNN. The performance of the IRSFNN is compared with that of recurrent models, including an RSONFIN [2], a TRFN [7], a WRFNN [11], and an RSEFNN-LF [14]. These models use identical numbers of input variables, training data, test data, and training epochs as designed by the IRSFNN. For a fair comparison, the numbers of parameters in the IRSFNN have been kept similar to those in these compared models.

Apparently in Table 2.2, the RSEFNN-LF only uses local source, which is not enough to capture critical information for the system, thus the test error of the IRSFNN_TSK is lower than that of the RSEFNN-LF, even using fewer rules. Here, we also investigate the performance comparison of the IRSFNN-TSK and the IRSFNN-FuL, and results show that the IFSFNN-FuL achieves better performance. Finally, the results show that the test RMSEs of the IRSFNN-FuL and IRSFNN-TSK are smaller than those of the other networks.

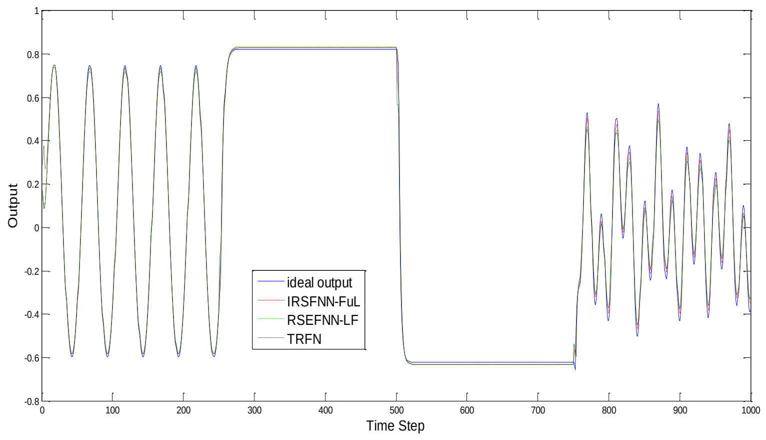

Fig. 2.6. Outputs of the dynamic plant (blue line) and IRSFNN-FuL (red line) in Example 2.

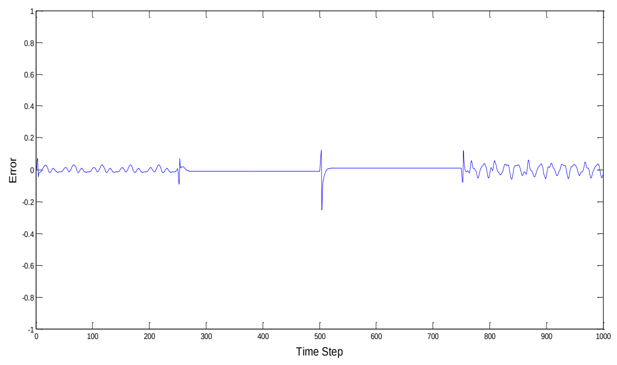

Fig. 2.7. Test errors between the IRSFNN-FuL and actual plant outputs.

TABLE 2.2 PERFORMANCE OF IRSFNN AND OTHER RECURRENT MODELS FOR DYNAMIC SYSTEM IDENTIFICATION IN EXAMPLE 2

| Models | RSONFIN [2] | WRFNN [11] | TRFN [7] | RSEFNN-LF [14] | IRSFNN (TSK) | IRSFNN (FuL) |
|---|---|---|---|---|---|---|
| Rules | 6 | 5 | 3 | 4 | 3 | 2 |
| Number of Parameters | 36 | 55 | 33 | 30 | 30 | 26 |
| Training RMSE | 0.03 | 0.057 | 0.007 | 0.016 | 0.014 | 0.011 |
| Test RMSE | 0.06 | 0.083 | 0.031 | 0.028 | 0.026 | 0.022 |

## *2.4.3. Example 3 (Chaotic Series Prediction)*

As introduced in [30], the IRSFNN is applied to predict the Henon chaotic sequence of a dynamic system with one delay and two sensitive parameters generated by the following equation:

$$y_p(t+1) = \text{-}P \cdot y_p{}^2(t) + Q \cdot y_p(t-1) + 1.0 \qquad (2.33)$$

Eq. (2.33), with P=1.4 and Q=0.3, produces a chaotic attractor. The initial states $[y_p(1), y_p(0)] = [0.4, 0.4]$ generate 2000 patterns, with the 1000 patterns used for training and the remaining 1000 patterns used for testing. In this example, we do not use extra lagged values ( $n_u$ =0) in the consequent part. The training procedure uses the plant output $y_p(t+1)$ as the desired output $y_d(t+1)$. The system has a single output so that only output variable $y_p(t)$ is fed as input to the IRSFNN. The training epoch in the IRSFNN is set to 90. The structure learning threshold $f_{th}$ is set to 0.2 and number of rules generated is 5 after the training procedure. Fig. 2.8 shows the phase plot of the actual and IRSFNN predicted results for the test patterns

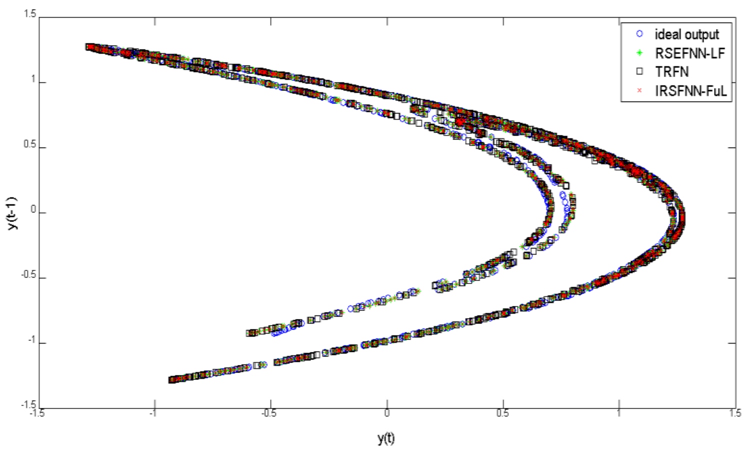

Fig. 2.8. Results of the phase plot for the chaotic system(blue), RSEFNN_FL (green), TRFN(black) and IRSFNN-FuL(red).

Table 2.3 includes the network size, parameter numbers, and training and test RMSEs of the IRSFNN. The TSK-type IRSFNN and functional-link-based IRSFNN both use four rules. We find that the latter achieves greater learning performance. In a meaningful comparison, the number of parameters of the IRSFNN must be similar to that of compared models. The compared recurrent models include a recurrent FNN [12], a wavelet recurrent FNN [13], a TSK-type recurrent fuzzy network [9], and a recurrent self-evolving FNN with local feedback [16], where the locally recurrent is a simple structure but its performance is superior to that of other compared models. The training epochs, training data and test data of the compared models are identical to the conditions of the IRSFNN. Table 2.3 shows that the IRSFNN exhibits the best performance by using an interactively recurrent structure.

TABLE 2.3 PEFORMANCE OF IRSFNN AND OTHER RECURRENT MODELS FOR CHAOTIC SEQUENCE PREDICTION IN EXAMPLE 3

| Models | RFNN [10] | WRFNN [11] | TRFN-S [7] | RSEFNN-LF [14] | IRSFNN (TSK) | IRSFNN (FuL) |
|---|---|---|---|---|---|---|
| Rules | 15 | 7 | 6 | 9 | 4 | 3 |
| Number of Parameters | 60 | 70 | 66 | 45 | 32 | 40 |
| Training RMSE | 0.463 | 0.191 | 0.028 | 0.032 | 0.017 | 0.016 |
| Test RMSE | 0.469 | 0.188 | 0.027 | 0.023 | 0.015 | 0.014 |

### 2.4.4. Example 4 (Mackey-Glass Chaotic Series Prediction)

The time series prediction problem used in this example is the well-known Mackey-Glass chaotic series. The Mackey-Glass time series is generated from the delay differential equation:

$$\frac{dx(t)}{dt}=\frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)}-0.1x(t) \tag{2.34}$$

where $\tau$ =17 and the initial value is given as $x(0)$ =1.2. Four past values are used to predict $x(t)$, and the input-output data format is $[x(t-24),x(t-18),x(t-12),x(t-6);x(t)]$. As discussed in [3, 9, 14, 16], 1000 patterns are generated from t=124 to t=1123, with the first

500 patterns being used for training and the remaining 500 for testing. The IRSFNN's training epoch is set to 500, and its structure threshold $f_{th}$ is set to 0.001. After 500 epochs of training procedure, seven rules are generated. The four input dimensions contain 28 fuzzy sets. Fig. 2.9 displays the prediction results of the IRSFNN-FuL, and Fig. 2.10 shows the prediction error between desired output and the IRSFNN. Figs. 2.9–2.10 show an excellent match, suggesting that the proposed scheme in the network indicates and excellent ability to predict the Mackey-Glass time series. Table 2.4 shows the performance, including rules, a total number of parameters, and training and test RMSE for the TSK-type and Functional-ink-based IRSFNNs.

Table 2.4 lists the performance comparison of the IRSFNN with recently developed fuzzy systems designed by particle swarm algorithms or neural learning ([1], [7], [12], [14], and [31–37]). Both local linear wavelet NN (LLWNN) [36] and fuzzy wavelet NN (FWNN) [37] employ the wavelet neural network in the consequent part, which has the ability to localize in both time and frequent space. The compared models with particle swarm algorithm were proposed as a clustering-aided simplex particle swarm optimization (CSPSO) [34], a self-evolving evolutionary learning algorithm (SEELA) designed for neural fuzzy inference system [32] and FLNFN-CCPSO [35]. The FLNFN-CCPSO also uses the function-link-based neural network to the consequent part in the FLNFN-CCPSO.

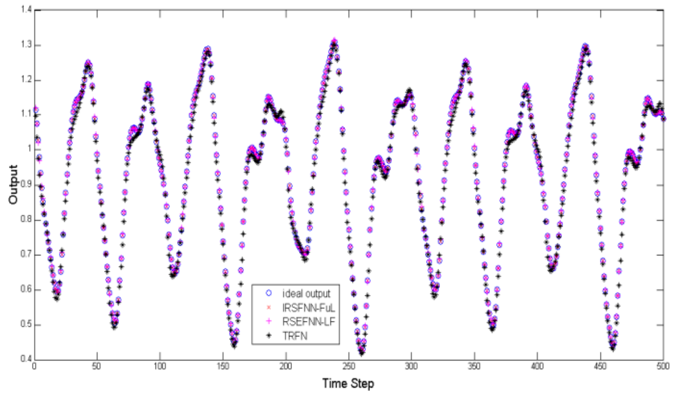

Fig. 2.9. Test result of chaotic series prediction using IRSFNN-FuL.

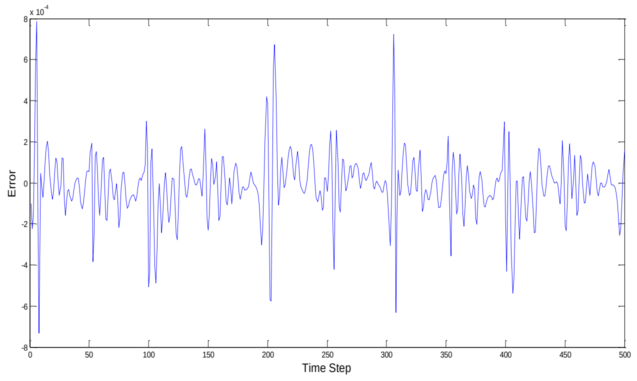

Fig. 2.10. Prediction errors between the IRSFNN-FuL and actual outputs in Example 4.

TABLE 2.4 PERFORMANCE OF IRSFNN AND OTHER MODELS FOR MACKEY-GLASS CHAOTIC SEQUENCE PREDICTION PROBLEM IN EXAMPLE 4

| Models | Rules | Number of parameters | Train RMSE | Test RMSE |
|---|---|---|---|---|
| D-FNN [12] | 10 | 100 | – | 0.0082 |
| G-FNN [1] | 10 | 90 | – | 0.0056 |
| Recurrent ANFIS [31] | – | – | – | 0.0013 |
| SEELA [32] | 9 | 198 | 0.0067 | 0.0068 |
| SuPFuNIS [33] | 10 | 94 | – | 0.0057 |
| TRFN-S [7] | 5 | 95 | – | 0.0124 |
| CSPSO [34] | 10 | 104 | – | 0.0064 |
| FLNFN-CCPSO [35] | – | – | 0.0083 | 0.0084 |
| LLWNN+Hybrid [36] | – | 110 | 0.0033 | 0.0036 |
| FWNN [37] | 16 | 128 | 0.0023 | 0.0023 |
| RSEFNN-LF [14] | 9 | 94 | 0.0032 | 0.0031 |
| IRSFNN (TSK) | 5 | 90 | 0.0040 | 0.0039 |
| IRSFNN (FuL) | 4 | 100 | 0.0002 | 0.0002 |

The proposed models, especially the IRSFNN-FuL, show superior performance to compared models. Although the performance of the IRSFNN-TSK is similar to that of

RSEFNN-LF and FWNN, rule number used in the IRSFNN-TSK is fewer than those in RSEFNN-LF and FWNN. The FWNN does not use recurrent structure to memorize previous states and only considers wavelet characteristic in the consequent part to address dynamic systems. Hence larger rules should be taken by FWNN to obtain good performance. Unlike the FWNN's consequent, the consequent of IRSFNN-TSK uses a simple structure of linear combination of input variables. For a fair comparison of using nonlinear system in the consequent part, the test RMSE of IRSFNN-FuL is 11 times lower than that of FWNN. Finally, our proposed IRSFNN-FuL has obtained the best performance among the competitors.

### *2.4.5. Example 5 (Prediction of Box-Jenkins Time Series)*

In this example, we consider the use of a real word data set to assess the IRSFNN performance. Many literatures [31, 36–41] use Box-Jenkins time series to assess the performance of real word data. Box-Jenkins time series data (gas furnace data) was downloaded from UCI repository, which were recorded from a combustion process of a methane-air mixture [36, 42], describes the operation of a gas furnace process with a gas flow rate $u(t)$ and a concentration of $CO_2$ $y(t)$. To predict the process, $u(t-4)$ and $y(t-1)$ are fed as inputs to the IRSFNN for predicting output $y(t)$. If the appropriate lag number $n_u$ is known in advance, then more past values can be included in the IRSFNN-FuL consequent part for obtaining a greater performance. Therefore, the past value $u(t-3)$, i.e., $n_u$=1, is used for the IRSFNN-FuL consequent part. The Box-Jenkins time series data provide the 296 available input-output pairs.

For a meaningful comparison, the training samples from the first 200 pairs are used, and the remaining 92 pairs are used for the test samples to predict IRSFNN performance. The structure learning threshold $f_{th}$ is set at 0.01, and the learning factor $\eta$ is set at 0.08. After 100 training epochs of an IRSFNN-FuL, four rules are generated. The training epoch in the

IRSFNN is the same as that in these existing models. As in an IRSFNN, the compared models, except the Recurrent ANFIS [31], LLWNN [36] and HyFIS [38], utilize the identical data set which is normalized. The three models, Recurrent ANFIS, LLWNN and HyFIS, use identical data set but with a scaled down output to estimate the performance. Hence they could obtain a lower test RMSE than other compared models. We could assume that the performance of IRSFNN_TSK is superior to that of the above models in terms of the same training and test data set. Fig. 2.11 shows the prediction results of the IRSFNN. Fig. 2.12 displays the prediction error between the actual time-series output and the IRSFNN output. As shown in Example 4, Table 2.5 lists the parameter numbers, training RMSE, and test RMSE. Table 2.5 also shows rules, a total number of network parameters, and training and test error of these compared models, including an HyFIS [38], a local linear wavelet NN with hybrid learning (LLWNN+hybrid) [36], a recurrent ANFIS [31], a tree-based neural fuzzy inference system (TNFIS) [39], a Fuzzy neural network (FuNN) [40], a fuzzy wavelet NN (FWNN) [37], and TSK-type recurrent fuzzy network with supervised learning (TRFN-S) [7]. As can be seen in Table 2.5, the IRSFNN-TSK utilizes fewer rules and achieves a similar performance with the FWNN. For a fair comparison in the consequent part, the test error of IRSFNN-FuL is 31 times lower than that of FWNN. Generally, the results from real world data indicate that the IRSFNN achieves smaller test RMSE than the other compared models.

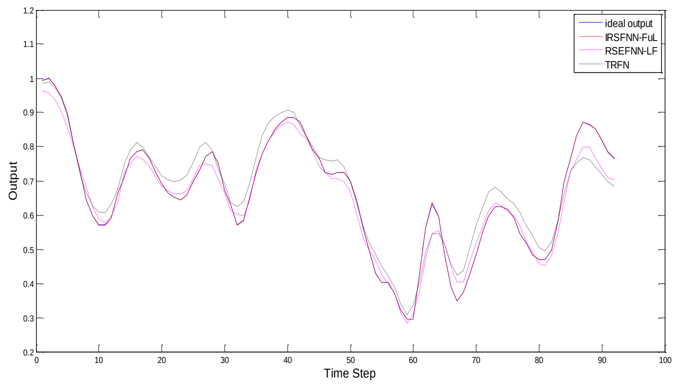

Fig. 2.11. Test result of Box-Jenkins series using IRSFNN-FuL with three rules in Example 5.

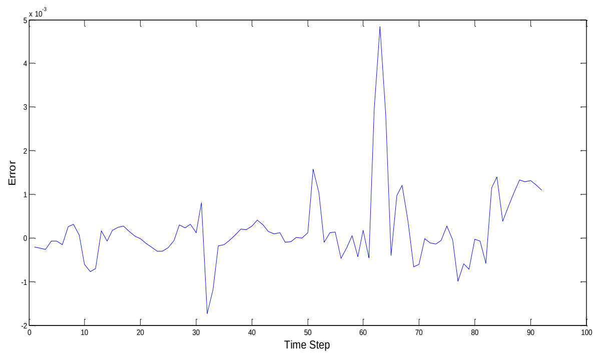

Fig. 2.12. Prediction errors between the IRSFNN-FuL and actual outputs in Example 5.

Table 2.5 PERFORMANCE OF IRSFNN AND OTHER MODELS FOR BOX-JENKINS PREDICTION IN EXAMPLE 5

| Models | Rules | Number of parameters | Train RMSE | Test RMSE |
|---|---|---|---|---|
| HyFIS [31] | – | – | – | 0.0205 |
| Recurrent ANFIS [23] | – | – | 0.006 | 0.0193 |
| TRFN-S [9] | 5 | 65 | 0.0524 | 0.0482 |
| TNFIS [32] | – | 43 | 0.0245 | 0.0230 |
| FuNN [33] | – | – | – | 0.0226 |
| LLWNN+Hybrid [28] | – | 56 | – | 0.0138 |
| FWNN [29] | 9 | 57 | 0.0189 | 0.0279 |
| RSEFNN-LF [16] | 7 | 56 | 0.0172 | 0.0344 |
| IRSFNN (TSK) | 5 | 65 | 0.0121 | 0.0297 |
| IRSFNN (FuL) | 3 | 51 | 0.00062 | 0.0009 |

# Chapter 3

# A Mutually Recurrent Interval type-2 Neural Fuzzy System (MRIT2NFS)

## 3.1 Brief Introduction of Type-2 Fuzzy Systems

This section describes the structure of a Type-2 FLS, which is a system that effectively addresses uncertainties associated with fuzzy rule base, but does not explicitly account for input measurement uncertainties. The entire structure of a type-2 FLS is exhibited in Fig. 3.1. The overall structure of s type-2 FLS is very similar to that of a type-1 FLS, the major structural difference being that the defuzzifier block of a type-1 FLS is replaced by the output processing block in a type-2 FLS. The output processing block includes type-reduction and defuzzification. The detailed mathematic function of each block is introduced below.

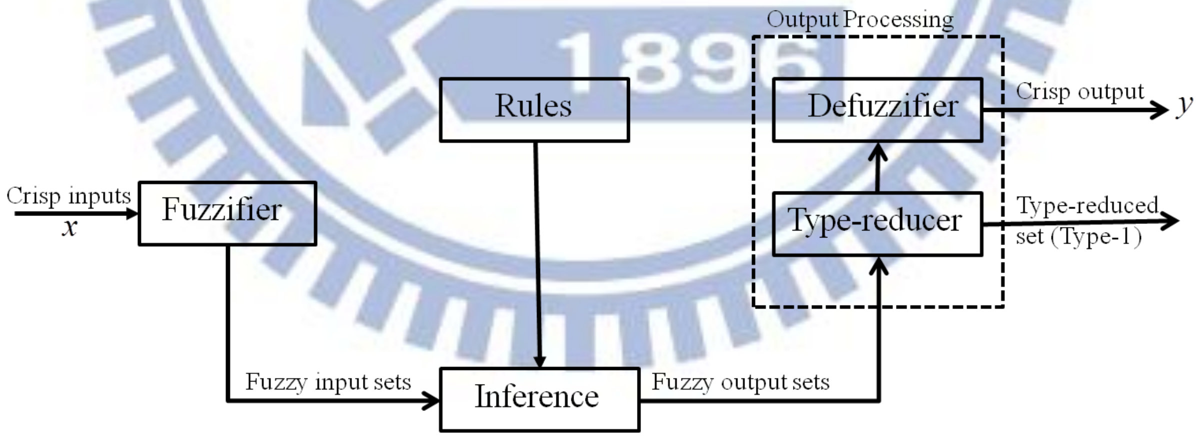

Fig. 3.1. Structure of a Type-2 FLS

**(1) Fuzzifier:** Let a crisp value become a type-2 fuzzy set. Fig 3.2 shows different types of type-2 fuzzy sets, including (a) Gaussian MF with uncertain spread, (b) Gaussian MF with uncertain mean, (c) sigmoid MF with inflection uncertainty; (d) triangular type-2

MF; (e) Granulated sigmoid MF with granulation uncertainties. In this paper, a Gaussian type-2 fuzzy set that is differentiable with uncertain mean is widely used.

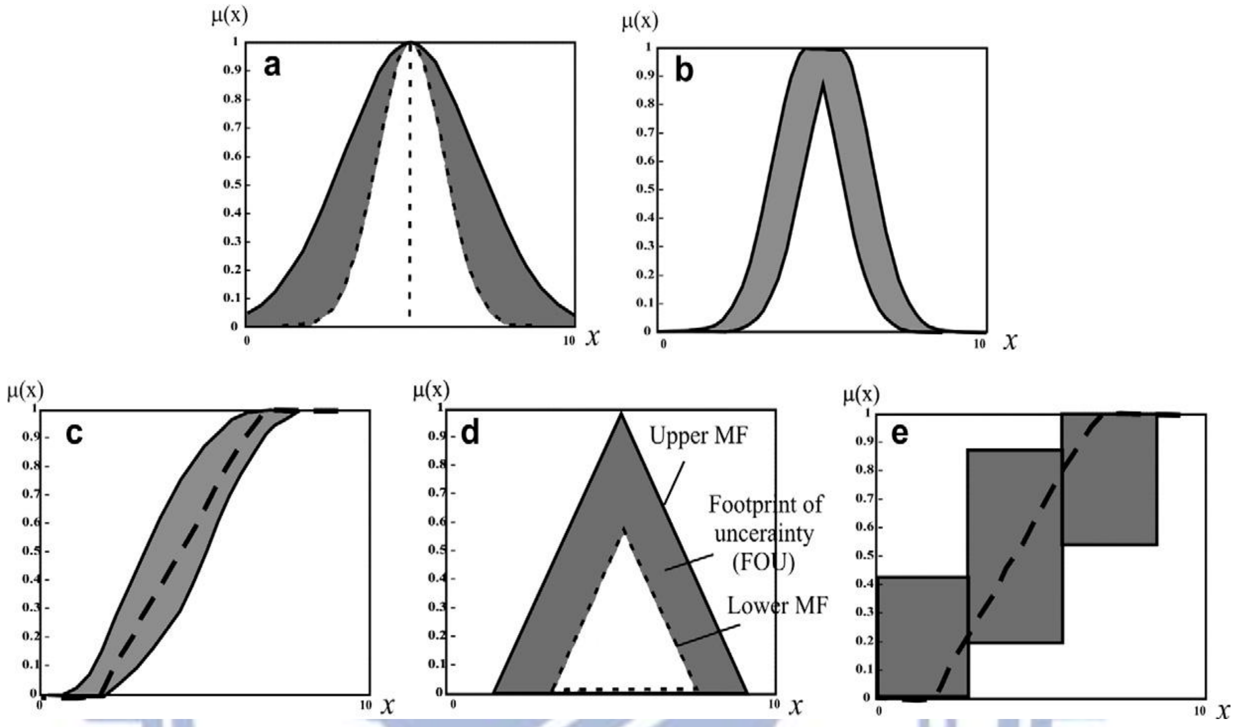

Fig. 3.2. The shaded region of FOU for different type-2 fuzzy sets; (a) Gaussian MF with uncertain spread; (b) Gaussian MF with uncertain modal value; (c) sigmoid MF with inflection uncertainty; (d) triangular type-2 MF; (e) Granulated sigmoid MF with granulation uncertainties.

**(2) Rules:** Consider a type-2 FLS have n input variables $x_1 \in X_1, \cdots, x_n \in X_n$ and one output variable $y \in Y$ and hence, the rules can be expressed as follows

$$R^i : \text{IF } x_1 \text{ is } \tilde{A}_1^i \text{ and } \cdots \text{ and } x_n \text{ is } \tilde{A}_n^i, \text{ THEN } y \text{ is } \tilde{G}^i \qquad i = 1, \cdots, M \tag{3.1}$$

where M is a total number of rules. This rule represents a type-2 relation between the input space $X_1 \times \cdots \times X_n$, and the output space, $Y$, of the type-2 FLS.

**(3) Inference Engine:** The inference engine in a type-2 FLS is very similar to that in a type-1 FLS. The inference engine combines rules and gives a mapping from input type-2 fuzzy sets to output type-2 fuzzy sets. To do this procedure need to compute unions and intersections of type-2 fuzzy sets, as well as compositions of type-2 relations. Let $\tilde{A}_1^i \times \cdots \times \tilde{A}_n^i = \tilde{F}^i$; then, Eq. (3.1) can be re-written as

$$R^i : \tilde{A}_1^i \times \cdots \times \tilde{A}_n^i \rightarrow \tilde{G}^i = \tilde{F}^i \rightarrow \tilde{G}^i \tag{3.2}$$

Therefore, the membership function can be expressed as

$$\mu_{R^i}(\mathbf{x}, y) = \mu_{\tilde{F}^i \rightarrow \tilde{G}^i}(\mathbf{x}, y) \tag{3.3}$$

In general, we only use the product or minimum t-norms for the meet. Each rule $R^i$ determines a type-2 fuzzy set $\tilde{B}^i = \tilde{A}_{\mathbf{x}} \circ R^i$ and hence, the input-output relation is as follows:

$$\mu_{\tilde{B}^i}(y) = \mu_{\tilde{B}^i} = \mu_{\tilde{A}_{\mathbf{x}} \circ R^i}(y) = \bigcup_{x \in X} [\mu_{\tilde{A}_{\mathbf{x}}}(\mathbf{x}) \mu_{R^i}(\mathbf{x}, y)] \tag{3.4}$$

**(4) Type-Reduction:** As the type-2 fuzzy output sets are derived, the system must be processed next by the output processor. The output processor consists of type-reduction and defuzzification. For the first operation of the output processor, it represents a mapping of a type-2 fuzzy set into a type-1 fuzzy set. Type-reduction methods, including centroid, center-of-sums, height, modified, and center-of-sets type-reduction, have been discussed. Here, we also concern computational complexity of type-reduction methods. Fig 3.3 illustrates the location of five type-reduction methods on a complexity scale. As seen by Fig. 3.3 the centroid and center-of-sums type-reducers are more computational load. Apparently, height and modified height type-reducers will significantly reduce the computational complexity; however, there can be a problem with only one rule generated. As a result, the center-of-sets type-reducer is superior to other type-reducers.

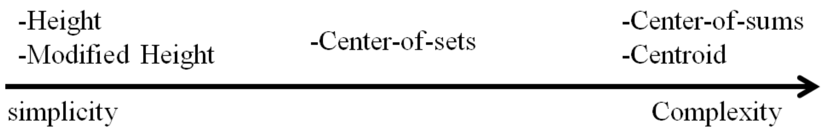

Fig. 3.3. The plots of complexity comparison with five type-reducers

**(5) Defuzzification:** An interval set is given from the completed type-reduction process, and then, we defuzzify it based on the average of $y_l$ and $y_r$. Finally, the defuzzified output of type-2 FLS is

$$y(\mathbf{x}) = \frac{y_l + y_r}{2} \tag{3.5}$$

## 3.2 MRIT2NFS Structure

This section introduces the structure of an MRIT2NFS. This multi-input multi-output (MIMO) system consists of $n_u$ inputs and $n_o$ outputs. We represent the input and output of the dynamic system by $\boldsymbol{u}$ and $\mathbf{y}_p$, respectively, where $\boldsymbol{u} = (u_1, \cdots, u_{n_u})^T$ and $\boldsymbol{y_p} = (y_{p1}, \cdots, y_{pn_o})^T$. Fig. 3.4 shows the proposed six-layered MRIT2NFS structure. The detailed function of each layer is discussed next.

*Layer 1 (Input layer):* The inputs are crisp values. Only the current state $x(t) = (u(t), y_p(t))$ is fed as input to this layer. This is in contrast to usual feed-forward FNNs where both current and some past states are fed as inputs to input layer when such networks are used to model time varying systems. To further clarify the inputs used in the system, we consider a system with one control input u(t) and one system output y(t). Then at t=1, u(1) and y(1) are used as inputs and the system output computed by the rules consequents is y(2). Similarly at t=2, u(2) and y(2) are used as inputs and the computed system output is y(3). Thus, current input and output, both are used to define the rule antecedent. Note that, this is a fan-out layer and hence there is no weight to be adjusted in this layer.

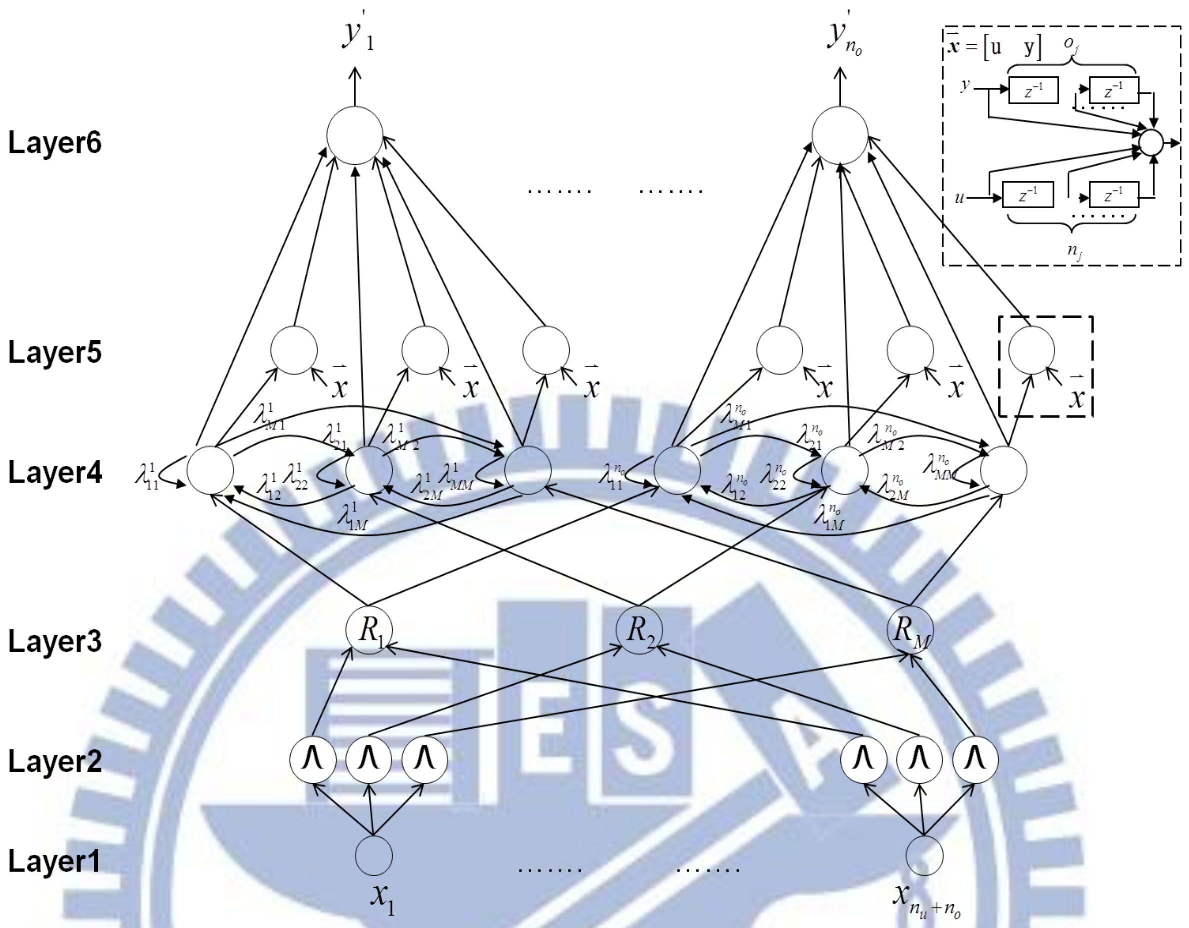

Fig. 3.4. The proposed six-layer MRIT2NFS structure, where each recurrent fuzzy rule in layer 4 forms an internal feedback and an interaction loop and each node in layer 5 performs a linear combination of current and lagged network inputs.

*Layer 2 (MF layer):* Each node in this layer performs fuzzification of one of the ($n_u$ + $n_o$) input variables using an interval type-2 membership function (MF), where $n_u$ and $n_o$ are the numbers of control inputs and system outputs. For the *i*th interval type-2 fuzzy set $\tilde{A}_j^i$ on the input variable $x_j$, $j = 1,...,(n_u + n_o)$, a Gaussian primary MF having a fixed standard deviation σ and an uncertain mean that takes on values in $[m_1, m_2]$ is used as shown in Eq. (3.6):

$$\mu_{\tilde{A}_j^i} = \exp\{-\frac{1}{2}(\frac{x_j - m_j^i}{\sigma_j^i})^2\} \equiv N(m_j^i, \sigma_j^i; x_j), \quad m_j^i \in [m_{j1}^i,\ m_{j2}^i] \tag{3.6}$$

Fig. 3.5 depicts one such membership function [52]. The uncertainty associated with the primary membership can be modeled in a more general manner. For example, we can consider uncertainty about the center of the membership function as well as that about the spread of the membership function. Here, following the protocol used in most type-2 neural fuzzy systems in the literature, we assumed that there is uncertainty only about the mean (the mean is not known with certainty) of the Gaussian membership function. This choice yields an interval type-2 fuzzy set as depicted in Fig. 3.5.

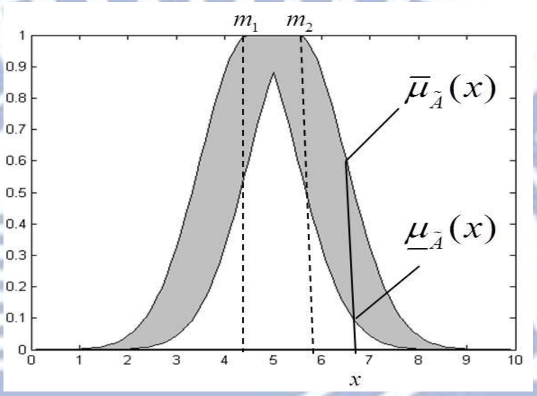

Fig. 3.5. An interval type-2 fuzzy set with Gaussian shape whose center (mean) is not known with certainty (the mean has an uncertainty). The mean can vary between $m_1$ and $m_2$, $m_1 < m_2$.

The footprint of uncertainty (FOU) [52] (the shaded region in Fig. 3.5) of this MF can be represented by the two bounding membership functions: upper MF, $\bar{\mu}_j^i$ and lower MF, $\underline{\mu}_j^i$, where

$$\bar{\mu}_j^i(x_j)=\begin{cases} N(m_{j1}^i,\sigma_j^i;x_j), & x_j < m_{j1}^i \\ 1, & m_{j1}^i \le x_j \le m_{j2}^i \\ N(m_{j2}^i,\sigma_j^i;x_j), & x_j > m_{j2}^i \end{cases} \tag{3.7}$$

and

$$\underline{\mu}_j^i(x_j)=\begin{cases} N(m_{j2}^i,\sigma_j^i;x_j), & x_j \le \dfrac{m_{j1}^i+m_{j2}^i}{2} \\ N(m_{j1}^i,\sigma_j^i;x_j), & x_j > \dfrac{m_{j1}^i+m_{j2}^i}{2} \end{cases} \tag{3.8}$$

Thus, the output of each node can be represented by an interval $[\underline{\mu}_j^i, \overline{\mu}_j^i]$ [52]. Another popular choice for type-2 membership function is Trapezoidal MF. A general trapezoidal MF involves 8 parameters to define it while our Gaussian MF needs just three parameters and thereby drastically reduces the degrees of freedom (number of free variables) of the system. Moreover, the Gaussian function is differentiable, which helps us to use gradient based tuning methods and hence we use it here.

*Layer 3 (Spatial firing layer):* Each node in this layer represents the antecedent part of a fuzzy rule and it computes the spatial firing strength. We call it "spatial" as it does not depend on any temporal input and also we distinguish it from the "temporal" firing strength that is computed in the next layer. To compute the spatial firing strength $F^i$, each node performs a fuzzy meet operation on the inputs that it receives from layer 2 using an algebraic product operation. There are M nodes in this layer, where each node corresponds to one rule. Each node in this layer is connected with $n_u$+$n_o$ nodes of the previous layer. The structure learning process starts with no rule (M=0) and the first data point is used to generate the first rule making M=1.Then with new incoming data points, depending on how well a new data point matches with existing rules, new rules are generated. In Chapter 3.3.1 we explain in details how these M rules are generated. The spatial firing strength is an interval [45] and is computed as follows [59]:

$$F^i = [\underline{f}^i, \overline{f}^i],\ \ i = 1,\ldots,M \tag{3.9}$$

$$\overline{f}^i = \prod_{j=1}^{n_u+n_o} \overline{\mu}_j^i, \quad \underline{f}^i = \prod_{j=1}^{n_u+n_o} \underline{\mu}_j^i \tag{3.10}$$

As explained earlier, in Eq. (3.9) $M$ is the total number of rules. Use of Eq. (3.10) to compute the firing strength is probably the most common compared to the use of other T-norms [29, 46, 53, 57, 58, 60–69]. Product is also used for computing rule firing strength even with Type-1 Takagi-Sugeno type systems [2, 4, 6–10, 12, 14, 21, 40]. A first look at Eq.

(3.10) suggests that with a large number of antecedents the firing strength will approach zero. But this actually does not cause a problem for two reasons. The main reason is that the defuzzified output is computed as a convex combination of consequent values, where the weights of the convex combinations are the normalized temporal firing strength, which is computed using the firing strengths in Eq. (3.10). Moreover, for practical systems, the number of antecedent clauses involved in a rule is usually not very large. One can of course use minimum as the T-norm to compute the firing interval, but min is not differentiable (while product is) and hence it is difficult to use gradient based tuning algorithm, which we use here. The problem associated with the use of minimum can be avoided by using a softer but differentiable version of minimum [43, 44], but this makes the learning rules quite complicated. So we restrict ourselves to product only.

*Layer 4 (Temporal firing layer):* There are $M \times n_o$ nodes in this layer. Each node in this layer is a recurrent rule node, which generates an internal feedback (self-loop) and external interconnection with mutual feedbacks. As a result, the recurrent weights $\lambda$ s are represented as self-loop and interconnection weights. The output of a recurrent rule node is a temporal firing strength that depends not only on current spatial firing strength but also on the previous temporal firing strength. In order to compute the temporal firing strength using Eq. (3.11), the nodes in this layer store the immediately past temporal firing strength (i.e., each node is equipped with some memory). Once the training is starting, the initial value of the past temporal firing strength is set to zero.

The temporal firing strength $[\overline{\psi}_i^q(t), \underline{\psi}_i^q(t)]$, $i = 1, \cdots, M$ and $q = 1, \cdots, n_o$ is computed combining the spatial firing strength $F^i(t)$ and previous temporal firing strength $\psi_i^q(t-1)$ using the following equation

$$\psi_i^q(t) = \sum_{k=1} (\lambda_{ik}^q \cdot \psi_k^q(t-1)) + (1-\gamma_i^q) \cdot F^i(t) \tag{3.11}$$

where $\psi_i^q(t)$s lie in [0,1], and $\lambda_{ik}^q = \frac{C_{ik}^q}{M}$ is the rule interaction weight between itself and other rules. Here $C_{ik}^q$ is the feedback related to node i of layer 4 from node k of layer 4 that is related to the qth output node. Thus for local (internal) feedback i=k and for i not equal to k, we get external feedback to the *i*th node of layer 4 from the kth rule (antecedents in layer 4) node. When a new rule is generated, the initial value of $C_{ik}^q$ in local feedback (i.e., for i=k) , is set to 0.5 and for i not equal to k, $C_{ik}^q$ is set to 0.2. The initial $C_{ik}^q$ may be adjusted for different examples to yield better results. Thus the total feedback to the ith rule node (Layer 4) for the qth output node is $\gamma_i^q = \sum_{k=1}^{M} \lambda_{ik}^q$ . The interval in Eq. (3.11) now may be written as

$$[\overline{\psi}_i^q(t), \underline{\psi}_i^q(t)] = \sum_{k=1}^{M} \lambda_{ik}^q \cdot [\overline{\psi}_k^q(t-1), \underline{\psi}_k^q(t-1)] + (1-\gamma_i^q)\cdot[\overline{f}^i(t), \underline{f}^i(t)] \tag{3.12}$$

where

$$\overline{\psi}_i^q(t) = \sum_{k=1} (\lambda_{ik}^q \cdot \overline{\psi}_k^q(t-1)) + (1-\gamma_i^q)\cdot \overline{f}^{\mathrm{i}}(t) \tag{3.13}$$

and

$$\underline{\psi}_i^q(t) = \sum_{k=1} (\lambda_{ik}^q \cdot \underline{\psi}_k^q(t-1)) + (1-\gamma_i^q)\cdot \underline{f}^i(t). \tag{3.14}$$

Note that for the first epoch of the online learning, i.e., for t=1, Eq. (3.11) will only use $(1-\gamma_i^q)\cdot F^i(t)$ because the initial past temporal firing strength is set to 0. For subsequent epochs, since the nodes in the layer 4 store past firing strengths, Eq. (3.11) can be computed without any problem.

*Layer 5 (Consequent layer):* Each node in this layer is called a consequent node and functions as a linear model with exogenous inputs and lagged values. The output of a consequent node is a linear combination of current input states $x(t) = (u(t), y_p(t)) = (u_1(t), \cdots, u_{n_u}(t), y_{p1}(t), \cdots, y_{pn_o}(t))$ and their lagged values

$(u_j(t-1),\cdots,u_j(t-N_j),y_{pk}(t-1),\cdots,y_{pk}(t-O_j))$. In Fig. 3.4, a consequent node is zoomed to elaborate its functioning. The output $\tilde{y}_q^i(t+1)$, $i=1,...,M$; $q=1,...,n_o$, of the $i$ th rule node connecting to the $q$ th output variable is computed as follows:

$$\tilde{y}_q^i(t+1)=\sum_{j=0}^{n_u}\sum_{k=0}^{N_j}\tilde{a}_{jkq}^i u_j(t-k)+\sum_{j=1}^{n_o}\sum_{k=0}^{O_j}\tilde{a}_{(j+n_u)kq}^i y_{pj}(t-k) \tag{3.15}$$

where $u_0(t)=1$ and $N_0=0$, $N_j$ and $O_j$ are the numbers of lagged control input $u_j(t)$ and system output $y_{pj}(t)$, respectively, and $\tilde{a}_{jkq}^i$ s are interval valued coefficient denoted by

$$\tilde{a}_{jkq}^i=[c_{jkq}^i-s_{jkq}^i,c_{jkq}^i+s_{jkq}^i] \tag{3.16}$$

where $c_{jkq}^i$ and $s_{jkq}^i$ are the center and spread of an interval type-1 set respectively. For an interval type-1 set, the membership value of every point over the interval [ $c_{jkq}^i$ - $s_{jkq}^i$, $c_{jkq}^i$ + $s_{jkq}^i$ ] is unity. Note that, in order to compute Eq. (3.15) some past values of $u$ and $y$ are needed and these values are stored at appropriate nodes in this layer. In other words, nodes in this layer have some local memory. The inclusion of lagged values of $u(t)$ and $y_p(t)$ in the linear consequent part instead of the antecedent part simplifies the computation process of the network for modeling of dynamic systems, especially when interval type-2 fuzzy sets are used. The output $\tilde{y}_q^i(t+1)$ is an interval type-1 set, denoted by $[\tilde{y}_{lq}^i,\tilde{y}_{rq}^i]$, where indices $l$ and $r$ denote left and right limits, respectively. According to Eqs. (3.15) and (3.16), the node output is

$$\begin{aligned}\tilde{y}_q^i=[\tilde{y}_{lq}^i,\tilde{y}_{rq}^i]&=\sum_{j=0}^{n_u}\sum_{k=0}^{N_j}[c_{jkq}^i-s_{jkq}^i,c_{jkq}^i+s_{jkq}^i]\cdot u_j(t-k)\\&+\sum_{j=1}^{n_o}\sum_{k=0}^{O_j}[c_{(j+n_u)kq}^i-s_{(j+n_u)kq}^i,\ c_{(j+n_u)kq}^i+s_{(j+n_u)kq}^i]\cdot y_{pj}(t-k)\end{aligned} \tag{3.17}$$

That is,

$$\tilde{y}_{lq}^i = \sum_{j=0}^{n_u}\sum_{k=0}^{N_j} c_{jkq}^i u_j(t-k) + \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} c_{(j+n_u)kq}^i y_{pj}(t-k) - \sum_{j=0}^{n_u}\sum_{k=0}^{N_j} s_{jkq}^i |u_j(t-k)| - \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} s_{(j+n_u)kq}^i |y_{pj}(t-k)|$$

(3.18)

and

$$\tilde{y}_{rq}^i = \sum_{j=0}^{n_u}\sum_{k=0}^{N_j} c_{jkq}^i u_j(t-k) + \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} c_{(j+n_u)kq}^i y_{pj}(t-k) + \sum_{j=0}^{n_u}\sum_{k=0}^{N_j} s_{jkq}^i |u_j(t-k)| + \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} s_{(j+n_u)kq}^i |y_{pj}(t-k)|$$

(3.19)

*Layer 6 (Output layer)*: Each node in this layer corresponds to one output variable. For defuzzification operation, the $q$ th output layer node computes the network output variable $y'_q$ using type-reduction. The type-reduced set is an interval set $[y'_{lq}, y'_{rq}]$. The outputs $y'_{lq}$ and $y'_{rq}$ can be computed using the Karnik-Mendel (KM) iterative procedure [45]. Using the KM procedure, we shall rewrite the expressions for $y'_{lq}$ and $y'_{rq}$ of $[y'_{lq}, y'_{rq}]$ in suitable forms so that we can derive the learning rules easily. In KM procedure, the consequent values are re-ordered in ascending order. Let $\tilde{y}_{lq}$ and $\tilde{y}_{rq}$ be the original consequent values and $\hat{y}_{lq}$ and $\hat{y}_{rq}$ be the corresponding rule ordered (in ascending order) consequent values. Then the relationship between $\tilde{y}_{lq}$, $\tilde{y}_{rq}$, $\hat{y}_{lq}$, and $\hat{y}_{rq}$ is

$$\hat{y}_{lq} = Q_l \tilde{y}_{lq} \quad \text{and} \quad \hat{y}_{rq} = Q_r \tilde{y}_{rq} \tag{3.20}$$

here $Q_l$ and $Q_r$ are $M \times M$ appropriate permutation matrices to reorder the values. Let $\overline{\psi}_q = (\overline{\psi}_q^1(t), \overline{\psi}_q^2(t) .. \overline{\psi}_q^M(t))^T$ and $\underline{\psi}_q = (\underline{\psi}_q^1(t), \underline{\psi}_q^2(t) ... \underline{\psi}_q^M(t))^T$ . According to [59], the output $y'_{lq}$ can be computed as follows:

$$y'_{lq} = \frac{\sum_{i=1}^{L}(Q_l\bar{\boldsymbol{\psi}}_q)_i\,\hat{y}_{lq}^i + \sum_{i=L+1}^{M}(Q_l\underline{\boldsymbol{\psi}}_q)_i\,\hat{y}_{lq}^i}{\sum_{i=1}^{L}(Q_l\bar{\boldsymbol{\psi}}_q)_i + \sum_{i=L+1}^{M}(Q_l\underline{\boldsymbol{\psi}}_q)_i} = \frac{\bar{\boldsymbol{\psi}}_q^T\mathbf{Q}_\mathbf{l}^\mathbf{T}\mathbf{E}_\mathbf{1}^\mathbf{T}\mathbf{E}_\mathbf{1}\mathbf{Q}_\mathbf{l}\tilde{\mathbf{y}}_\mathbf{lq} + \underline{\boldsymbol{\psi}}_q^T\mathbf{Q}_\mathbf{l}^\mathbf{T}\mathbf{E}_\mathbf{2}^\mathbf{T}\mathbf{E}_\mathbf{2}\mathbf{Q}_\mathbf{l}\tilde{\mathbf{y}}_\mathbf{lq}}{\mathbf{p}_\mathbf{l}^\mathbf{T}\mathbf{Q}_\mathbf{l}\bar{\boldsymbol{\psi}}_q + \mathbf{b}_\mathbf{l}^\mathbf{T}\mathbf{Q}_\mathbf{l}\underline{\boldsymbol{\psi}}_q} \tag{3.21}$$

where $L$ and $R$ denote the left and right crossover-over points, respectively. The end-points L and R are defined in [45]. The other vectors and matrices involved in Eq. (3.21) are defined as

$$\mathbf{p}_\mathbf{l} = (\underbrace{1,1,\cdots,1}_{L},0,\cdots,0)^T \in \Re^{M\times 1},\ \ \mathbf{b}_\mathbf{l} = (0,\ldots,0,\underbrace{1,\ldots,1}_{M-L})^T \in \Re^{M\times 1} \tag{3.22}$$

$$\mathbf{E}_\mathbf{1} = (e_1,e_2,\ldots,e_L,0,\ldots,0) \in \Re^{L\times M}\text{, and }\ \mathbf{E}_\mathbf{2} = (0,\ldots,0,\varepsilon_1,\varepsilon_2,\ldots,\varepsilon_{M-L}) \in \Re^{(M-L)\times M}, \tag{3.23}$$

where $\boldsymbol{e}_i \in \Re^{L\times 1}$ and $\boldsymbol{\varepsilon}_i \in \Re^{M-L}$ are unit vectors, whose all but ith element is zero and the ith element is one. In Eq. (3.21), $(Q_l\,\psi_q)$ produces a vector in M dimension where the components of $(Q_l\,\bar{\psi}_q)$ are permuted version of components in $\bar{\psi}_q$ . Also $(Q_l\,\bar{\psi}_q)_i$ represents the ith component of $(Q_l\,\bar{\psi}_q)$ . Thus, equation (3.24) computes a convex combination of $\bar{y}_{rq}^i; i = 1,\ldots,M$ values. The weights of the convex combination are computed from the components of $(Q_l\,\bar{\psi}_q)$, which are nothing but the temporal firing strength in Eq. (3.12).

Similarly, the output $y'_{rq}$ can be computed as follows:

$$y'_{rq} = \frac{\sum_{i=1}^{R}(Q_r\underline{\boldsymbol{\psi}}_q)_i\,\hat{y}_{rq}^i + \sum_{i=R+1}^{M}(Q_r\bar{\boldsymbol{\psi}}_q)_i\,\hat{y}_{rq}^i}{\sum_{i=1}^{R}(Q_r\underline{\boldsymbol{\psi}}_q)_i + \sum_{i=R+1}^{M}(Q_r\bar{\boldsymbol{\psi}}_q)_i} = \frac{\underline{\boldsymbol{\psi}}_q^T\mathbf{Q}_\mathbf{r}^\mathbf{T}\mathbf{E}_\mathbf{3}^\mathbf{T}\mathbf{E}_\mathbf{3}\mathbf{Q}_\mathbf{r}\tilde{\mathbf{y}}_\mathbf{rq} + \bar{\boldsymbol{\psi}}_q^T\mathbf{Q}_\mathbf{r}^\mathbf{T}\mathbf{E}_\mathbf{4}^\mathbf{T}\mathbf{E}_\mathbf{4}\mathbf{Q}_\mathbf{r}\tilde{\mathbf{y}}_\mathbf{rq}}{\mathbf{p}_\mathbf{r}^\mathbf{T}\mathbf{Q}_\mathbf{r}\underline{\boldsymbol{\psi}}_q + \mathbf{b}_\mathbf{r}^\mathbf{T}\mathbf{Q}_\mathbf{r}\bar{\boldsymbol{\psi}}_q} \tag{3.24}$$

where

$$\boldsymbol{p}_r = (\underbrace{1,1,\ldots,1}_{R},0,\ldots0)^T \in \Re^{M\times 1},\ \ \boldsymbol{b}_r = (0,\ldots,0,\underbrace{1,\ldots,1}_{M-R})^T \in \Re^{M\times 1} \tag{3.25}$$

$$\mathbf{E}_\mathbf{3} = (\boldsymbol{e}_1,\boldsymbol{e}_2,\ldots,\boldsymbol{e}_R,\mathbf{0},\ldots,\mathbf{0}) \in \Re^{R\times M},\ \ \mathbf{E}_\mathbf{4} = (\mathbf{0},\ldots,\mathbf{0},\boldsymbol{\varepsilon}_1,\boldsymbol{\varepsilon}_2,\ldots,\boldsymbol{\varepsilon}_{M-R}) \in \Re^{(M-R)\times M}, \tag{3.26}$$

and where $\boldsymbol{e}_i \in \Re^{R\times 1}$ and $\boldsymbol{\varepsilon}_i \in \Re^{M-R}$ are unit vectors (all but ith element are zero and the ith element is 1). All of unit vectors are defined in [59]. Such an expression is helpful in deriving the proposed parameter learning algorithm discussed in Chapter 3.3.2. Finally, the defuzzification operation defuzzifies the interval set $[y'_{lq}, y'_{rq}]$ by computing the average of $y'_{lq}$ and $y'_{rq}$. Hence, the defuzzified output for network output variable $y'_q$ is

$$y'_q = \frac{y'_{lq} + y'_{rq}}{2} \quad . \tag{3.27}$$

## 3.3 MRIT2NFS Learning

Initially, there is no rule in an MRIT2NFS. All of the recurrent type-2 fuzzy rules evolve from simultaneous structure and parameter learning. The following sections introduce the structure and parameter learning algorithm explicitly.

### *3.3.1 Structure learning*

The on-line rule is generated according to the structure learning algorithm. A previous study [62] utilized the rule firing strength as a criterion for type-1 fuzzy rule generation. This idea is extended to type-2 fuzzy rule generation using an MRIT2NFS. The spatial firing strength $F^i$ in Eq. (3.9) is used to decide whether a new rule should be generated. The type-2 rule firing strength is an interval. The center of the spatial firing interval, $f_c = \frac{1}{2}(\overline{f}^i + \underline{f}^i),$ is used as a criterion for rule generation. The first incoming data point **x** is used to generate the first fuzzy rule, and the uncertain mean and width of the type-2 fuzzy membership functions associated with this rule are set as :

$$[m^1_{j1}, m^1_{j2}] = [x_j - 0.1, x_j + 0.1] \text{ and } \sigma = \sigma_{\text{fixed}} \qquad j = 1, \cdots, n_u + n_o, \tag{3.28}$$

where $\sigma_{\text{fixed}}$ is a predefined value (we use $\sigma_{\text{fixed}}$ =0.3 in this paper) that determines the width of the memberships associated with a new rule. Subsequently, for each of new incoming data **x**(t) we find

$$I = \arg \max_{1 \le i \le M(t)} f_c^i(t), \tag{3.29}$$

where $M(t)$ is the number of existing rules at time $t$. If $f_c^I(t) \le f_{th}$ ($f_{th}$ is a pre-specified threshold), then a new fuzzy rule is generated. The idea is that if the present data point does not match well with any of the existing rules, then a new rule is generated. Here also we use the same procedure to assign the uncertain mean as done for the very first rule; i.e., for the new rule, the means of the corresponding type-2 fuzzy sets are defined as

$$[m_{j1}^{M(t)+1},\ m_{j2}^{M(t)+1}] = [x_j(t) - 0.1,\ x_j(t) + 0.1] \quad ; j=1, \ldots, n_u+n_o. \tag{3.30}$$

The width of the each fuzzy set associated with a new rule is defined as follows:

$$\sigma^{M(t)+1} = \beta \cdot \left(\sum_{j=1}^{n_u+n_o} \left(x_j - \frac{m_{j1}^I + m_{j2}^I}{2}\right)^2\right)^{\frac{1}{2}} \tag{3.31}$$

In Eq. (3.29), *I* is the index of the best matching rule, and $m_{j1}^I$ and $m_{j2}^I$ are the means of the membership function of the *j*th antecedent clause of the *I*th Rule. Eqs. (3.28) and (3.30) indicate that for the mean, the width of the uncertain region is 0.2. If the uncertainty associated with the mean is made too small, then the type-2 fuzzy sets become similar to type-1 fuzzy sets. On the other hand, if the width of the uncertain region is too large, then the uncertain mean covers most of input domain. Eq. (3.31) indicates that the initial width is equal to the Euclidean distance between current input data **x** and the center of best matching rule for this data point times an overlapping parameter $\beta$. In this study $\beta$ is set to 0.5, so that the width of new type-2 fuzzy set is half of the Euclidean distance from the center of the best matching rule, and an adequate overlap between adjacent rules is realized.

### *3.3.2 Parameter Learning*

Along with the learning of the structure, the parameters are also learnt. All free parameters of the MRIT2NFS are adjusted with each incoming training data regardless of whether a rule is generated or not. For clarity, let us just consider the q^th output of the network. The parameter learning process updates the network parameters minimizing the error function

$$E = \frac{1}{2}[y'_q(t+1) - y_d(t+1)]^2, \tag{3.32}$$

where $y'_q(t+1)$ and $y_d(t+1)$ represent the MRIT2NFS output and the desired output, respectively. The parameters in the consequent part are learned based on the rule-ordered Kalman filter algorithm [29] as described next. To compute $y'_{lq}$ and $y'_{rq}$ in the KM iterative procedure, the required precondition is that $\tilde{y}_{lq}$ and $\tilde{y}_{rq}$ are rearranged in an ascending order. As the consequent values $\tilde{y}_{lq}$ and $\tilde{y}_{rq}$ change, their rule-ordering may also change. The Eq. (3.20) indicated the arranged consequent values with respect to the original rule order. According to the mapping Eqs. (3.21) and (3.24) are expressed by $\tilde{y}_{lq}$ and $\tilde{y}_{rq}$ as follows [29]:

$$y'_{lq} = \phi_{lq}{}^T \tilde{\mathbf{y}}_{\mathbf{lq}}, \quad \phi_{lq} = \frac{\bar{\boldsymbol{\psi}}_q^T \mathbf{Q}_\mathbf{l}^\mathbf{T} \mathbf{E}_\mathbf{1}^\mathbf{T} \mathbf{E}_\mathbf{1} \mathbf{Q}_\mathbf{l} + \underline{\boldsymbol{\psi}}_q^T \mathbf{Q}_\mathbf{l}^\mathbf{T} \mathbf{E}_\mathbf{2}^\mathbf{T} \mathbf{E}_\mathbf{2} \mathbf{Q}_\mathbf{l}}{\mathbf{p}_\mathbf{l}^\mathbf{T} \mathbf{Q}_\mathbf{l} \bar{\boldsymbol{\psi}}_q + \mathbf{b}_\mathbf{l}^\mathbf{T} \mathbf{Q}_\mathbf{l} \underline{\boldsymbol{\psi}}_q} \in \Re^{M\times 1} \tag{3.33}$$

$$y'_{rq} = \phi_{rq}{}^T \tilde{\mathbf{y}}_{\mathbf{rq}}, \quad \phi_{rq} = \frac{\underline{\boldsymbol{\psi}}_q^T \mathbf{Q}_\mathbf{r}^\mathbf{T} \mathbf{E}_\mathbf{3}^\mathbf{T} \mathbf{E}_\mathbf{3} \mathbf{Q}_\mathbf{r} + \bar{\boldsymbol{\psi}}_q^T \mathbf{Q}_\mathbf{r}^\mathbf{T} \mathbf{E}_\mathbf{4}^\mathbf{T} \mathbf{E}_\mathbf{4} Q_r}{\mathbf{p}_\mathbf{r}^\mathbf{T} \mathbf{Q}_\mathbf{r} \underline{\boldsymbol{\psi}}_q + \mathbf{b}_\mathbf{r}^\mathbf{T} \mathbf{Q}_\mathbf{r} \bar{\boldsymbol{\psi}}_q} \in \Re^{M\times 1} \tag{3.34}$$

Thus, the output $y'_q$ in Eq. (3.27) can be re-expressed as

$$\begin{aligned} y'_q &= \frac{1}{2}(y'_{lq} + y'_{rq}) = \frac{1}{2}(\boldsymbol{\phi}_{lq}{}^T \tilde{\boldsymbol{y}}_{lq} + \boldsymbol{\phi}_{rq}{}^T \tilde{\boldsymbol{y}}_{rq}) = [\bar{\boldsymbol{\phi}}_{lq}{}^T \quad \bar{\boldsymbol{\phi}}_{rq}{}^T] \begin{bmatrix} \tilde{\boldsymbol{y}}_{lq} \\ \tilde{\boldsymbol{y}}_{rq} \end{bmatrix} \\ &= [\bar{\phi}_{lq}^1 \cdots \bar{\phi}_{lq}^M \; \bar{\phi}_{rq}^1 \cdots \bar{\phi}_{rq}^M] \begin{bmatrix} \tilde{y}_{lq}^1 \\ \vdots \\ \tilde{y}_{lq}^M \\ \tilde{y}_{rq}^1 \\ \vdots \\ \tilde{y}_{rq}^M \end{bmatrix} \end{aligned} \tag{3.35}$$

where $\bar{\boldsymbol{\phi}}_{lq}{}^T = 0.5\boldsymbol{\phi}_{lq}{}^T$ and $\bar{\boldsymbol{\phi}}_{rq}^T = 0.5\boldsymbol{\phi}_{rq}{}^T$. According to Eqs. (3.18) and (3.19), Eq. (3.35) can be further expressed as follows [29],

$$y'_q = [\bar{\boldsymbol{\phi}}_{lq}^{\,T} \quad \bar{\boldsymbol{\phi}}_{rq}^{\,T}]\begin{bmatrix}\tilde{\boldsymbol{y}}_{lq}\\ \tilde{\boldsymbol{y}}_{rq}\end{bmatrix} = [\bar{\phi}_{lq}^1 \ \cdots \bar{\phi}_{lq}^M \ \bar{\phi}_{rq}^1 \ \cdots \bar{\phi}_{rq}^M]$$

$$\begin{bmatrix}
\sum_{j=0}^{n_u}\sum_{k=0}^{N_j} c_{jkq}^1 u_j(t-k) + \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} c_{(j+n_u)kq}^1 y_{pj}(t-k) - \sum_{j=0}^{n_u}\sum_{k=0}^{N_j} s_{jkq}^1 |u_j(t-k)| - \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} s_{(j+n_u)kq}^1 |y_{pj}(t-k)| \\
\vdots \\
\sum_{j=0}^{n_u}\sum_{k=0}^{N_j} c_{jkq}^M u_j(t-k) + \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} c_{(j+n_u)kq}^M y_{pj}(t-k) - \sum_{j=0}^{n_u}\sum_{k=0}^{N_j} s_{jkq}^M |u_j(t-k)| - \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} s_{(j+n_u)kq}^M |y_{pj}(t-k)| \\
\sum_{j=0}^{n_u}\sum_{k=0}^{N_j} c_{jkq}^1 u_j(t-k) + \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} c_{(j+n_u)kq}^1 y_{pj}(t-k) + \sum_{j=0}^{n_u}\sum_{k=0}^{N_j} s_{jkq}^1 |u_j(t-k)| + \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} s_{(j+n_u)kq}^1 |y_{pj}(t-k)| \\
\vdots \\
\sum_{j=0}^{n_u}\sum_{k=0}^{N_j} c_{jkq}^M u_j(t-k) + \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} c_{(j+n_u)kq}^M y_{pj}(t-k) + \sum_{j=0}^{n_u}\sum_{k=0}^{N_j} s_{jkq}^M |u_j(t-k)| + \sum_{j=1}^{n_o}\sum_{k=0}^{O_j} s_{(j+n_u)kq}^M |y_{pj}(t-k)|
\end{bmatrix} \tag{3.36}$$

In Eq. (3.36) $c_{jkq}^l$ and $s_{jkq}^l$ are the coefficients of the linear equations involved in the consequents. During the on-line structure learning, the dimension of $\tilde{y}_{lq}$ and $\tilde{y}_{rq}$ increases with time, and the positions of $c_{jkq}$ and $s_{jkq}$ change accordingly within the same vector. For keeping the position of $c_{jkq}$ and $s_{jkq}$ unaltered in the vector, the rule-ordered Kalman filtering algorithm rearranges elements in rule order in Eq. (3.36). Let $\tilde{\mathbf{v}}_{TSK}$ denote the vector all consequent parameters, i.e.,

$$\tilde{\mathbf{v}}_{TSK} = [c_{00q}^1 \cdots c_{(n_o+n_u)N_{n_o}q}^1 \ s_{00q}^1 \cdots s_{(n_o+n_u)N_{n_o}q}^1 \cdots\cdots c_{00q}^M \cdots c_{(n_o+n_u)N_{n_o}q}^M \ s_{00q}^M \cdots s_{(n_o+n_u)N_{n_o}q}^M]^T \tag{3.37}$$

where the consequent parameters are placed according to the rule order. Eq. (3.36) can now be re-expressed as

$$\begin{aligned} y'_q &= [\bar{\phi}_{c1}u_0 \ \ldots \ \bar{\phi}_{c1}y_{pn_o}(t-O_{n_o}) \ -\bar{\phi}_{s1}|u_0| \ \ldots \ -\bar{\phi}_{s1}|y_{pn_o}(t-O_{n_o})| \ \ldots \ \ldots \\ &\quad \bar{\phi}_{cM}u_0 \ \ldots \ \bar{\phi}_{cM}\ y_{pn_o}(t-O_{n_o}) - \bar{\phi}_{sM}|u_0| \ldots \ -\bar{\phi}_{sM}|y_{pn_o}(t-O_{n_o})| \ ]\tilde{\mathbf{v}}_{TSK} \\ &= \bar{\boldsymbol{\phi}}'^{\,T}_{TSK}\tilde{\mathbf{v}}_{TSK} \end{aligned} \tag{3.38}$$

where $\bar{\phi}_{cq}^{j} = \bar{\phi}_{lq}^{j} + \bar{\phi}_{rq}^{j}$ and $\bar{\phi}_{sq}^{j} = \bar{\phi}_{rq}^{j} - \bar{\phi}_{lq}^{j}$, $j = 1, \ldots, M$. The consequent parameter vector $\tilde{\mathbf{v}}_{TSK}$ is updated by executing the following rule-ordered Kalman filtering algorithm [29]

$$\begin{aligned}
&\tilde{\mathbf{v}}_{TSK}(\mathbf{t+1}) = \tilde{\mathbf{v}}_{TSK}(\mathbf{t}) + \mathbf{S}(\mathbf{t+1})\bar{\boldsymbol{\phi}}'_{TSK}(\mathbf{t+1})(y^{d}(t+1) - \bar{\boldsymbol{\phi}}'^{\,T}_{TSK}(\mathbf{t+1})\tilde{\mathbf{v}}_{TSK}(\mathbf{t})) \\
&\mathbf{S}(\mathbf{t+1}) = \frac{1}{\kappa}[\mathbf{S}(\mathbf{t}) - \frac{\mathbf{S}(\mathbf{t})\bar{\boldsymbol{\phi}}'_{TSK}(\mathbf{t+1})\bar{\boldsymbol{\phi}}'^{\,T}_{TSK}(\mathbf{t+1})\mathbf{S}(\mathbf{t})}{\kappa + \bar{\boldsymbol{\phi}}'^{\,T}_{TSK}(\mathbf{t+1})\mathbf{S}(\mathbf{t})\bar{\boldsymbol{\phi}}'_{TSK}}]
\end{aligned} \tag{3.39}$$

where $0 < \kappa \le 1$ is a forgetting factor. (Just to keep parity with a previous study [29], we have used $\kappa$=0.99995; however, our experience suggests that one can use $\kappa$=1.0 without much effect, as expected, on the overall performance.) The dimension of $\tilde{\mathbf{v}}_{TSK}$ and $\bar{\boldsymbol{\phi}}'_{TSK}$, and the matrix $S$ increases when a new rule is generated. When a new rule evolves, MRIT2NFS augments $\mathbf{S}(\mathbf{t})$ as follows

$$\mathbf{S}(\mathbf{t}) = block\ diag[\mathbf{S}(\mathbf{t})\ \ C \cdot I] \in \Re^{2(M+1)(\sum_{j=0}^{n_u}(N_j+1)+\sum_{j=1}^{n_o}(O_j+1))\times 2(M+1)(\sum_{j=0}^{n_u}(N_j+1)+\sum_{j=1}^{n_o}(O_j+1))} \tag{3.40}$$

where $C$ is a large positive constant (we use C=10) and the size of the identity matrix $\mathbf{I}$ is

$$2(\sum_{j=0}^{n_u}(N_j+1) + \sum_{j=1}^{n_o}(O_j+1)) \times 2(\sum_{j=0}^{n_u}(N_j+1) + \sum_{j=1}^{n_o}(O_j+1)) \tag{3.41}$$

Note that S(0) is 1x1 matrix. We used S(0)=[10]. The antecedent parameters of MRIT2NFS are tuned by the gradient descent algorithm. For convenience of notations for the gradient descent learning rules, according to [59], Eq. (3.21) can be re-written as

$$y'_{lq} = \frac{\bar{\boldsymbol{\psi}}_q^T \boldsymbol{a}_{lq} + \underline{\boldsymbol{\psi}}_q^T \boldsymbol{b}_{lq}}{\bar{\boldsymbol{\psi}}_q^T \boldsymbol{c}_{lq} + \underline{\boldsymbol{\psi}}_q^T \boldsymbol{d}_{lq}} \tag{3.42}$$

where

$$\mathbf{a_{lq}} = \mathbf{Q_l^T E_1^T E_1 Q_l \tilde{y}_{lq}} \in \Re^{M\times 1},\ \ \mathbf{b_{lq}} = \mathbf{Q_l^T E_2^T E_2 Q_l \tilde{y}_{lq}} \in \Re^{M\times 1} \tag{3.43}$$

$$\mathbf{c_{lq}} = \mathbf{Q_l^T p_l} \in \Re^{M\times 1},\quad \mathbf{d_{lq}} = \mathbf{Q_l^T b_l} \in \Re^{M\times 1} \quad . \tag{3.44}$$

Similarly, Eq. (3.24) can be re-written as

$$y'_{rq} = \frac{\underline{\boldsymbol{\psi}}_q^T \boldsymbol{a}_{rq} + \bar{\boldsymbol{\psi}}_q^T \boldsymbol{b}_{rq}}{\underline{\boldsymbol{\psi}}_q^T \boldsymbol{c}_{rq} + \bar{\boldsymbol{\psi}}_q^T \boldsymbol{d}_{rq}} \tag{3.45}$$

where

$$\mathbf{a_{rq}} = \mathbf{Q_r^T E_3^T E_3 Q_r \tilde{y}_{rq}} \in \Re^{M\times 1},\ \ \mathbf{b_{rq}} = \mathbf{Q_r^T E_4^T E_4 Q_r \tilde{y}_{rq}} \in \Re^{M\times 1} \tag{3.46}$$

$$\mathbf{c_{rq}} = \mathbf{Q_r^T p_r} \in \Re^{M\times 1},\quad \mathbf{d_{rq}} = \mathbf{Q_r^T b_r} \in \Re^{M\times 1} \tag{3.47}$$

Using gradient descent algorithm, we have

$$\lambda_{ik}^{q}(t+1) = \lambda_{ik}^{q}(t) - \eta \frac{\partial E}{\partial \lambda_{ik}^{q}(t)} \tag{3.48}$$

where $\eta$ is a learning constant ( $\eta = 0.075$ in this paper) and

$$\frac{\partial E}{\partial \lambda_{ik}^{q}} = \frac{\partial E}{\partial y_q'}\left(\frac{\partial y_q'}{\partial y_{lq}'}\frac{\partial y_{lq}'}{\partial \lambda_{ik}^{q}} + \frac{\partial y_q'}{\partial y_{rq}'}\frac{\partial y_{rq}'}{\partial \lambda_{ik}^{q}}\right) = \frac{1}{2}(y_q' - y_d)\left[\left(\frac{\partial y_{lq}'}{\partial \bar{\psi}_i^q} + \frac{\partial y_{rq}'}{\partial \bar{\psi}_i^q}\right)\frac{\partial \bar{\psi}_i^q}{\partial \lambda_{ik}^{q}} + \left(\frac{\partial y_{lq}'}{\partial \underline{\psi}_i^q} + \frac{\partial y_{rq}'}{\partial \underline{\psi}_i^q}\right)\frac{\partial \underline{\psi}_i^q}{\partial \lambda_{ik}^{q}}\right] \tag{2.49}$$

Where

$$\frac{\partial y_{lq}'}{\partial \bar{\psi}_i^q} = \frac{a_{lqi} - y_{lq}' c_{lqi}}{\bar{\boldsymbol{\psi}}_q^T \boldsymbol{c}_{lq} + \underline{\boldsymbol{\psi}}_q^T \boldsymbol{d}_{lq}},\quad \frac{\partial y_{rq}'}{\partial \bar{\psi}_i^q} = \frac{b_{rqi} - y_{rq}' d_{rqi}}{\underline{\boldsymbol{\psi}}_q^T \boldsymbol{c}_{rq} + \bar{\boldsymbol{\psi}}_q^T \boldsymbol{d}_{rq}} \tag{3.50}$$

$$\frac{\partial y_{lq}'}{\partial \underline{\psi}_i^q} = \frac{b_{lqi} - y_{lq}' d_{lqi}}{\bar{\boldsymbol{\psi}}_q^T \boldsymbol{c}_{lq} + \underline{\boldsymbol{\psi}}_q^T \boldsymbol{d}_{lq}},\quad \frac{\partial y_{rq}'}{\partial \underline{\psi}_i^q} = \frac{a_{rqi} - y_{rq}' c_{rqi}}{\underline{\boldsymbol{\psi}}_q^T \boldsymbol{c}_{rq} + \bar{\boldsymbol{\psi}}_q^T \boldsymbol{d}_{rq}} \tag{3.51}$$

$$\frac{\partial \bar{\psi}_i^q}{\partial \lambda_{ik}^{q}} = \bar{\psi}_k^q(t-1) - \bar{f}^i(t),\quad \frac{\partial \underline{\psi}_i^q}{\partial \lambda_{ik}^{q}} = \underline{\psi}_k^q(t-1) - \underline{f}^i(t) \tag{3.52}$$

Details of the learning equations for parameters, including $m_{j1}$ , $m_{j2}$ ,and $\sigma$ , of the antecedents can be found in [59].

*Pruning of less-Important Rules*: Our system involves many recurrent weights. Depending on the nature of the underlying system that we are trying to model, all of these recurrent feedbacks may not be important. In fact, if the magnitude of a recurrent weight is very low, then that weight will not have much effect on the system output and hence such weights / feedbacks can be dropped. This is what we do. If the absolute value of a recurrent weight is less than a pre-defined threshold ε, we delete that connection. Once we delete some recurrent weights (feedback connections), we must adapt the system in its new environment and hence, we retrain the network a few epochs (here we use only five epochs). The MRIT2NFS that use this type of recurrent weight elimination approach is called MRIT2NFS-$\varepsilon$. It can be kept similar performance and effectively reduces less-useful recurrent weights.

## 3.4 Simulation Results

This section describes application of MRIT2NFS on five problems. These examples include identification of two single-input-single-output (SISO) dynamic systems (Examples 1–2), one multi-input-multi-output (MIMO) dynamic system (Example 3), prediction of chaotic time series (Example 4), and identification of nonlinear system plant (Example 5). For all but Example 5, we normalize the data sets in [-1, 1]. Example 5 is not a dynamical system, and it is comparatively easy to learn. So we did not normalize the data, but we have used the same membership definition as in Eq. (3.28). We shall see later that even in this case the performance of the system is very satisfactory indicating the robustness of our system. The performance of MRIT2NFS is compared with that of recurrent and feedback type-1 and type-2 FNNs.

### *3.4.1 Example 1 (SISO Dynamic System Identification).*

This example uses MRIT2NFS to identify an SISO linear time-varying system, which was introduced in [7]. The dynamic system with lagged inputs is guided by the following difference equation:

$$y_p(t+1) = f(y_p(t), y_p(t-1), y_p(t-2), u(t), u(t-1)) \tag{3.53}$$

where

$$f(x_1, x_2, x_3, x_4, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) x_4}{1 + x_2^2 + x_3^2} \tag{3.54}$$

The system has a single input (i.e., $n_u$=1) and a single output (i.e., $n_o$=1). The current variables $u(t)$ and $y_p(t)$ are fed as inputs to the MRIT2NFS input layer. The current output of the plant depends on two previous outputs and one previous input. Therefore, the consequent part parameters of MRIT2NFS are set as $N_1 = 2$ and $O_1$=1. The training procedure minimizes the square error between the output of the system $y_p(t+1)$ and the target output $y_d(t+1)$. To train the MRIT2NFS, we follow the same computational protocols as in [7], i.e., we use only ten epochs and there are 900 time steps in each epoch. In each epoch, the first 350 inputs are random values uniformly distributed over [-2, 2] and the remaining 550 training inputs are generated from a sinusoid defined by $1.05\sin(\pi t/45)$. This type of training is analogous to an online training process, where the total number of online training time steps is 9000. The structure learning threshold $f_{th}$ influences the number of fuzzy rules to be generated. After training, two recurrent fuzzy rules are generated when $f_{th}$ is set to 0.02. Table 3.1 shows the root–mean-squared error (RMSE) on the training data. To validate the identified system, as adopted in [9] and we use the following input:

$$u(t)=\begin{cases}\sin(\frac{\pi t}{25}), & t<250\\ 1.0, & 250\le t<500\\ -1.0, & 500\le t<750\\ 0.3\sin(\frac{\pi t}{25})+0.1\sin(\frac{\pi t}{32}) & \\ \quad +0.6\sin(\frac{\pi t}{10}), & 750\le t<1000\end{cases} \tag{3.55}$$

Fig. 3.6 compares the actual output with the output produced by MRIT2NFS for the test input generated using Eq. (3.55), while Fig. 3.7 shows the test error between the desired output and actual output produced by MRIT2NFS. Fig. 3.6 and Fig. 3.7 together reveal a very good match suggesting that MRIT2NFS architecture along with our system identification scheme does a very good job of identifying the dynamical system with feedback.

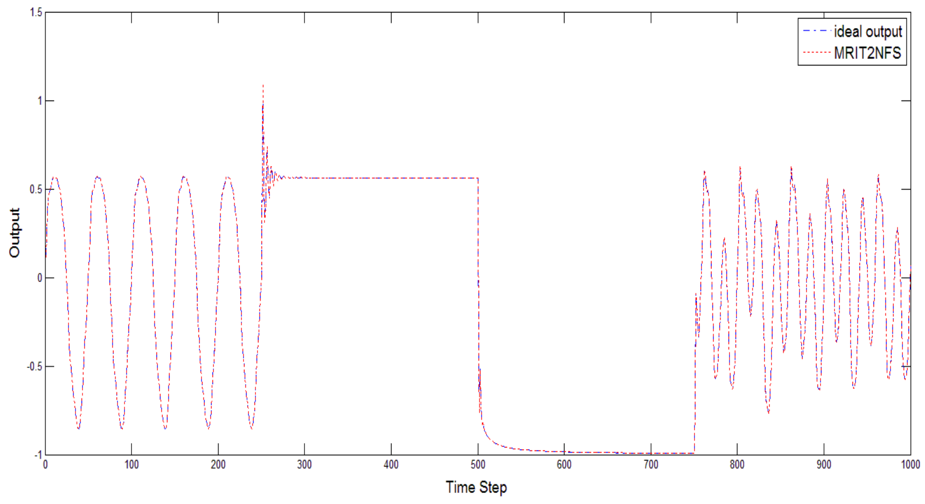

Fig. 3.6. Outputs of the dynamic plant (dashed-dotted line) and MRIT2NFS (dotted line) in Example 1.

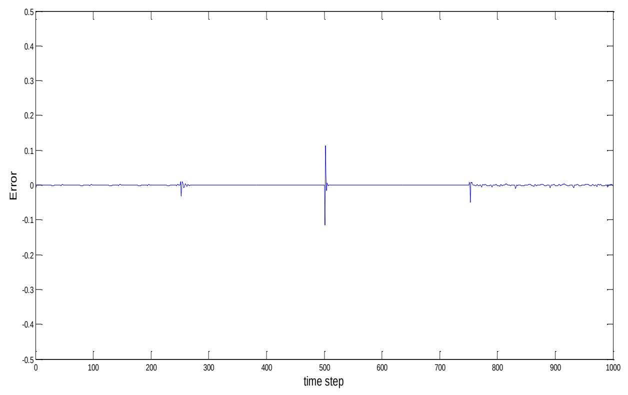

Fig. 3.7. Test errors between the MRIT2NFS and actual plant outputs.

Table 3.1 compares the performance of MRIT2NFS with that of seven different approaches including TSK-type feed-forward type-1 and type-2 FNNs, a recurrent NN, and type-1 recurrent FNNs. The comparison is done in terms of number of rules, number of free parameters, training RMSE, and test RMSE. As in MRIT2NFS, all these networks use the same information including number of input variables, training data, test data, and training epochs. In order to make a fair comparison, the total number of parameters of the feed-forward type-1 FNN is kept similar to that of feed-forward interval type-2 FNN. The number of parameters in an interval type-2 FNN is larger than that in a feed-forward type-1 FNN because of extra free parameters in type-2 fuzzy sets and rule consequent part. Consequently, the number of rules used in a feed-forward type-1 FNN is larger than that in a feed-forward interval type-2 FNN, as shown in Table 3.1.

TABLE 3.1. PERFORMANCE OF MRIT2NFS AND OTHER RECURRENT MODELS FOR SISO PLANT IDENTIFICATION IN EXAMPLE 1.

| Models | Number of Rules | Number of Parameters | Training RMSE | Test RMSE |
|---|---|---|---|---|
| Feedforward Type-1 FNN | 7 | 42 | 0.0178 | 0.0528 |
| WRFNN [11] | 5 | 55 | 0.064 | 0.098 |
| TRFN-S [7] | 5 | 60 | 0.021 | 0.041 |
| RSEFNN [14] | 4 | 32 | 0.02 | 0.0397 |
| Feedforward Type-2 FNN | 4 | 48 | 0.032 | 0.052 |
| RSEIT2FNN -UM [29] | 2 | 38 | 0.0048 | 0.049 |
| RIFNN [69] | 4 | 36 | 0.023 | 0.0465 |
| MRIT2NFS | 2 | 40 | **0.0008** | **0.0078** |

The compared feed-forward type-2 FNN is an interval type-2 FNN with uncertain means, where all network parameters are learned by gradient descent algorithm. Our result reveals the advantage of using recurrent structure in the MRIT2NFS, which achieves smaller test RMSE than that by the feed-forward type-2 FNN. All of the compared recurrent type-2 FNNs use the same fuzzy sets in the antecedent part, i.e., interval type-2 fuzzy sets with uncertain means. The performance of MRIT2NFS is also compared with other interval type-2 FNNs with recurrent structure, including a recurrent self-evolving interval type-2 fuzzy neural network with uncertain means (RSEIT2FNN-UM) [29] and a recurrent interval-valued fuzzy neural network (RIFNN) [69]. In these cases also, MRIT2NFS yields a better test accuracies.

However, RIFNN and RSEIT2FNN-UM use a marginally lower number of free parameters. Our results demonstrate that MRIT2NFS can effectively capture information about the system using mutual feedbacks and outperforms the RSEIT2FNN, which only uses local feedbacks. The recurrent type-1 FNNs considered include, the wavelet-based RFNN (WRFNN) [11], TSK-type recurrent fuzzy network with supervised learning (TRFN-S) [7], and recurrent self-evolving fuzzy neural network with local feedback (RSEFNN-LF) [14]. Table 3.1 indicates that the test error of the TRFN and the RIFNN are very close for the noise-free environment, but for noisy environment the TRFN-S is found to perform better. We have also compared the performance between TRFN and RSEIT2FNN. The recurrent structure with only local feedbacks in RSEIT2FNN may not be adequate for this example because the rules lack the information from other rules. For this reason, the performance of the TRFN is found to be better than that of the RSEIT2FNN (Table 3.1).

TABLE 3.2 INFLUENCE OF $f_{th}$ and $\varepsilon$ ON THE PERFORMANCE OF MRIT2NFS WITH $\beta = 0.5$

| Models | $f_{th}$ | Number of Rules | | Number of Parameters | Training RMSE | Test RMSE |
|---|---|---|---|---|---|---|
| MRIT2NFS | 0.02 | 2 | | 40 | 0.0008 | 0.0078 |
| | 0.15 | 3 | | 63 | 0.0007 | 0.0075 |
| | 0.25 | 4 | | 88 | 0.0009 | 0.0060 |
| MRIT2NFS-$\varepsilon$ | $f_{th}$ | Number of Rules | $\varepsilon$ | Number of Parameters | Training RMSE | Test RMSE |
| | 0.02 | 2 | 0.4 | 40 | 0.0008 | **0.0071** |
| | | | 0.6 | 40 | 0.00077 | 0.0076 |
| | | | 0.8 | 38 | 0.0009 | 0.0088 |
| | 0.15 | 3 | 0.4 | 63 | 0.0005 | **0.00772** |
| | | | 0.6 | 61 | 0.00068 | 0.00768 |
| | | | 0.8 | 58 | 0.00075 | 0.0078 |
| | 0.25 | 4 | 0.4 | 87 | 0.0006 | 0.0055 |
| | | | 0.6 | 86 | 0.001 | **0.0040** |
| | | | 0.8 | 83 | 0.0007 | 0.0066 |

Like any other Type-2 method, the proposed type-2 methods demand more computation but it can yield more quality outputs. Moreover, although each learning step is computationally more expensive compared to that of its Type-1 counterpart, our network, using the same number of iterations, yields a better solution (faster convergence) than the Type-1 systems.

There are a few parameters, $f_{th}, \beta,$ and $\varepsilon$, that are involved in the learning of MRIT2NFS. We now investigate the influence of these parameters on the performance of MRIT2NFS. In addition we shall also consider the robustness of the system against noise in the inputs. The threshold parameter, $f_{th}$, decides the number of rules in the MRIT2NFS while the parameter $\varepsilon$ in MRIT2NFS–ε is used to decide the feedback connections in layer 4 that could be removed. Table 3.2 shows the MRIT2NFS performance for different values of $f_{th}$ and $\varepsilon$ when $\beta = 0.5$.

As expected, larger values of $f_{th}$ result in larger numbers of rules and larger ε reduces the number of tunable parameters in the system. Table 3.2 suggests that different choices of $f_{th}$ and $\varepsilon$ although change the number of rules marginally, the training and test errors practically do not change. Thus at least for this data set, our system is quite robust with respect to the choice of these two parameters. From a user point of view, the network with the smallest number of free parameters that can provide the desired level of performance should be the preferred network. Because with a larger degrees of freedom, the chances of having more local optima and getting stuck to one of them would usually be higher. Next, we investigate the effect of $\beta$ on the performance of MRIT2NFS for a constant value of $f_{th}$. A small value of $\beta$ generates larger numbers of rules because of the smaller width of the initial type-2 fuzzy sets.

Table 3.3 shows the performance of MRIT2NFS for different values of $\beta$ when $f_{th}$=0.02. From Table 3.3 we observe that the network performance (both training and test error) is not sensitive to variations in $\beta$ when $f_{th}$=0.02, although the number of rules decreases as $\beta$ increases. It is interesting to note that as $\beta$ increases from 0.2 to 0.7, the number of rules decreases from 5 to 2 without affecting the performance of the system. In fact, with 2 rules, both the training and test performances are slightly improved over the case with 5 rules. However, for a given problem, if the goal is to find the optimal parameters, we can use a two-level cross validation mechanism.

TABLE 3.3 INFLUENCE OF $\beta$ ON THE PERFORMANCE OF MRIT2NFS WITH $f_{th} = 0.02$

| $\beta$ | 0.2 | 0.3 | 0.5 | 0.6 | 0.7 |
|---|---|---|---|---|---|
| Number of Rules | 5 | 4 | 2 | 2 | 2 |
| Training RMSE | 0.0016 | 0.0009 | 0.0008 | **0.00076** | 0.00078 |
| Test RMSE | 0.0098 | 0.0091 | **0.0078** | 0.008 | 0.0087 |

Next we assess how robust the network is with respect to measurement noise in the plant output. Since the plant output $y_p$ is fed back as an input to the network, a noise in the measurement of the plant output $y_p$ is likely to have an effect on the performance of the system. The experiment also uses the control input sequence in Eq. (3.55). We consider three levels of Gaussian noise with standard deviations (STDs): 0.1, 0.3, and 0.5. We use 30 simulations for the statistical analysis. Table 3.4 shows the performance of MRIT2NFS for the three different noisy environments. For the purpose of comparison, we also use the same noisy environment to assess the noise tolerance of the other networks, including feed-forward type-1 and type-2 FNN, TRFN [7], RSEIT2FNN [29], and RIFNN [69]. The results in Table

3.4 indicate that the feed-forward type-2 FNN can deal with noise much better than the feed-forward type-1 FNN. The consequent part of the TRFN is a function of current input variables. The consequent part in the RSEIT2FNN and the MRIT2NFS is of Takagi-Sugeno-Kang (TSK)-type that consists of system output $y_p$ and its previous values. As a result of this, the rate of increase in the RMSE with increase in σ for MRIT2NFS may be marginally higher than that for other networks which do not use feedback of system outputs. Finally, The MRIT2NFS achieves better performance than the other compared FNNs for noise-free and noisy cases.

TABLE 3.4 PERFORMANCE OF MRIT2NFS AND OTHER FEEDFORWARD AND RECURRENT MODELS WITH DIFFERENT NOISE LEVEL IN EXAMPLE 1

| Models | | Feedforward Type-1 FNN | TRFN-S [7] | Feedforward Type-2 FNN | RSEIT2FNN -UM[29] | RIFNN [69] | MRIT2NFS |
|---|---|---|---|---|---|---|---|
| Number of Rules | | 7 | 5 | 4 | 2 | 4 | 2 |
| Number of Parameters | | 42 | 60 | 48 | 38 | 36 | 40 |
| Test RMSE | STD=0.1 | 0.121 | 0.062 | 0.056 | 0.168 | 0.057 | **0.021** |
| | STD=0.3 | 0.312 | 0.133 | 0.242 | 0.522 | 0.192 | **0.05** |
| | STD=0.5 | 0.458 | 0.162 | 0.361 | 0.783 | 0.315 | **0.098** |

## 3.4.2 *Example 2 (SISO Dynamic System Identification)*

We now consider the following dynamic system with longer input delays:

$$y_p(t+1) = 0.72y_p(t) + 0.025y_p(t-1)u_1(t-1) + 0.01u_1^2(t-2) + 0.2u_1(t-3) \qquad (3.56)$$

This plant is the same as the one used in [7]. This system has a single input ($n_u$=1) and a single output ($n_o$=1). Thus, the current values of $u(t)$ and $y_p(t)$ are fed as inputs to the MRIT2NFS input layer. The current output of the plant depends on one previous output and three previous inputs. Therefore, for MRIT2NFS $N_1$=3 and $O_1$=1. The training data and time steps are the same as those used in Example 1. In MRIT2NFS training, the structure learning threshold is set to 0.02. After 90 epochs of training, two rules are generated. To test the identified system, the test signal used in Example 1 is also adopted here. Fig. 3.8 shows the outputs of the plant and those of the MRIT2NFS for these test inputs. Fig. 3.9 shows the test error between the outputs of MRIT2NFS and of the plant. Table 3.5 shows the structure, and training and test RMSEs of MRIT2NFS. The performance of MRIT2NFS-$\varepsilon$ ($\varepsilon = 0.6$ is used in this paper) with the same network size is also shown in Table 3.6. Like Example 1, Table 3.6 shows that MRIT2NFS and MRIT2NFS-$\varepsilon$ have similar performance.

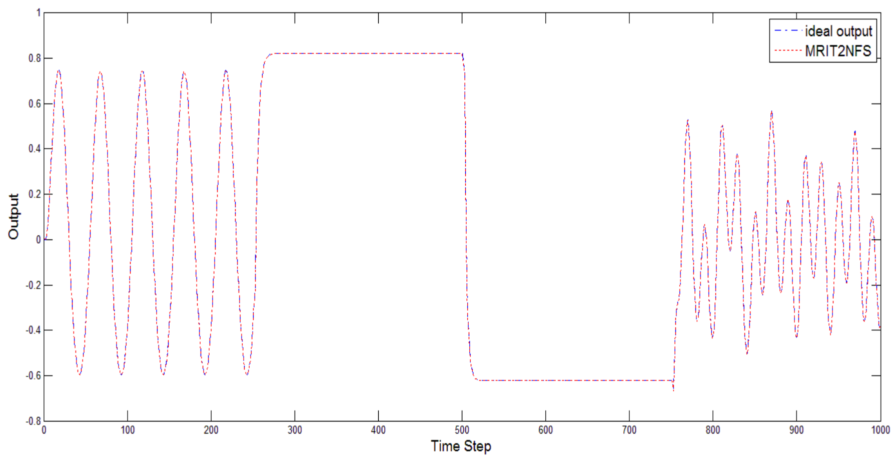

Fig. 3.8. Outputs of the dynamic plant (dashed-dotted line) and MRIT2NFS (dotted line) in Example 2.

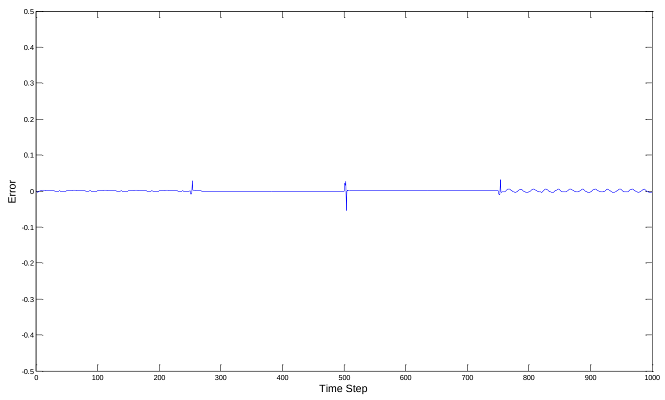

Fig. 3.9. Test errors between the MRIT2NFS and actual plant outputs.

The performance of MRIT2NFS is compared with that of feed-forward type-1 and typ-2 FNNs and recurrent type-1 FNNs. These models also use the same number of training epochs and training and test data as used for the MRIT2NFS. Table 3.5 also depicts the number of rules and parameters, and the training and test RMSEs of these compared networks. These results show that the MRIT2NFS achieves better performance than that of other networks. In Table 3.6 we investigate the effect of different choices $f_{th}$ *and* $\varepsilon$ on the performance of MRIT2NFS when $\beta = 0.5$.

TABLE 3.5 PERFORMANCE OF MRIT2NFS AND OTHER RECURRENT MODELS FOR SISO PLANT IDENTIFICATION IN EXAMPLE 2.

| Models | Number of Rules | Number of Parameters | Training RMSE | Test RMSE |
|---|---|---|---|---|
| Feedforward Type-1 FNN | 7 | 49 | 0.007 | 0.032 |
| WRFNN [11] | 5 | 55 | 0.0574 | 0.083 |
| TRFN-S [7] | 5 | 60 | 0.0076 | 0.0286 |
| RSEFNN [14] | 4 | 30 | 0.0156 | 0.0279 |
| Feedforward Type-2 FNN | 4 | 48 | 0.0155 | 0.038 |
| RSEIT2FNN -UM [29] | 2 | 45 | 0.0034 | 0.006 |
| RIFNN [69] | 4 | 36 | 0.0125 | 0.0288 |
| MRIT2NFS | 2 | 44 | **0.0013** | **0.0028** |

As done for Example 1, here also using the same computational protocols we study the robustness of MRIT2NFS in noisy environments. Table 3.7 summarizes the results for the MRIT2NFS with different noise levels (Gaussian noise with STD of 0.1, 0.3, and 0.5, and with 30 Monte Carlo realizations for each case). As revealed by Table 3.7, for noisy environments, the MRIT2NFS achieves smaller RMSE than the other compared FNNs except the RIFNN. Note that, for noise-free data for the same problem, Table 3.5 reveals that

MRIT2NFS performs better than RIFNN, but with noisy data RIFNN performs better. A possible reason for this may be that the consequent part in the RIFNN is a constant interval-valued set and is not a function of the system output, $y_p$, that are noisy. But for MRIT2NFS the rule consequents are functions of current output $y_p$, which are noisy. Therefore, the impact of noise on RIFNN is much weaker than that on MRIT2NFS. These results show that MRIT2NFS and MRIT2NFS-$\varepsilon$ have similar performance. Table 3.5 reveals that the test error for the MRIT2NFS is smaller than that of the feed-forward type-1 and type-2 FNNs and recurrent type-1and type-2 FNNs.

TABLE 3.6 INFLUENCE OF $f_{th}$ and $\varepsilon$ ON THE PERFORMANCE OF AN MRIT2NFS WITH $\beta = 0.5$

| Models | $f_{th}$ | Number of Rules | | Number of Parameters | Training RMSE | Test RMSE |
|---|---|---|---|---|---|---|
| MRIT2NFS | 0.02 | 2 | | 44 | 0.0013 | 0.0028 |
| | 0.15 | 3 | | 69 | 0.0005 | 0.00274 |
| | 0.25 | 4 | | 96 | 0.0008 | 0.0032 |
| MRIT2NFS-$\varepsilon$ | $f_{th}$ | Rules | $\varepsilon$ | Number of Parameters | Training RMSE | Test RMSE |
| | 0.02 | 2 | 0.4 | 44 | 0.00173 | **0.0025** |
| | | | 0.6 | 42 | 0.0011 | 0.0031 |
| | | | 0.8 | 42 | 0.0014 | 0.0032 |
| | 0.15 | 3 | 0.4 | 69 | 0.00051 | **0.00287** |
| | | | 0.6 | 69 | 0.00052 | 0.00366 |
| | | | 0.8 | 64 | 0.00059 | 0.0041 |
| | 0.25 | 4 | 0.4 | 96 | 0.0007 | **0.0042** |
| | | | 0.6 | 94 | 0.0065 | 0.0048 |
| | | | 0.8 | 87 | 0.0009 | 0.0051 |

TABLE 3.7 PERFORMANCE OF MRIT2NFS AND OTHER FEEDFORWARD AND RECURRENT MODELS WITH DIFFERENT NOISE LEVEL IN EXAMPLE 2

| Models | | Feedforward Type-1 FNN | TRFN-S | Feedforward Type-2 FNN | RIFNN | RSEIT2FNN -UM | MRIT2NFS |
|---|---|---|---|---|---|---|---|
| Number of Rules | | 7 | 5 | 4 | 4 | 2 | 2 |
| Number of Parameters | | 49 | 60 | 48 | 36 | 42 | 44 |
| Test RMSE | STD=0.1 | 0.145 | 0.097 | 0.087 | – | 0.072 | **0.056** |
| | STD=0.3 | 0.309 | 0.276 | 0.246 | **0.15** | 0.217 | 0.202 |
| | STD=0.5 | 0.501 | 0.453 | 0.416 | **0.23** | 0.358 | 0.327 |

## *3.4.3 Example 3 (Chaotic Series prediction)*

In this example, which is introduced in [71], we use MRIT2NFS to predict the chaotic behavior of a dynamic system with one delay and two sensitive parameters that are generated by the following equation:

$$y_p(t+1) = -P \cdot y_p{}^2(t) + Q \cdot y_p(t-1) + 1.0 \qquad (3.57)$$

Eq. (3.57), with P=1.4 and Q=0.3, produces a chaotic attractor. The system has no control input (i.e., $n_u$=0) and a single output (i.e., $n_o$=1) so that only output variable $y_p(t)$ is fed as input to the MRIT2NFS. It is a second order system with one delay, therefore, $O_1$= 1. The training procedure uses the plant output $y_p(t+1)$ as the desired output $y_d(t+1)$. Starting from the initial state $[y_p(1), y_p(0)]$= [0.4, 0.4], two thousand patterns are generated of which the first 1000 patterns are used for training and the remaining 1000 patterns are used for testing. Here also the structure learning threshold $f_{th}$ is set to 0.2 and number of rules generated is 5 after 90 epochs of training.

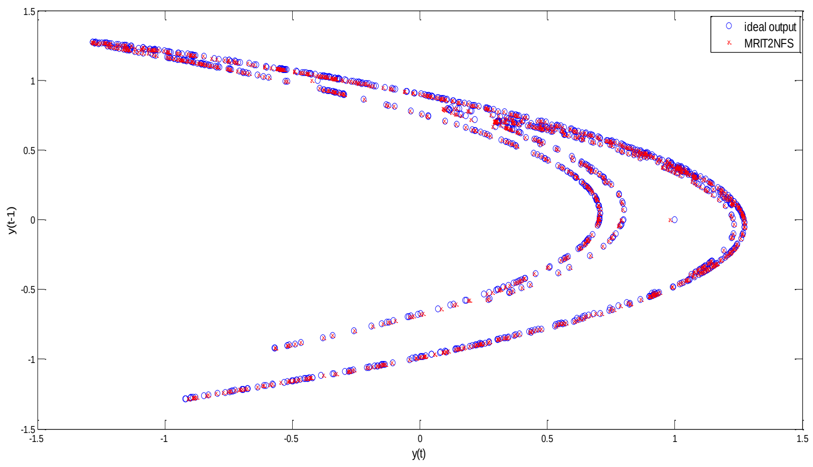

Fig. 3.10 Results of the phase plot for the chaotic system (O) and MRIT2NFS (X).

Fig. 3.10 displays the phase plot of the actual and MRIT2NFS predicted results for the test patterns. Table 3.8 includes the network size, and training and test RMSEs of MRIT2NFS. The performance of MRIT2NFS-$\varepsilon$ is also depicted in Table 3.8. Like the other two examples, the performance of MRIT2NFS and MRIT2NFS-$\varepsilon$ are quite similar. The performance of MRIT2NFS is also compared with that of feed-forward type-1 and type-2 FNNs, and recurrent type-1 FNNs, including WRFNN [11], TRFN-S [7], and RSEFNN [14]. From Table 3.8 we find that MRIT2NFS-$\varepsilon$ exhibits the best performance using almost the minimum number free parameters, while MRIT2NFS achieves the next best performance although it uses a few more free parameters.

In this case, we study the noise tolerance of our system with three levels of Gaussian noise with standard deviations of 0.3, 0.5, and 0.7. Based on 30 Monte Carlo realizations, in Table 3.9, we summarize the test RMSEs of the feed-forward type-1 and type-2 FNNs, recurrent type-1 FNNs, and MRIT2NFS. Table 3.8 shows that the test error of the MRIT2NFS is much smaller than that of the RIFNN for noise-free data. However, Table 3.9 depicts that the test error of the MRIT2NFS for noisy environment is very close to that of the RIFNN. The possible reason for this is the same as explained in Example 2.These results also reveal that the test error of MRIT2NFS is smaller than those of the compared networks for all of the three noise levels.

TABLE 3.8 PERFORMANCE OF MRIT2NFS AND OTHER FEEDFORWARD AND RECURRENT MODELS IN EXAMPLE 3

| Models | Number of Rules | Number of Parameters | Training RMSE | Test RMSE |
|---|---|---|---|---|
| Feedforward Type-1 FNN | 15 | 60 | 0.234 | 0.240 |
| WRFNN [11] | 7 | 70 | 0.191 | 0.188 |
| TRFN-S [7] | 6 | 66 | 0.0296 | 0.0245 |
| RSEFNN [14] | 7 | 35 | 0.032 | 0.021 |
| Feedforward Type-2 FNN | 8 | 56 | 0.201 | 0.200 |
| RSEIT2FNN -UM [29] | 6 | 60 | 0.0043 | 0.0047 |
| RIFNN [69] | 9 | 54 | 0.073 | 0.051 |
| MRIT2NFS -$\varepsilon$(0.6) | 5 | 57 | **0.0028** | **0.0023** |
| MRIT2NFS | 5 | 70 | 0.0041 | 0.0037 |

TABLE 3.9 PERFORMANCE OF MRIT2NFS AND OTHER FEEDFORWARD AND RECURRENT MODELS WITH DIFFERENT NOISE LEVEL IN EXAMPLE 3

| Models | | Feedforward Type-1 FNN | TRFN-S | Feedforward Type-2 FNN | RIFNN | RSEIT2FNN -UM | MRIT2NFS |
|---|---|---|---|---|---|---|---|
| Number of Rules | | 15 | 5 | 4 | 9 | 6 | 5 |
| Number of Parameters | | 60 | 60 | 48 | 54 | 60 | 70 |
| Test RMSE | STD=0.3 | 0.621 | 0.604 | 0.617 | 0.575 | 0.528 | **0.513** |
| | STD=0.5 | 1.114 | 0.955 | 0.917 | **0.725** | 0.803 | 0.814 |
| | STD=0.7 | 1.706 | 1.256 | 1.180 | 1.040 | **1.020** | 1.032 |

### *3.4.4 Example 4 (MIMO Dynamic System Identification)*

In this example, we consider the plant described by the following equation

$$y_{p1}(t+1) = 0.5\cdot[\frac{y_{p1}(t)}{1+y_{p2}^2(t)} + u_1(t-1)] \tag{3.58}$$

$$y_{p2}(t+1) = 0.5\cdot[\frac{y_{p1}(t)y_{p2}(t)}{1+y_{p2}^2(t)} + u_2(t-1)] \tag{3.59}$$

This MIMO dynamic system was also studied in [7]. This plant has two inputs ($n_u$=2) and two outputs ($n_o$=2). So four current input-output values $u_1(t)$, $u_2(t)$, $y_{p1}(t)$, and $y_{p2}(t)$ are fed to the input layer of the network. The present output of the plant depends on control inputs with one time-step delay and current plant states. Therefore, the lag numbers $N_1$, $N_2$, $O_1$ and $O_2$ in MRIT2NFS are set to 1, 1, 0, and 0, respectively. The desired outputs for MRIT2NFS training are $y_{p1}(t+1)$ and $y_{p2}(t+1)$. The MRIT2NFS is trained in an online manner from time step $t=1$ to $t$=11000. The two control inputs $u_1(t)$ and $u_2(t)$ are independent and identically distributed (i.i.d) uniform random sequences over [$-1.4$, 1.4] for $t$ =1 to $t$ =4000. For the remaining 7000 times steps sinusoid signals generated by $\sin(\pi t/45)$ are used for both $u_1(t)$ and $u_2(t)$. The learning coefficient $\eta$ and the threshold $f_{th}$ are set to 0.075 and 0.05, respectively. Based on a compromise between network size and performance, the threshold value 0.05 is used, as discussed in Example 1. For this data set, the training results in three rules. Table 3.10 shows the structure and RMSE of MRIT2NFS. To evaluate the effectiveness of the identified network, we use the following two control input sequences:

$$u_1(t) = u_2(t) = \begin{cases} \sin(\pi t/25), & 1001 \le t < 1250 \\ 1.0, & 1250 \le t < 1500 \\ -1.0, & 1500 \le t < 1750 \\ 0.3\sin(\pi t/25) + 0.1\sin(\pi t/32) + 0.6\sin(\pi t/10), & 1750 \le t < 2000 \end{cases} \tag{3.60}$$

Fig. 3.10 shows a very good match between the actual output and the network output. Table

3.10 shows the test RMSEs of $y_{p1}$ and $y_{p2}$. We also compare the performance of MRIT2NFS with that of memory NN (MNN) [70], feed-forward type-1 and type-2 FNNs, and recurrent type-1 FNNs. The MNN is a kind of recurrent NN and has been applied to the same problem in [7]. For the recurrent FNNs, we use the same training data, test data, and the number of training epochs as those for MRIT2NFS, except in the case of the MNN, where a total number of 77000 time steps are used in [70]. Table 3.10 shows that the performance of MRIT2NFS is better than that of feed-forward and recurrent networks.

In this case also we investigate the noise tolerance of MRIT2NFS with the same three levels of noise that are used in the previous example. Table 3.11 summarizes the results for feed-forward type-1 and type-2 FNNs, TRFN [7], RSEIT2FNN [29], and MRIT2NFS over 30 Monte Carlo realizations. In this case too we find that under noisy environments, the test RMSEs of MRIT2NFS is better than those of the compared networks.

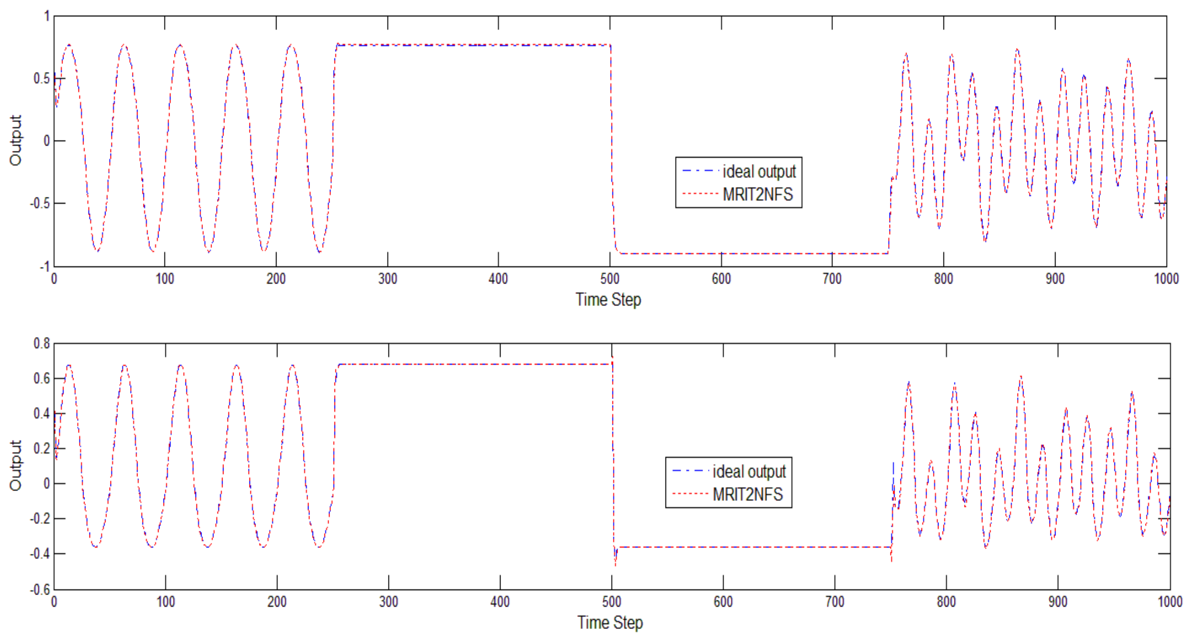

Fig. 3.11. Output of the MIMO system (dashed-dotted curve) and MRIT2NFS (dotted curve) in Example 4. (a) Output $y_{p1}$. (b) Output $y_{p2}$

TABLE 3.10 PERFORMANCE OF MRIT2NFS AND OTHER RECURRENT MODELS FOR MIMO PLANT IDENTIFICATION IN EXAMPLE 4.

| Models | Number of Rules | Number of Parameters | Training RMSE | Test RMSE $y_{p1}$ | Test RMSE $y_{p2}$ |
|---|---|---|---|---|---|
| Feedforward Type-1 FNN | 7 | 126 | 0.0422 | 0.0373 | 0.0480 |
| MNN [70] | – | 131 | – | 0.0186 | 0.0327 |
| RFNN [10] | 13 | 182 | 0.0687 | 0.0824 | 0.0801 |
| WRFNN [11] | 7 | 182 | 0.0449 | 0.0472 | 0.0502 |
| TRFN-S [7] | 7 | 189 | 0.0382 | 0.0396 | 0.0383 |
| Feedforward Type-2 FNN | 4 | 128 | 0.0496 | 0.0411 | 0.0423 |
| RSEIT2FNN [29] | 3 | 126 | 0.0036 | 0.0081 | **0.0113** |
| MRIT2NFS | 3 | 132 | 0.0039 | **0.0066** | 0.0116 |

TABLE 3.11 PERFORMANCE OF MRIT2NFS AND OTHER MODELS WITH DIFFERENT NOISE LEVELS IN EXAMPLE 4.

| Models | | Feedforward Type-1 FNN | Feedforward Type-2 FNN | TRFN-S | RSEIT2FNN | MRIT2NFS |
|---|---|---|---|---|---|---|
| Number of Rules | | 7 | 4 | 7 | 3 | 3 |
| Number of Parameters | | 126 | 128 | 189 | 126 | 132 |
| Test RMSE $y_{p1}$ | STD=0.3 | 0.256 | 0.189 | 0.188 | **0.16** | 0.165 |
| | STD=0.5 | 0.417 | 0.307 | 0.316 | 0.316 | **0.258** |
| | STD=0.7 | 0.578 | 0.420 | 0.427 | 0.427 | **0.354** |
| Test RMSE $y_{p2}$ | STD=0.3 | 0.197 | 0.145 | 0.143 | 0.090 | **0.078** |
| | STD=0.5 | 0.322 | 0.233 | 0.226 | 0.155 | **0.122** |
| | STD=0.7 | 0.443 | 0.323 | 0.298 | 0.227 | **0.193** |

### *3.4.5 Example 5 (HANG nonlinear system)*

This example uses an MRIT2NFS-$\varepsilon$ to deal with the nonlinear characteristics of a plant. This is a very well studied system but it is not recurrent in nature. The HANG [72] system is defined by the equation:

$$y = (1 + x_1^{-1} + x_2^{-1.5})^2 \ , \qquad 1 \le x_1, x_2 \le 5. \tag{3.61}$$

To obtain the training data as done in [72], we have generated 50 random pairs of $(x_1, x_2), 1 \le x_1, x_2 \le 5$, and computed the corresponding outputs using Eq. (3.61). This system has only two current control inputs $x_1(t)$ and $x_2(t)$ ($n_u$=2) and no external output ($n_o$=0). Therefore, only two input states $x_1$ and $x_2$ are fed as input to the MRIT2NFS. The current output of the plant depends on current control inputs with no time delay. Thus, in this case we set $N_1$ =0 and $O_1$ =0 in the MRIT2NFS consequent part. Fig. 3.12 shows a pictorial representation of HANG. The threshold $f_{th}$ is set to be 0.02, and the number of rules is 3 after 100 epochs of training. Table 3.12 shows the number of rules, parameters, and test RMSE of the MRIT2NFS. For comparison, Table 3.12 also shows the test RMSEs of feed-forward type-1 and type-2 FNNs, and TRFN using the same I-O data. The result shows that the test error of the MRIT2NFS-$\varepsilon$ is marginally better than feed-forward type-2 FNN. Since this is not a dynamic system, the recurrent architecture is not likely to yield any additional benefits.

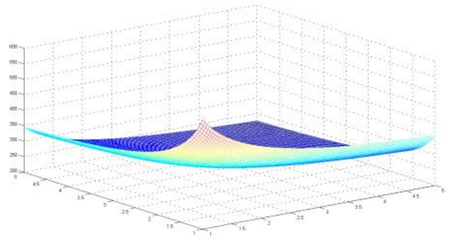

Fig. 3.12. A pictorial representation of HANG.

TABLE 3.12 PERFORMANCE OF MRIT2NFS-$\varepsilon$ AND OTHER MODELS FOR MODELING OF NONLINEAR SYSTEM IN EXAMPLE 5.

| Models | Feedforward Type-1 FNN | TRFN-S [7] | Feed-forward Type-2 FNN | MRIT2NFS -$\varepsilon$ (0.8) |
|---|---|---|---|---|
| Iteration | 100 | 100 | 100 | 100 |
| Number of Rules | 5 | 3 | 3 | 3 |
| Number of Parameters | 35 | 30 | 36 | 40 |
| Test RMSE | 0.312 | 0.244 | 0.26 | **0.206** |

# Chapter 4

# Conclusions

This dissertation proposes the combination of a novel recurrent structure and type-1 and type-2 NFSs, namely an interactively recurrent self-evolving fuzzy neural network (IRSFNN) and a mutually recurrent interval type-2 neural fuzzy system (MRIT2NFS), respectively, to identify time-varying systems (systems with temporal behavior). Identification of such systems is difficult because the plant output depends on the present state as well as on previous states and past outputs. Both IRSFNN and MRIT2NFS approaches are quite effective in modeling dynamic systems because of their on-line learning manner and recurrent structure. For the two proposed models, their learning methods are based on simultaneous structure and parameter learning. The online structure learning algorithm enables the network to efficiently identify the required structure of the network and does not need to set any initial structure in advance. The proposed recurrent structure not only effectively stores local (internal) information but also collects critical information from global transaction. Additionally, we next introduce the rest of IRSFNN's merit. The IRSFNN employs the functional-link NN (FLNN) for the consequent part of its fuzzy rule. In the simulations, the functional-link-based IRSFNN outperformed the TSK-type IRSFNN. The learning algorithm of the variable-dimensional Kalman filter helps improve network accuracy by tuning the consequent part parameters, and accounts for the change in the network size during learning.

Next, we introduce the MRIT2NFS structure and its merits. Unlike the IRSFNN, the MRIT2NFS uses type-2 fuzzy sets in the premise clause of fuzzy rules, which is able to effectively address rule uncertainties associated with information and data in the knowledge base. For the structure learning of MRIT2NFS, we have used type-2 fuzzy set theoretic concepts to evolve the structure of the network in an online manner. We use the rule-ordered Kalman filter algorithm to tune the consequent parameters to yield a very effective learning.

We have also proposed a strategy to eliminate redundant recurrent weights. This is quite effective particularly when we have more rules. We have tested our system on several dynamic systems and one algebraic (non-recurrent) system and compared the performance with several existing state-of-the-art systems. We have compared the performance of our system with both Type-1 and Type-2 state-of-the-art FNNs. Among the two Type-1 FNNs, one is of feed-forward type and other is a recurrent network with linear consequents. Our results have demonstrated the consistently superior performance of MRIT2NFS over both the Type-1 (recurrent and feed-forward) systems as well as recurrent Type-2 systems. We have demonstrated the superior performance both in terms of modeling capability as well as noise handling capability. Our system is found to be quite robust with respect to noisy data. In order to make a fair comparison, we have tried to keep the number of free parameters in all systems comparable.

In the future, we would like to reduce the consequent parameters because too many parameters may result in computational intensive when number of rules and input variables are larger. The FLNN consists of trigonometric functions to replace conventional TSK-type in the consequent part of the IRSFNN. Therefore, the choice of useful features is an important issue. In this investigation for the consequent part of MRIT2NFS, we have assumed knowledge about the system order and number of delayed inputs. In absence of this information, one can use sufficiently large number of delayed inputs and past outputs in the consequents and then find the useful ones using a concept similar to the feature attenuating gates as done in [44]. However, use of such a concept may not ensure that all useful delayed inputs and outputs are consecutive in time. The other alternative could be to use a validation scheme to find the best combination of number of lagged inputs and outputs to be used in the consequents. Second, our proposed models enable to widely use in a variety of applications, that is, prediction and estimation of bio-engineering, and visual and speech recognizing.

# Bibliography

[1] Y. Gao and M. J. Er, “NARMAX time series model prediction: feedforward and recurrent fuzzy neural network approaches,” *Fuzzy Sets and Syst.*, vol. 150, no. 2, pp.331–350, Mar. 2005.

[2] C. F. Juang and C. T. Lin, “A recurrent self-organizing neural fuzzy inference network,” *IEEE Trans. Neural Networks*, vol. 10, no. 4, pp. 828–845, Jul. 1999.

[3] G. C. Mouzouris and J. M. Mendel, “Dynamic nonsingleton fuzzy logic systems for nonlinear modeling,” *IEEE Trans. Fuzzy Syst.*, vol. 5, no. 2,pp. 199–208, May 1997.

[4] Y. C. Wang, C. J. Chien, and C. C. Teng, “Direct adaptive iterative learning control of nonlinear systems using an output-recurrent fuzzy neural network,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 34, no. 5, 2144–2154, Jun. 2004.

[5] D. G. Stavrakoudis and J. B. Theocharis, “A recurrent fuzzy neural network for adaptive speech prediction,” in *Proc. IEEE Internat. Conf. on Syst., Man, Cybern.*, Montreal, QC, Canada, Oct. 2007, pp. 2056–2061.

[6] J. B. Theocharis, “A high-order recurrent neuro-fuzzy system with internal dynamics: application to the adaptive noise cancellation,” *Fuzzy Sets and Syst.*, vol.157, no. 4, pp. 471–500, Feb. 2006.

[7] C. F. Juang, “A TSK-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithm,” *IEEE Trans. Fuzzy Syst.*, vol. 10, no. 2, pp. 155–170, Apr. 2002.

[8] C. F. Juang and J. S. Chen, “Water bath temperature control by a recurrent fuzzy controller and its FPGA implementation,” *IEEE Trans. Ind. Electron.*, vol. 53, no. 3, pp. 941–949, Jun. 2006.

[9] J. Zhang and A. J. Morris, “Recurrent neuro-fuzzy networks for nonlinear process modeling,” *IEEE Trans. Neural Networks*, vol. 10, no. 2, pp. 313–326, Mar. 1999.

[10] C. H. Lee and C. C. Teng, "Identification and control of dynamic systems using recurrent fuzzy neural networks," *IEEE Trans. Fuzzy Syst.*, vol. 8, no. 4, pp. 349–366, Aug. 2000.

[11] C. J. Lin and C. C. Chin, Prediction and identification using wavelet-based recurrent fuzzy neural networks, *IEEE Trans. Syst. Man Cybern. B, Cybern.*, vol. 34, no. 5, pp. 2144–2154, Oct. 2004.

[12] P. A. Mastorocostas and J. B. Theocharis, "A recurrent fuzzy-neural model for dynamic system identification," *IEEE Trans. Syst. Man, Cybern. B, Cybern.*, vol. 32, no.2, pp. 176–190, Apr. 2002.

[13] J. S. Wang and Y. P. Chen, "A Hammerstein Recurrent Neurofuzzy Network with an Online Minimal Realization Learning Algorithm," *IEEE Trans. Fuzzy Syst.* vo.16, no. 6, pp.1597–1612, Dec. 2008.

[14] C. F. Juang, Y. Y. Lin, and C. C. Tu, "A recurrent self-evolving fuzzy neural network with local feedbacks and its application to dynamic system processing," Fuzzy Sets and Systems, vol. 161, pp. 2552–2568, Oct. 2010.

[15] J. Hu and J. Wang, "Global Stability of Complex-valued Recurrent Neural Networks with Time-Delays," *IEEE Trans. Neural Networks Learning Syst.*, vol. 23, no. 6, pp. 853–865, Jun. 2012.

[16] F. Ornelas-Tellez, E. N. Sanchez, and A. G. Loukianov, "Discrete -Time Neural Inverse Optimal Control for Nonlinear via Passivation," *IEEE Trans. Neural Networks. Learning Syst.*, vol. 23, no. 8, pp. 1327–1339, Aug. 2012.

[17] L. X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans Syst., Man, Cybern.*, vol. 22, no. 6, pp. 1414–1427, Nov. –Dec. 1992.

[18] C. J. Lin and C. T. Lin, "An ART-based fuzzy adaptive learning control network," *IEEE Trans. Fuzzy Syst.*, vol. 5, no. 4, pp. 477–496, Nov. 1997.

[19] W. S. Lin, C. H. Tsai, and J. S. Liu, “Robust neuro-fuzzy control of multivariable systems by tuning consequent membership functions,” *Fuzzy Sets and Syst.*, vol. 124, no. 2, pp. 181–195, Dec. 2001.

[20] J. S. R. Jang, “ANFIS: Adaptive-network-based fuzzy inference system,” *IEEE Trans. on Syst., Man, and Cybern.*, vol. 23, pp. 665–685, Jun. 1993.

[21] C. F. Juang and C. T. Lin, “An on-line self-constructing neural fuzzy inference network and its applications,” *IEEE Trans. Fuzzy Syst.*, vol. 6, no.1, pp. 12–31, Feb. 1998.

[22] Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks*, MA: Addison–Wesley, 1989.

[23] J. C. Patra, R. N. Pal, and B. N. Chatterji, and G. Panda, “Identification of nonlinear dynamic systems using functional link artificial neural networks,” *IEEE Trans Syst., Man, Cybern.*, vol. 29, pp. 254–262, Apr. 1999.

[24] M. Alci, “Fuzzy Rule-base Driven Orthogonal Approximation,” *Neural Comp. Appl.*, vol. 17, nos. 5–6, pp. 501–507, 2008.

[25] M. Alci, “Fuzzy Systems on Orthogonal Bases,” *Dynamics of Continuous Discrete and Impulsive Systems-Series A Mathematical Analysis*, vol. 14, 2007, pp. 671–676.

[26] M. Alci and S. Beyhan, “Trigonometric Functions Based Neural Networks for System Identification,” *in Proc. 5th IFAC Intl.*, May 2007, pp. 141–144.

[27] S. Beyhan and M. Alci, “An orthogonal ARX network for Identification and Control of Nonlinear Systems,” in *Proc. XXII Int. Symp. Inf., Commun. Autom. Technolo.*, Oct. 2009, pp. 1–5.

[28] J. Y. Chang, Y. Y. Lin, M. F. Han, and C. T. Lin, “A functional-link based Interval Type-2 Compensatory Fuzzy Neural Network for Nonlinear System modeling,” in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Jun. 2011, pp. 939–943.

[29] C. F. Juang, R. B. Huang, and Y. Y. Lin, “A recurrent self-evolving interval type-2 fuzzy neural network for dynamic system processing,” *IEEE Trans. Fuzzy Syst.*, vol. 17, no. 5,

pp. 1092–1105, Oct. 2009.

[30] G. Chen, Y. Chen, and H. Ogmen, “Identifying chaotic system via a wiener-type cascade model,” *IEEE Trans. Contr. Syst.*, vol. 17, no. 5, pp. 29–36, Oct. 1997.

[31] H. Tamura, K. Tanno, H. Tanaka, C. Vairappan, and Z. Tang, “Recurrent type ANFIS using local search technique for time series prediction,” in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, Dec. 2008, pp. 380–383.

[32] C. J. Lin, C. H. Chen, and C. T. Lin, “Efficient self-evolving evolutionary learning for neuro-fuzzy inference systems,” *IEEE Trans. Fuzzy Syst.*, vol.16, no. 6, pp. 1476–1490, Dec. 2008.

[33] S. Paul, S. Kumar, “Subsethood-product fuzzy neural inference system (SuPFuNIS),” *IEEE Trans. Neural Networks*, vol. 13, no. 3, pp. 578–599, May 2002.

[34] C. F. Juang, C. H. Hsu, and I. F. Chung, “Automatic construction of feedforward/recurrent fuzzy systems by clustering-aided simplex particle swarm optimization,” *Fuzzy Sets and Syst.*, vol. 158, no. 18, pp. 1979–1996, Sep. 2007.

[35] C. J. Lin, C. H. Chen, and C. T. Lin, “A hybrid of cooperative particle swarm optimization and cultural algorithm for neural fuzzy networks and its prediction applications,” *IEEE Trans. Syst., Man, Cybern., C, Appl. Rev.*, vol. 39, no. 1, pp. 55–68, Jan. 2009.

[36] Y. Chen, B. Yang, and J. Dong, “Time-series prediction using a local linear wavelet neural network,” *Neurocomput.*, vol. 69, nos. 4–6, pp. 449–465, Jan. 2006.

[37] S. Yilmaz and Y. Oysal, “Fuzzy wavelet neural network models for prediction and identification of dynamic system,” *IEEE Trans. Neural Networks*, vol. 21, no.10, pp. 1599–1609, Oct. 2010.

[38] J. Kim and N. Kasabov, “HyFIS: Adaptive neuro-fuzzy inference systems and their application to nonlinear dynamical systems,” *Neural Netw.*, vol. 12, no. 9, pp. 1301–1319, Nov. 1999.

[39] E. Y. Cheu, H. C. Quek, and S. K. Ng, “TNFIS: Tree-based neural fuzzy inference system,” in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Jun. 2008, pp. 398–405.

[40] N. K. Kasabov, J. Kim, M. J. Watts, and A. R. Gray, “FuNN/2—Afuzzy neural network architecture for adaptive learning and knowledgeacquisition,” *Inform. Sci.*, vol. 101, nos. 3–4, pp. 155–175, Oct. 1997.

[41] M. Alci and M. H. Asyali, “Nonlinear system identification via Laguerre network based fuzzy system,” *Fuzzy Sets and Syst.*, vol. 160, no. 24, Dec. 2009.

[42] G. E. P. Box, *Time Series Analysis, Forecasting and Control*, San Francisco, CA: Holden Day, 1970.

[43] J. B. Theocharis and G. Vachtsevanos, “Recursive learning algorithms for training fuzzy recurrent models,” *Int. J. Intell. Syst.*, vol. 11, no. 12, pp. 1059–1098, 1996.

[44] C. F. Juang and C. D. Hsieh, “A Locally recurrent fuzzy neural network with support vector regression for dynamic-system modeling,” *IEEE Trans. Fuzzy Syst.*, vol.18, no. 2, pp. 261–273, Apr. 2010.

[45] N. N. Karnik, J. M. Mendel, and Q. Liang, “Type-2 fuzzy logic systems,” *IEEE Trans. Fuzzy Syst.*, vol.7, no. 6,pp. 643–658, Dec. 1999.

[46] J. M. Mendel, Uncertain Rule-Based Fuzzy Logic System: Introduction and New Directions, Prentice Hall, Upper Saddle River, NJ, 2001.

[47] J. M. Mendel and R.I. John, “Type-2 fuzzy sets made simple,” *IEEE Trans. Fuzzy Syst.*, vol. 10, no.2, pp.117–127, Apr. 2002.

[48] J. M. Mendel, “Type-2 fuzzy sets and systems: An overview”, *IEEE Computational Intelligence Magazine*, vol. 2, no. 1, pp 20–29, 2007.

[49] R. John and S.Coupland, “Type-2 fuzzy logic: A historical view,” *IEEE Computational Intelligence Magazine*, vol. 2, no.1, pp57–62, 2007.

[50] J. Zeng, L. Xie, and Z. Q. Liu, “Type-2 fuzzy Gaussian mixture models,” *Pattern Recognition*, vol. 41,no. 12, pp 3636–3643, Dec. 2008.

[51] Q. Liang and J. M. Mendel, “Equalization of nonlinear time-varying channels using type-2 fuzzy adaptive filters,” *IEEE Trans. Fuzzy Syst.*, vol. 8, no. 551–563, Oct. 2000.

[52] Q. Liang and J. M. Mendel, “Interval type-2 fuzzy logic systems: theory and design,” *IEEE Trans. Fuzzy Syst.*, vol. 8, no. 5, pp. 535–550, Oct. 2000.

[53] H. Hagras, “Comments on dynamical optimal training for interval type-2 fuzzy neural network (T2FNN),” *IEEE Trans. Syst., Man, Cybern. B*, vol. 36, no. 5, pp. 1206–1209, Oct. 2006.

[54] J. Zeng and Z. Q. Liu, “Type-2 fuzzy hidden Markov models and their application to speech recognition,” *IEEE Trans. Fuzzy Syst.*, vol. 14, no. 3, pp. 454–467, Jun. 2006.

[55] C. H. Wang and F.C. H Rhee.” Uncertain fuzzy clustering: interval type-2 fuzzy approach to C-means,” *IEEE Trans. Fuzzy Syst.*, vol. 15, no. 1, pp. 107–120, Feb. 2007.

[56] G. M. Mendez and O. Castillo, “Interval type-2 TSK fuzzy logic systems using hybrid learning algorithm, ”in Proc. *IEEE Int. Conf. Fuzzy Syst.*, pp. 230–235, May 22–25, 2005.

[57] C. H. Lee, Y. C. Lin, and W. Y. Lai, “Systems identification using type-2 fuzzy neural network (Type-2 FNN) systems,” in *Proc. IEEE Int. Symp. Computational Intelligence in Robotics and Automation*, vol. 3, pp. 1264–1269, Jul. 16–20, 2003.

[58] C. H. Wang, C. S. Cheng, and T. T. Lee, “Dynamical optimal training for interval type-2 fuzzy neural network (T2FNN),” *IEEE Trans. Syst., Man, Cybern. B*, vol. 34, no. 3, pp. 1462-1477, Jun. 2004.

[59] J. M. Mendel, “Computing derivatives in interval type-2 fuzzy logic system,” *IEEE Trans. Fuzzy Syst.*, vol. 12, no. 1, pp. 84–98, Feb. 2004.

[60] Y. C. Lin and C. H. Lee, “System Identification and Adaptive Filter Using a Novel Fuzzy Neuro system,” *Int. Journal of Computational Cognition*, vol. 5, no. 1, pp. 15–26, 2007.

[61] W. S. Chan, C. Y. Lee, C. W. Chang, and Y. H. Chang, “Interval type-2 fuzzy neural network for ball and beam systems,” *IEEE conf. Syst. Sci. Eng.*, pp. 315–320, Jul. 1–3, 2010.

[62] C. F. Juang and Y. W. Tsao, ”A self-evolving interval type-2 fuzzy neural network with online structure and parameter learning,” *IEEE Trans. Fuzzy Syst.*, vol. 16, no. 6, pp. 1411–1424, Dec. 2008.

[63] C. D. Li, J. Q. Yi, and D. B. Zhao, “Interval type-2 fuzzy neural network controller (IT2FNNC) and its application to a coupled-tank liquid-level control system, “in *Proc. Int. Conf. Innovative Comput. Inf. Contr.*, pp. 508, Jun. 18–20, 2008.

[64] F. J. Lin and P. H. Chou, “Adaptive control of two-axis motion control system using interval type-2 fuzzy neural network,” *IEEE Trans. Ind. Electron.*, vol. 56, no. 1, pp. 178–193, Jan. 2009.

[65] C. F. Juang, R. B. Huang, and W. Y. Cheng, “An Interval Type-2 Fuzzy Neural Network with Support Vector Regression for Noisy Regression Problems,” *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 4, pp. 686–699, Aug. 2010.

[66] R. H. Abiyev and O. Kaynak, “Type-2 Fuzzy Neural Structure for Identification and control of time-varying plants,” *IEEE Trans. Ind. Electron.*, vol. 57, no. 12, pp. 4147–4159, Dec. 2010.

[67] C. H. Lee, H. H. Chang, C. T. Kuo, J. C. Chien, and T. W. Hu, “A Novel Recurrent Interval Type-2 Fuzzy Neural Network for Nonlinear Channel Equalization,” in *Proc. Int. Muticonf. Engineers and computer scientists*, vol. 1, March 18–20, 2009.

[68] Y. Y. Lin, J. Y. Chang, and C.T. Lin,” An Internal/Interconnection Recurrent Type-2 Fuzzy Neural Network (IRT2FNN) for Dynamic System Identification,” in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, pp. 733–737, Oct 10–13, 2010.

[69] C. F. Juang, Y. Y. Lin, and R. B. Huang, ”Dynamic system modeling using a recurrent interval-valued fuzzy neural network and its hardware implementation,” *Fuzzy Sets Syst.*, vol. 179, pp. 83–99, May 2011.

[70] P. S. Sastry, G. Ssntharam, and K. P. Unnikrishnan, “Memory neural networks for identification and control of dynamic systems,” *IEEE Trans. Neural Networks*, vol. 5, pp. 306–319, Mar. 1994.

[71] G. Chen, Y. Chen, and H. Ogmen, “Identifying chaotic system via a wiener-type cascade model,” *IEEE Trans. Contr. Syst.*, pp. 29–36, Oct. 1997.

[72] M. Sugeno and T. Yasukawa, “A fuzzy-logic-based approach to qualitative modeling,” *IEEE Trans. Fuzzy Syst.*, vol. 1, no. 1, pp. 7–31, Feb. 1993.

[73] A. Laha, N. R. Pal and J. Das, “Land cover classification using fuzzy rules and aggregation of contextual information through evidence theory,” *IEEE Trans. Geosci. Remote Sensing*, vol. 24, no. 6, pp.1633–1641, 2006.

[74] N. R. Pal and S. Saha, “Simultaneous Structure Identification and Fuzzy Rule Generation for Takagi–Sugeno Models,” *IEEE Trans. Syst., Man, Cybern. B*, vol. 38, no. 6, pp. 1626–1638, Dec. 2008.

[75] C. S. Lee, M. H. Wang, and H. Hagras, “A Type-2 Fuzzy Ontology and Its Application to Personal Diabetic-Diet Recommendation,” *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 2, pp. 374–395, Apr. 2010.

[76] D. Hidalgo, O. Castillo, and P. Melin, “Type-1 and Type-2 fuzzy inference systems as integration methods in modular neural networks for multimodal biometry and its optimization with genetic algorithms,” *Inform. Sci.*, vol. 179, pp. 2121–2145, 2009.

[77] X. Chen, Y. Li, R. Harrison, and Y. Q. Zhang, “Type-2 fuzzy logic-based classifier fusion for support vector machines,” *Applied Soft Computing*, vol. 8, pp.1222–1231, 2008.

[78] X. Liu and J. M. Mendel, “Connect Karnik–Mendel Algorithms to Root-Finding for Computing the Centroid of an Interval Type-2 Fuzzy Set,” *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 4, pp. 652–565, Aug. 2011.

[79] D. Wu, J. M. Mendel, and S. Coupland, “Enhanced Interval Approach for Encoding Words Into Interval Type-2 Fuzzy Sets and Its Convergence Analysis,” *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 3, pp. 499–513, June. 2012.

[80] D. Wu and J. M. Mendel, “Linguistic Summarization Using IF–THEN Rules and Interval Type-2 Fuzzy Sets,” *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 1, pp. 136–151, Feb. 2011.

[81] O. Linda and M. Manic, “General Type-2 Fuzzy C-Means Algorithm for Uncertain Fuzzy Clustering,” *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 5, pp. 883–897, Oct. 2012.

[82] D. Wu and J. M. Mendel, “On the Continuity of Type-1 and Interval Type-2 Fuzzy Logic Systems,” *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 1, pp. 179–192, Feb 2011.

[83] C. Y. Yeh, W. H. R. Jeng, and S. J. Lee, “An Enhanced Type-Reduction Algorithm for Type-2 Fuzzy Sets,” *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 2, pp. 227–240, Apr. 2011.

[84] D. Zhai and J. M. Mendel, “Enhanced Centroid-Flow Algorithm for Computing the Centroid of General Type-2 Fuzzy Sets,” *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 5, pp. 939–956, Oct. 2012.

[85] D. Zhai and J. M. Mendel, “Comment on “Toward General Type-2 Fuzzy Logic Systems Based on zSlices,” *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 5, pp. 996–997, Oct. 2012.

[86] D. Zhai and J. M. Mendel, “Computing the Centroid of a General Type-2 Fuzzy Set by Means of the Centroid-Flow Algorithm,” *IEEE Trans. Fuzzy Syst.*, vol. 19, no.3, pp. 401–422, June. 2011.

[87] O. Linda and M. Manic, “Monotone Centroid Flow Algorithm for Type Reduction of General Type-2 Fuzzy Sets,” *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 5, pp. 805–819, Oct. 2012.

[88] R. Hosseini, S. D. Qanadli, S. Barman, M. Mazinani, T. Ellis, and J. Dehmeshki, "An Automatic Approach for Learning and Tuning Gaussian Interval Type-2 Fuzzy Membership Functions Applied to Lung CAD Classification System," *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 2, pp. 224–234, Apr. 2012.

[89] M. Nie and W. W. Tan, "Analytical Structure and Characteristics of Symmetric Karnik–Mendel Type-Reduced IntervalType-2 Fuzzy PI and PD Controllers," *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 3, pp. 416–430, 2012.

[90] D. Wu, "On the Fundamental Differences Between Interval Type-2 and Type-1 Fuzzy Logic Controllers," *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 5, pp. 832–848, Oct. 2012.

[91] S. Barkat, A. Tlemcani, and H. Nouri, "Noninteracting Adaptive Control of PMSM Using Interval Type-2 Fuzzy Logic Systems," *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 5, pp. 925–936, Oct. 2011.

[92] R. A. Aliev, W. Pedrycz, B. G. Guirimov, R. R. Aliev, U. Ilhan, M. Babagil, and S. Mammadli, "Type-2 fuzzy neural networks with fuzzy clustering and differential evolution optimization," *Inf. Sci.*, vol. 181, pp. 1591–1608, 2011.

## Vita

## 博士候選人學經歷資料

姓名：林洋印 (Yang-Yin Lin)

性別：男

生日：民國 71 年 5 月 28 日

出生地：高雄市

論文題目：

中文：具有相互影響之遞迴式自我演化類神經模糊系統及其應用

英文：A Novel Recurrent Self-evolving Neural Fuzzy System and Its Applications

學歷：

- 民國 94 年 6 月，高雄應用科技大學電子工程系畢業
- 民國 97 年 6 月，中興大學電機工程所碩士班畢業
- 民國 102 年 4 月，國立交通大學電機與控制工程學系博士班，提博士論文口試

# Publication List

## 著作目錄

姓名：林洋印(Yang-Yin Lin)

已刊登或被接受之期刊論文：

[1] **<u>Yang-Yin Lin</u>**, Jyh-Yeong Chang, and Chin-Teng Lin, “Identification and Prediction of Dynamic Systems Using an Interactively Recurrent Self-evolving Fuzzy Neural Network (IRSFNN),” *IEEE Trans. Neural Netw. Learning Syst.*, vol. 24, no. 2, pp. 310–321, Feb. 2013.

[2] **<u>Yang-Yin Lin</u>**, Jyh-Yeong Chang, and Chin-Teng Lin, “A TSK-type-based Self-Evolving Compensatory Interval Type-2 Fuzzy Neural Network (TSCIT2FNN) and Its Applications,” accepted to appear in *IEEE Trans. on Ind. Electron.*, 2013.

[3] **<u>Yang-Yin Lin</u>**, Jyh-Yeong Chang, Nikhil R. Pal and Chin-Teng Lin, “A Mutually Recurrent Interval Type-2 Neural Fuzzy System (MRIT2NFS) with Self-evolving Structure and Parameters,” *IEEE Trans. Fuzzy Syst.*, vol. 21, no. 6, 2013.

[4] Chia-Feng Juang, **<u>Yang-Yin Lin</u>**, Ren-Bo Huang, “Dynamic system modeling using a recurrent interval-valued fuzzy neural network and its hardware implementation,” Fuzzy Sets and Syst., vol. 179, no.1, pp. 83-99, 2011.

[5] Chia-Feng Juang, **<u>Yang-Yin Lin</u>**, Chiu-Chuan Tu: A recurrent self-evolving fuzzy neural network with local feedbacks and its application to dynamic system processing. Fuzzy Sets and Syst., vol. 161, no. 19, pp. 2552-2568, 2010.

[6] Chia-Feng Juang, Ren-Bo Huang, **<u>Yang-Yin Lin</u>**, “A Recurrent Self-Evolving Interval Type-2 Fuzzy Neural Network for Dynamic System Processing,” *IEEE Trans. on Fuzzy Syst.*, vol.17, no. 5, pp. 1092-1105, 2009.

研討會論文：

[1] Jyh-Yeong Chang, **<u>Yang-Yin Lin</u>**, Ming-Feng Han and Chin-Teng Lin, “A Functional-Link based Interval Type-2 Compensatory Fuzzy Neural Network for Nonlinear System Modeling,” 2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011), Taipei, Taiwan, pp. 939–943 ,Jun. 27–30, 2011.

[2] Chin-Teng Lin, Ming-Feng Han, **<u>Yang-Yin Lin</u>**, Shih-Hui Liao and Jyh-Yeong Chang, “Neuro-Fuzzy System Design Using Differential Evolution with Local Information,” *2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011),* Taipei, Taiwan, pp. 1003–1006, Jun. 27–30, 2011.

[3] Jyh-Yeong Chang, **<u>Yang-Yin Lin</u>**, Chin-Teng Lin, Chia-Feng Juang and Li-Wei Ko, “A

Function-Link based Type-2 Fuzzy Neural Network for identification and modeling of time-varying plants," *The 17th National Conference on Fuzzy Theory and its Applications*, Hualien, Taiwan, Nov. 3–4, 2010.

[4] Chin-Teng Lin, Ming-Feng Han, **Yang-Yin Lin**, Jyh-Yeong Chang and Li-Wei Ko, "Differential Evolution based Optimization of Locally Recurrent Neuro-Fuzzy System for Dynamic System Identification," *The 17th National Conference on Fuzzy Theory and its Applications*, Hualien, Taiwan, pp. 702–707, Nov. 3–4, 2010.

[5] **Yang-Yin Lin**, Jyh-Yeong Chang and Chin-Teng Lin, "An Internal/Interconnection Recurrent Type-2 Fuzzy Neural Network (IRT2FNN) for dynamic system identification," *IEEE International Conference on Systems, Man and Cybernetics*, pp. 733–737, 2010.