University of Alberta

A Neuro-fuzzy Architecture Incorporating Complex Fuzzy Logic

by

Zhifei Chen 

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

Edmonton, Alberta
Spring, 2008

Your file Votre référence
ISBN: 978-0-494-45791-7
Our file Notre référence
ISBN: 978-0-494-45791-7

Abstract

Complex fuzzy sets have recently been a topic of interest in the fuzzy systems community. However, to date, no practical application of this concept has yet been proposed. The goal of this thesis is to create a time series forecasting system, which will be the first practical application of the complex fuzzy sets. We have constructed a neuro-fuzzy architecture, named ANCFIS, inducing complex fuzzy rules from time-series. The challenge of this architecture is how to update the parameters of complex fuzzy sets. We have developed a novel derivative-free optimization technique to overcome this problem: the Variable Neighborhood Chaotic Simulated Annealing (VNCSA) algorithm, and compare VNCSA against two existing alternatives: a chaotic simulated annealing technique, and Ant Colony Optimization algorithm. Our comparisons are carried out over one synthetic dataset and five real-world datasets. We found that the VNCSA algorithm leads to the best tracking error in the ANCFIS architecture, for all six datasets.

Acknowledgement

I owe a debt of gratitude to my supervisor, Dr Scott Dick from whom I have learned much about complex fuzzy logic. I wish to acknowledge his support and guidance throughout the project.

Special thanks to my dear friend James Man for his encouragement and his collaboration over the last 10 months.

I would also like to express thanks to my wife and daughter for their support during my working on the thesis.

Table of contents

List of Tables

List of Figures

1 Introduction

The development of complex fuzzy logic [1][2][3], a novel framework for logical reasoning, has been a topic of interest in fuzzy systems in recent years. Complex fuzzy logic is a generalization of traditional fuzzy logic, based on complex fuzzy sets. In the complex fuzzy logic, the inference rules are created and "fired" in a similar-fashion to the traditional fuzzy logic. The predicted output and a complex fuzzy rule are complex fuzzy sets. Complex fuzzy sets are a recently-proposed extension to the standard (type-1) fuzzy set theory. Where a Type-1 fuzzy set **A** in a universe of discourse **U** is characterized by a membership μ_A, which takes values between [0,1], a complex fuzzy set's membership function has the unit disc of the complex plane as its codomain. Equivalently, a complex fuzzy set is a set of ordered pairs $(x, \mu(x))$ where $x \in \mathbf{X}$ is an object from some universal set **X**, and $\mu(x) \in \mathbf{D}$, the set of complex numbers whose modulus is less than or equal to 1 [1][2][3][4]. This novel framework of complex fuzzy logic not only maintains all the advantages of using traditional fuzzy logic, but also at the same time allows for the definition of operations that allow complex fuzzy sets to interact in new and interesting ways. Complex membership functions possess "wave-like" properties, which allow them to constructively and destructively interfere with one another. Although the theory behind complex fuzzy logic is being developed, there have as yet been no applications of complex fuzzy logic to real-world problems. Until such applications demonstrate the utility of complex fuzzy sets and complex fuzzy logic, they will remain theoretical curiosities. Development of a practical complex fuzzy logic system is complicated by the nature of complex membership functions; it is very difficult to build an intuitive

understanding of complex fuzzy sets and their practical meaning, and so elicitation of a complex fuzzy system from expert knowledge is currently infeasible. Thus, we need to use inductive learning to create a complex fuzzy system.

In previous work [1], it was suggested that complex fuzzy sets would be a reasonable model for "approximately periodic" phenomena; patterns that approximately repeat themselves, but are never exactly the same twice. Such phenomena have previously been termed *regular* patterns, with regularity viewed as a generalization of stationarity [98]. The rationale is that complex fuzzy sets might themselves be periodic, with the sine function proposed as a possible candidate in [1]. We have been following up on this suggestion, attempting to build a *complex fuzzy inferential system* [3] to model regular phenomena. At this time, regularity finds its most common and important expression in the form of time-series data, and the practical problem of time-series forecasting. Accordingly, we are attempting to create a time-series forecasting algorithm using complex fuzzy logic; we suspect that such a system will be a simpler, more natural model of the time series than a type-1 fuzzy system.

Standard fuzzy systems, and their extensions, are rule-based expert systems that may be developed in one of two ways: either they can be elicited from a domain expert, or they may be induced from input/output data. Elicitation requires an expert to provide appropriate definitions of all fuzzy sets, as well as the rules themselves. However, since complex fuzzy sets have a two-dimensional membership function, they could not be simply represented by two distinct fuzzy sets. We and others believe complex fuzzy logic is useful when there is significant interaction between the amplitude and phase terms. If there is an existing link between the amplitude and the phase, it is counter-intuitive to

separate them and hope that their relationship can be re-established through an inference process. Therefore, we turn to inductive learning, and specifically artificial neural networks, to develop our complex fuzzy inferential system. Our work extensively modifies the ANFIS (Adaptive Neuro Fuzzy Inference System) architecture [5][6][7], incorporating complex fuzzy sets instead of type-1 fuzzy sets, and replacing the Takagi-Sugeno-Kang (TSK) rules with the interference-based complex fuzzy inference postulated in [3]. We have named this architecture the Adative Neuro-Complex-Fuzzy Inferential System (ANCFIS).

In the ANFIS architecture, layer 1 computes the membership of a crisp (numeric) input in each fuzzy set during the forward pass, and updates the parameters of the fuzzy sets during the backward pass using a gradient descent algorithm. However, when we apply gradient descent to the complex-valued signals in ANCFIS, we find that the partial derivatives of the error function with respect to the real-valued parameters of our complex fuzzy sets are complex-valued, and so we could not directly update these adaptive real-valued parameters as in classic ANFIS. Classical derivative-based methods could not be used here, and thus ANCFIS must utilize a derivative-free optimization technique to update the parameters in layer 1. We have developed a novel algorithm, Variable-Neighborhood Chaotic Simulated Annealing (VNCSA) for this purpose; our first objective in this thesis is to determine whether to use this algorithm to update the parameters of layer 1, or to employ a different derivative-free technique. We compare VNCSA against the well-known Ant Colony Optimization (ACO) technique [8][9][10], as well as a different Chaotic Simulated Annealing (CSA) algorithm [11].

Our second objective in this thesis is to demonstrate the utility of the ANCFIS technique for time series prediction. The performance of ANCFIS technique using VNCSA algorithm is tested against six time series datasets, and the out-of-sample root mean square errors are compared with those obtained by ANCFIS using ant colony optimization (ACO) and ANCFIS using CSA2 [16], as well as the classic ANFIS architecture.

Our main research contributions are summarized as follows:

1. Formulation of an ANCFIS architecture incorporating adaptive neuro fuzzy inference system with complex fuzzy logic. Our work is based on the well-known ANFIS architecture [5][6][7], extending the architecture and modifying the node functions to accommodate complex fuzzy logic. In addition, instead of having separate time-delayed inputs in the ANFIS, they are combined into one single input vector presented to the ANCFIS architecture.
2. Derivation of hybrid learning algorithms, analogous to the ANCFIS algorithm. In the forward pass of hybrid learning algorithm, the consequent parameters $\vec{p}, \vec{q}, r$ are estimated by the least squares method while the premise parameters of MFs are fixed. In the backward pass, the complex membership functions are updated by a combination of gradient descent and derivative-free optimization method while the consequent parameters are fixed.
3. Development of VNCSA algorithm. Based on the simulated annealing algorithm, a derivative-free VNCSA algorithm is proposed, where we introduce the behavior of one-dimension chaotic maps to simulated annealing (SA) to ensure small perturbations from adjacent candidate solution are imposed on the each candidate

solution. Through testing some classical nonlinear unconstraint and constraint benchmark optimization functions, the parameters determining the performance of this algorithm are discussed.

4. Experimental evaluation of the ANCFIS learning algorithm. Six problems in time series prediction are used in our experiments. We contrast ANCFIS using VNCSA with ANCFIS-CSA2, ANCFIS-ACO, and the classical ANFIS architecture.

The remainder of this thesis is organized into four chapters: chapter 2 presents a literature review covering the topics of type-1 fuzzy logic, complex fuzzy logic, ANFIS, complex-valued neural networks, the chaotic characteristics of one-dimension chaotic maps, classical chaotic simulated annealing algorithms, and the principles of ACO in combinational optimization. In chapter 3, we present the ANCFIS architecture. This includes the structure of ANCFIS, the node functions of the forward pass, the error-propagation model for the backward pass, the procedures of VNCSA algorithm and preliminary experimental results for the VNCSA algorithm on unconstrained and constrained benchmark optimization problems. In Section 4, the ANCFIS architecture is tested on a total of six datasets. One is a synthetic dataset comprised of two sinusoidal signals and the other five are forecasting datasets from the literature: Wölfer sunspot numbers [82]-[86], Mackey-Glass chaotic time series [5][6], the Santa Fe time series forecasting competition dataset A (laser) [80][81], the waves time series [82][87] and the Star time series [82]. Chapter 5 contains a summary and a discussion of future work.

2 Literature review

2.1 Type-1 fuzzy logic review

The research and application on type-1 fuzzy logic have made a tremendous progress since Zadeh published the first paper [12]. In 1965, Zadeh firstly introduced fuzzy sets as an extension of the classical notion of set. In recent years, the number and variety of applications of type-1 fuzzy logic have increased significantly. These applications range from consumer products such as cameras, camcorders, washing machines, and microwave ovens to industrial process control, medical instrumentation, decision-support systems, and financial portfolio selection. A type-1 fuzzy set is characterized by a two-dimensional MF. Type-1 fuzzy set theory provides a systematic calculus to handle imprecise and incomplete information linguistically, and it implements numerical calculation by using linguistic labels associated with a membership function. A type-1 fuzzy inference system using fuzzy if-then rules can effectively model human expertise in a specific application. Type-2 fuzzy logic [13][14][15][17][18] represents uncertainty using a function which is itself a type-1 fuzzy number. Sometimes type-2 fuzzy logic is referred to as fuzzy fuzzy because of this. A type-2 fuzzy set is characterized by a three-dimensional MF. Type 2 fuzzy logic is a way of representing and modelling uncertainty and imprecision. The basic premise using type-2 fuzzy logic is that most applications in the general area of modelling decision making have to cope with imprecision in data, knowledge, rules etc. It provides a way of not ignoring this imprecision but using it to make better computer systems. Since complex fuzzy sets are

an extension to type-1 fuzzy set theory, we will focus only on type-1 fuzzy logic. In the following, the fuzzy set refers to type-1 fuzzy set.

2.1.1 Type-1 fuzzy sets

Let X be a space of objects and x be a generic element of X. A classical set A, $A \subseteq X$, is defined by a collection of elements or objects $x \in X$, such that each x can either belong or not belong to the set A. By defining a "characteristic function" for each element $x \in X$, we can represent a classical set A by a set of ordered pairs $(x, 0)$ or $(x, 1)$, which indicates $x \notin A$ or $x \in A$, respectively.

Unlike the aforementioned conventional set, a fuzzy set (Zadeh, 1965) expresses the degree to which an element belongs to a set. Hence the characteristic function of a fuzzy set is allowed to have values between 0 and 1, which denotes the degree of membership of an element in a given set. Obviously, the definition of a fuzzy set is a simple extension of the definition of a classical set in which the characteristic function is permitted to have any values between 0 and 1. If the values of the membership function $\mu_A(x)$ is restricted to either 0 or 1, then A is reduced to a classical set and $\mu_A(x)$ is the characteristic function of A.

Definition of Type-1 Fuzzy sets [5]**:**

If X is a collection of objects expressed generically by x, then a fuzzy set A in X is defined as a set of ordered pairs:

$$A = \{(x, \mu_A(x) \mid x \in X\}. \tag{2.1}$$

The membership function is denoted by $\mu_A(x)$ and it maps each element of X to a membership grade between 0 and 1. Usually X is called the universe of discourse, and it may be composed of discrete objects or continuous space. The construction of a fuzzy

set depends on two things: the identification of a universe of discourse and the specification of a membership function. Alternatively, a fuzzy set A can be also expressed as

$$A = \begin{cases} \sum_{x_i \in X} \mu_A(x_i) / x_i, & \text{if X is a collection of discrete objects.} \\ \int_X \mu_A(x) / x, & \text{if X is a continuous space(usually the real line R).} \end{cases} \tag{2.2}$$

Containment, union, intersection, and complement are the most used operations on fuzzy sets [5],

(1) Containment or subset: Fuzzy set $\boldsymbol{A}$ is contained in fuzzy set $\boldsymbol{B}$ (or equivalently, $\boldsymbol{A}$ is a subset of $\boldsymbol{B}$), in symbols,

$$A \subseteq B \Leftrightarrow \mu_A(x) \leq \mu_B(x). \tag{2.3}$$

(2) Union of two fuzzy sets (disjunction): the union of two fuzzy set $\boldsymbol{A}$ and $\boldsymbol{B}$ is a fuzzy set $\boldsymbol{C}$, whose MF is associated with those of $\boldsymbol{A}$ and $\boldsymbol{B}$ by

$$\mu_C(x) = \mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x), \tag{2.4}$$

where $\vee$ refers to max.

(3) Intersection (conjunction): the intersection of two fuzzy set $\boldsymbol{A}$ and $\boldsymbol{B}$ is a fuzzy set $\boldsymbol{C}$, whose MF is associated with those of $\boldsymbol{A}$ and $\boldsymbol{B}$ by

$$\mu_C(x) = \mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x), \tag{2.5}$$

where $\wedge$ refers to min.

(4) Complement (negation): the complement of fuzzy set A, denoted by $\overline{A}$, is represented as

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x) \tag{2.6}$$

Membership functions

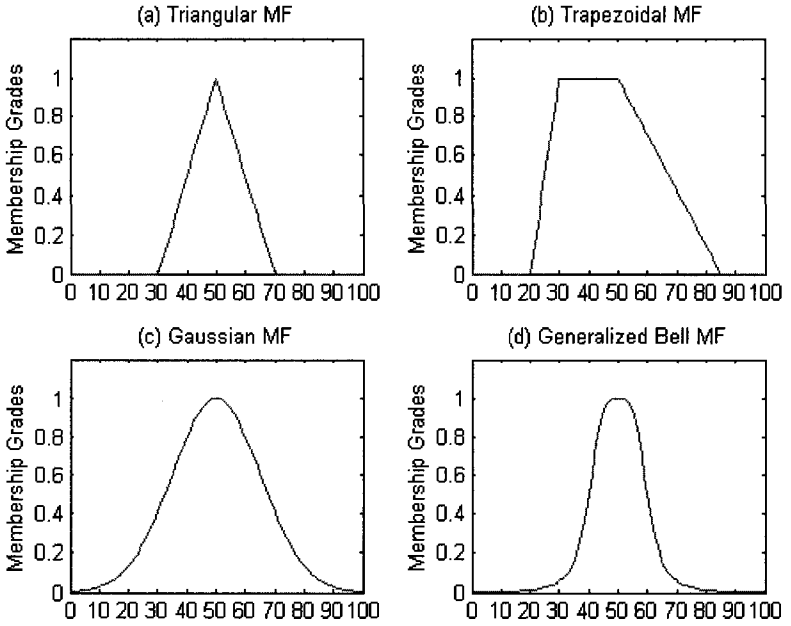


Fig. 2. 1 Examples of four classes of parameterized membership functions:
(a) triangular $(x;30,50,70)$**; (b) trapezoid** $(x;20,30,50,85)$**;**
(c) gaussian $(x;50,15)$**; (d) bell** $(x;10,2,50)$**, from [5]**

A fuzzy set is normally characterized using its membership function. Parameterized membership functions of a single variable are often used; common choices include triangular membership functions (specified by three parameters), trapezoidal membership functions (specified by four parameters), Gaussian membership functions (specified by two parameters) and generalized bell membership functions (specified by three parameters). Figure 2.1 provides examples of these membership functions.

2.1.2 Fuzzy relations and fuzzy composition

Fuzzy relations [99] represent a degree of presence or absence of association, interaction, or interconnectedness between the elements of two or more crisp sets. Let U

and V be two crisp sets. A fuzzy relation $R(U, V)$ is a fuzzy subset of the product space $U \times V$. The relation $R(U, V)$ is characterized by a membership function $\mu_R(x, y)$ where $x \in U$ and $y \in V$. $\mu_R(x, y)$ assigns each pair (x, y) a membership value between [0,1], which represents the degree to which the relation R holds for the elements x and y.

Relations can be explained by a simple example in our daily life using discrete fuzzy sets. Let us analyze the relationship between the color of a tomato x and the grade of maturity y and characterize the linguistic variable color by a crisp set X with three linguistic terms

$$X = \{green, yellow, red\},$$

and similarly the grade of maturity

$$Y = \{verdant, half\_mature, mature\},$$

A crisp formulation of a relation $X \rightarrow Y$ between the two crisp sets is listed in Table 2.1. This crisp relation R can be generalized to consider different degrees of strength of association or interaction between elements of two crisp sets. Degrees of association can be expressed by membership grades in a fuzzy relation in the same way as degrees of the set membership are represented in a fuzzy set. Using this in the fruit example, the table 2.1 can be changed to table 2.2.

Table 2. 1 A relation $X \rightarrow Y$ between the two crisp sets [99]

	verdant	*Half_mature*	*mature*
green	1	0	0
yellow	0	1	0
red	0	0	1

Table 2. 2 A fuzzy relation $X \rightarrow Y$ between the two crisp sets [99]

	verdant	*Half_mature*	*mature*
green	1	0.5	0
yellow	0.3	1	0.5
red	0	0.2	1

Composition of traditional fuzzy relations on the same product space is easily done through set operation such as union, intersection and complement. Composition of fuzzy relations on different product spaces can be done if they share a common set, for example: *R(U,V)* and *S(V,W)*. The composition of these two relations is denoted $C(U,W) = R(U,V) \circ S(V,W)$. The membership function for *C(U,W)* is given by the max-min composition of *R* and *S*

$$\mu_C(x,z) = \max_y \min[\mu_R(x,y), \mu_S(y,z)] = \vee_y [\mu_R(x,y) \wedge \mu_S(y,z], \tag{2.7}$$

where $\wedge$ and $\vee$ denote max and min, respectively.

2.1.3 Fuzzy reasoning

Fuzzy reasoning [5], also called approximate reasoning, is an inference process deriving conclusions based on a set of fuzzy if-then rules and known facts. A fuzzy if-then rule is in the form: *if x is A then y is B*, where *A* and *B* are linguistic values. Often "*x is A*" is called the antecedent or premise whereas "*y is B*" is called the consequence or conclusion. Traditionally, according to the modus ponens, we can inference the truth of a proposition B from the truth of A and the implication $A \rightarrow B$. As a matter of fact, modus ponens is usually used in an approximate manner.

For the single rule with single antecedent, the concept of inference is illustrated as

Premise 1 (fact): x is A'

Premise 2 (rule): *if* x is A then y is B,

Consequence(conclusion): y is B',

where A' is close to A and B' is close to B.

And the membership function of the resulting B' is defined as:

$$\mu_{B'}(y) = \max_x \min[\mu_{A'}(x), \mu_R(x,y)] = \vee_x[\mu_{A'}(x) \wedge \mu_R(x,y)]$$

$$= \vee_x[\mu_{A'}(x) \wedge \mu_A(x,y)] \wedge \mu_B(y), \tag{2.8}$$

where $\wedge$ and $\vee$ denote max and min, respectively.

For single rule with multiple antecedents, the inference procedure is illustrated as

Premise 1 (fact): x is A' and y is B'

Premise 2 (rule): *if* x is A and y is B then z is C,

Consequence(conclusion): z is C'.

And the membership function of the resulting C' is expressed as:

$$\mu_{C'}(z) = \vee_{x,y}[\mu_{A'}(x) \wedge \mu_{B'}(y)] \wedge [\mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z)]$$

$$= \underbrace{\{\vee_x [\mu_{A'}(x) \wedge \mu_A(x)]\}}_{\omega_1} \wedge \underbrace{\{\vee_y [\mu_{B'}(y) \wedge \mu_B(y)]\}}_{\omega_2} \wedge \mu_C(z)$$

$$= (\omega_1 \wedge \omega_2) \wedge \mu_C(z), \tag{2.9}$$

where $\omega_1 \wedge \omega_2$ is called the firing strength, and represents the degree to which the antecedent part of the rule is met. Again, $\wedge$ and $\vee$ denote max and min, respectively.

A graphic interpretations of the single rule with single antecedent and of the single rule with multiple antecedents are shown in Fig. 2.2 and Fig.2.3, separately.

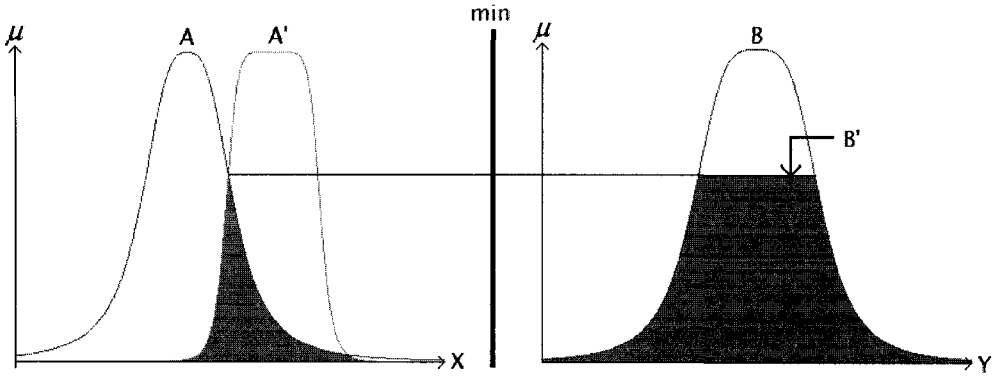


Fig. 2. 2 Graphic interpretation of fuzzy reasoning using the max-min composition, from[5][6].

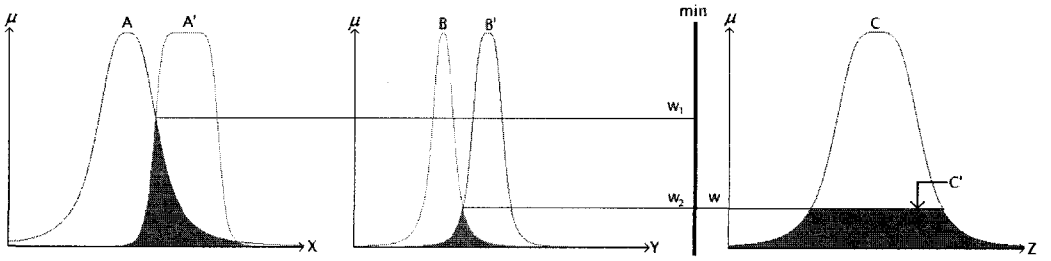


Fig. 2. 3 Approximate reasoning for multiple antecedents from [5][6]

2.1.4 Fuzzy inference systems

The basic computing framework of a fuzzy inference system is composed of a rule base consisting of fuzzy rules, a reasoning mechanism performing the inference procedure based on the rules and given facts to draw a reasonable conclusion, a database/dictionary defining the membership function employed in the fuzzy rules, and a fuzzification interface that converts the crisp data inputs into a fuzzy set. Since the outputs produced by the basic inference system are almost always fuzzy sets, we usually need to extract a crisp value best representing a fuzzy set by a defuzzification method. Therefore, the inference system should include a defuzzification interface that converts

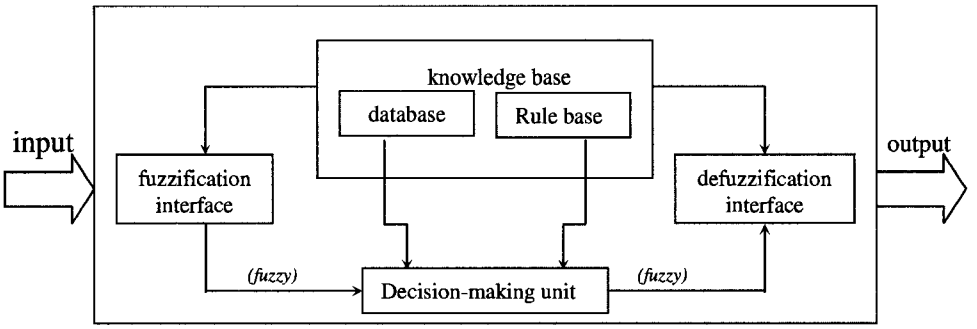


Fig. 2. 4 Fuzzy inference system [5][6]

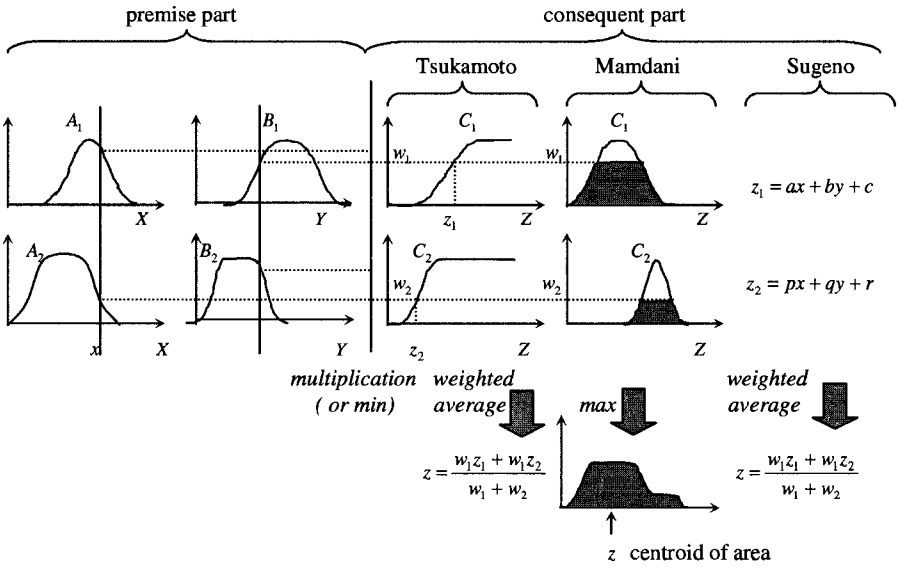


Fig. 2. 5 Normally applied fuzzy if-then rules and fuzzy inference mechanisms, from[5] [6]

the resulting consequents into a crisp output. A fuzzy inference system is shown in Fig. 2.4.

Normally fuzzy inference systems can be split into the three classes shown in Fig.2.5. The main differences between these them lie in the consequences of their fuzzy rules, and aggregation and defuzzification procedures.

The Mamdani fuzzy model [19] was the very first fuzzy controller. The problem was to control a steam engine and boiler combination by aids of a set of linguistic control

rules generated by experienced human operators. Since the plant takes only crisp values as inputs, a defuzzifier must be used to convert a fuzzy set to a crisp value. There are numerous methods for defuzzing a fuzzy set A of a universe of discourse Z; the most common defuzzification strategy is to adopt the centroid of the area under the fuzzy set curve. The computation of the centroid of an area is generally expensive, as it needs integration across a varying function. The centroid of an area, z_{COA}, is represented by

$$z_{COA} = \frac{\int_Z \mu_A(z) z dz}{\int_Z \mu_A(z) dz}, \tag{2.10}$$

where $\mu_A(z)$ is the aggregated output membership function.

The Sugeno fuzzy model [20][21][22], also known as TSK fuzzy model, differs in the form of the rule consequents. A TSK fuzzy rule has the form

if x is A and y is B then z = f(x,y),

where A and B are fuzzy sets in the antecedent and $z = f(x, y)$ is a crisp function in the consequent, $f(x, y)$ can be any function as long as it can completely represent the output of the model within the fuzzy region specified by the antecedent of the rule; polynomial functions are commonly used. When $f(x, y)$ is a 1st order polynomial, the fuzzy inference system is called a first-order Sugeno fuzzy model. If $f(x, y)$ is constant it is a zero-order model. The output level z_i of each rule is weighted by the firing strength w_i of the rule. The final output is the weighted average of all rule outputs, denoted as

$$z = \frac{\sum_{i=1}^{N} w_i z_i}{\sum_{i=1}^{N} z_i}, \tag{2.11}$$

where z_i is a 1st order polynomial.

In the Tsukamoto fuzzy model [23], the consequent of each fuzzy if-then rule is represented by a fuzzy set with a monotonic MF. The inferred output of each rule is defined as a crisp value induced by the rule's firing strength. The overall output is the weighted average of each rule's output, represented as

$$z = \frac{\sum_{i=1}^{N} w_i z_i}{\sum_{i=1}^{N} w_i}, \tag{2.12}$$

where z_i is a crisp value induced by the rule's firing strength. However, the Tsukamoto fuzzy model is not used very often; the Mamdani or Sugeno fuzzy models are vastly more popular.

2.2 ANFIS review

ANFIS [5][6] is a type of adaptive network that is functionally equivalent to a TSK fuzzy inference system. Each node represents a processing unit and the directed links indicate the flow direction of signals between the connected nodes. Some of nodes are adaptive, which means that the outputs of the nodes are dependent on modifiable parameters. By adjusting the parameters of just one single node in the first layer, changes will propagate towards further nodes; as a result, the behavior of the whole network will be changed. The links in this adaptive network are un-weighted. Based on the type of connections, adaptive networks are normally classified into two categories: feedforward (no feedback links) and recurrent (at least one directed cycle). ANFIS is a feedforward type network, having no links between nodes in the same layer.

The parameters of ANFIS are distributed into its nodes, so each node has its own set of local parameters. The union of all of the local parameters forms the adaptive network's overall parameter set. If a node's parameter set is not the empty set, then its node function can be modified through the adaptive parameters, and this kind of adaptive node is represented graphically using a square. On the other hand, if there is no parameter set pertaining to the node, this fixed node is indicated graphically using a circle (see Fig.2.6). ANFIS contains 5 different layers, where nodes in the same layer have the same node function and nodes in different layers have different node functions. Two of the layers (1 and 4) have adaptive (parameterized) nodes. The parameters in layer 1 are known as the premise parameters and the parameters in layer 4 are called the consequent parameters.

ANFIS aims to achieve a desired nonlinear mapping that is defined by a data set made up of desired input-output pairs of a target system to be modeled. The procedures used to regulate the node parameters to improve ANFIS's performance are referred to as the hybrid learning algorithm. This algorithm combines the gradient descent and least squares methods for identification of parameters.

ANFIS uses offline or "batch" learning. Each epoch consists of a forward pass and a backward pass. In the forward pass, after an input is presented, the node outputs are calculated starting from the first layer towards the right in the forward direction (see Fig.2.6) until the consequent parameters in layer 4 are identified using a least-squares method while the values of the premise parameters in layer 1 are fixed. After the parameters of the polynomial in layer 4 are identified, the error measure for each training data pair can be computed. The errors for each data pair are summed together, until all

Table 2. 3 Summary of the hybrid-learning rule used in ANFIS, from [5][6]

	Forward pass	Backward pass
Premise Parameters in layer 1	Fixed	Gradient descent
Consequent Parameters in layer 4	Least-squares estimator	Fixed
Signals	Node outputs	Error signals

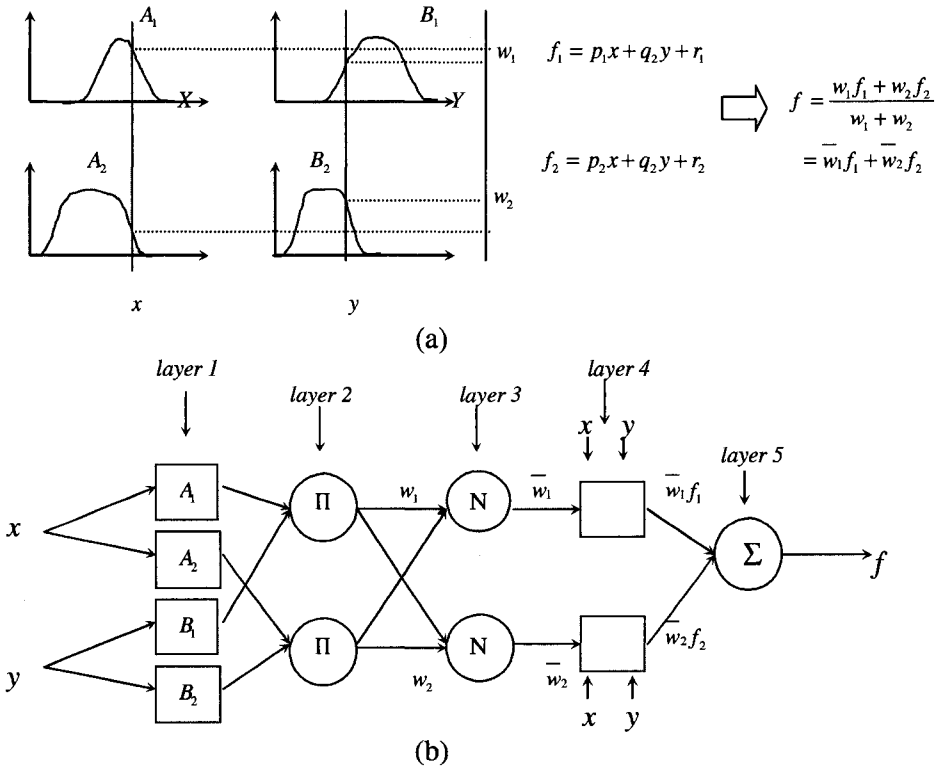


Fig. 2. 6 (a) Sugeno fuzzy reasoning (b) Equivalent ANFIS architecture, from [5][6]

training data pairs have been presented. In the backward pass, error signals propagate backward from layer 5 towards layer 1, and the premise parameters are updated using the gradient descent method while the consequent parameters are fixed. The gradient method alone is slow to converge and it is prone to get trapped in local minima. The hybrid-learning rule helps avoid this, and the hybrid algorithm also converges much faster since

the search space dimensions are reduced in comparison to the original steepest descent method. Table 2.3 gives a summary of the hybrid-learning rule.

To demonstrate the ANFIS architecture, consider a two-input 1st order Sugeno fuzzy model with 2 fuzzy if-then rules (shown in Fig. 2.6(a)), and the corresponding equivalent ANFIS (shown in Fig. 2.6(b)). Note that the diagram indicates the node connections in the forward direction in the forward pass of ANFIS. In the backward pass of ANFIS, the directional links between the nodes are reversed.

For a first-order Sugeno fuzzy model, a common rule set with two fuzzy if-then rules is the following:

If x is A_1 and y is B_1 then $f_1 = p_1 x + q_1 y + r_1$,

If x is A_2 and y is B_2 then $f_2 = p_2 x + q_2 y + r_2$.

2.2.1 ANFIS architecture

Layer 1: the node function of each node i (an adaptive node) in layer 1 is denoted by

$$O_{1,i} = \mu_{A_i}(x), \quad i = 1,2, \text{ and } O_{1,i} = \mu_{B_{i-2}}(y), i = 3,4. \tag{2.13}$$

in which x is the input to the i_{th} node, A_i and B_{i-2} are the linguistic labels (e.g. large, small) with respect to this node function. Namely, $O_{1,i}$ is the membership function grade of a fuzzy set A (where A = A_1, A_2, B_1 or B_2) and it indicates how much the given x matches the quantifier A_i. The membership function μ is any appropriate parameterized membership function such as the generalized bell function or Gaussian function (shown below):

$$\mu_A(x) = \frac{1}{1+\left[\left(\frac{x-c_i}{a_i}\right)^2\right]^{b_i}} \tag{2.14}$$

or

$$\mu_A(x) = \exp\left\{-\left(\frac{x-c_i}{a_i}\right)^2\right\} \tag{2.15}$$

Trapezoidal or triangular-shaped membership functions are also common candidates for node functions in layer 1. Parameters of the membership function in layer 1 are termed premise parameters. As discussed, in the forward pass, the premise parameters are fixed.

Layer 2: Each node labeled $\prod$ in layer 2 is a fixed node, and its output is the product of all incoming signals, and represents the firing strength of a fuzzy rule.

$$O_{2,i} = w_i = \mu_{A_i}(x)\mu_{B_i}(y), \quad i = 1,2. \tag{2.16}$$

Layer 3: Each node labeled N in layer 3 is a fixed node that normalizes each weight by the sum of all rules' firing degrees.

$$O_{3,i} = \overline{w_i} = \frac{w_i}{\sum_{j=1}^{2} w_j}, \quad i = 1,2. \tag{2.17}$$

Layer 4: Each node in layer 4 is an adaptive node with node function:

$$O_{4,i} = \overline{w_i} f_i = \overline{w_i}(p_i x + q_i y + r_i). \tag{2.18}$$

where $\bar{w}_i$ is the output of the 3rd layer, and the values p_i, q_i and r_i are called the consequent parameters. They are identified using a least squares method.

Layer 5: In this layer, there is only a single node, a fixed node labeled Σ, which calculates the overall output as the sum of all incoming signals. This is the output of ANFIS.

$$O_{5,1} = \sum_i \overline{w_i} f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}. \tag{2.19}$$

If the premise parameters are fixed in the forward pass, the overall output can be expressed as a linear combination of the consequent parameters. The output of the above network can be expressed as:

$$\begin{aligned} f &= \frac{w_1}{w_1 + w_2} f_1 + \frac{w_2}{w_1 + w_2} f_2 \\ f &= \overline{w_1}(p_1 x + q_1 y + r_1) + \overline{w_2}(p_2 x + q_2 y + r_2) \\ f &= (\overline{w_1} x) p_1 + (\overline{w_1} y) q_1 + (\overline{w_1}) r_1 + (\overline{w_2} x) p_2 + (\overline{w_2} y) q_2 + (\overline{w_2}) r_2, \end{aligned} \tag{2.20}$$

which is linear in the consequent parameter.

To speed up parameter identification of the adaptive networks and avoid convergence to local minima, here we describe a hybrid learning which integrate the gradient method with least squares estimate.

2.2.2 Gradient descent method in hybrid learning algorithm

The central part of the gradient descent [5][6] for ANFIS is to calculate the gradient vector. The gradient vector is the derivative of an error measure with respect to a parameter, which can be computed using the chain rule. This is referred to as the back-propagation learning rule since it is calculated from the output towards the inputs. The p th error measure E_p can be obtained from each training pair in our training data. The target is to minimize the total error measure

$$E = \sum_p E_p . \tag{2.21}$$

This can be realized by adjusting the parameters in the node functions. Before calculating the gradient vector, the following relationship should be observed:

Change in parameter α => change in output of nodes containing parameter α => change in network's outputs => change in error measure.

This means that the error measure can be minimized by adjusting the individual parameters of the network. In other words, a small change in a parameter α may have a high impact on the output of the node containing α. The basic concept in calculating the gradient vector is to pass a form of derivative information starting from the output layer, going backwards layer-by-layer until the input layer is reached. Note that there is an important distinction between this 'ordered derivative' and the 'partial derivative'. The partial derivative handles all of other input variables besides the one in question as constants when differentiation is performed. This is not true for the ordered derivative. The ordered derivative considers both direct and indirect paths that contribute to the casual relationship.

We assume that ANFIS has L layers, and its lth layer has $N(l)$ nodes. Then the output and function of node i $[i = 1,\ldots,N(l)]$ in lay l can be represented as $x_{l,i}$ and $f_{l,i}$, respectively. Due to the node output associated with its upcoming signals and its own parameter set, $x_{l,i}$ is written as follows:

$$x_{l,i} = f_{l,i}\left(x_{l-1,1},\cdots x_{l-1,N(l-1)},a,b,c\cdots\right), \tag{2.22}$$

where a, b, c, etc. are the parameters corresponding to this node.

Suppose that there are P entries in the given training data, the error measure for the pth($(1 \le p \le P)$ entry of training data entry can be defined as the sum of squared error, in symbols :

$$E_p = \sum_{k=1}^{N(L)}\left(d_k - x_{L,k}\right)^2, \tag{2.23}$$

where d_k is the kth component of the p_{th} target output vector, and $x_{L,k}$ is the kth component of actual output vector generated by the current p_{th} input vector. Therefore, the overall error measure is written as: $E = \sum_{p=1}^{P} E_p$.

In order to perform a learning procedure based on the gradient descent of E in the whole parameter set, firstly, we need to compute the error rate $\partial E_p / \partial x$ for the p_{th} training data and for each node output x. The error rate for the output node at (L,i) can be derived easily from Eq. (2.23).

$$\varepsilon_{L,i} = \frac{\partial^+ E_p}{\partial x_{L,i}} = \frac{\partial E_p}{\partial x_{L,i}} = -2(d_i - x_{L,i}) . \tag{2.24}$$

The error rate at the internal node (l,i) can be calculated by the chain rule:

$$\varepsilon_{l,i} = \frac{\partial^+ E_p}{\partial x_{l,i}} = \sum_{m=1}^{N(l+1)} \frac{\partial^+ E_p}{\partial x_{l+1,m}} \frac{\partial f_{l+1,m}}{\partial x_{l,i}} = \sum_{m=1}^{N(l+1)} \varepsilon_{l+1,m} \frac{\partial f_{l+1,m}}{\partial x_{l,i}}, \tag{2.25}$$

where $1 \le l \le L-1$. That means the error rate in the internal node can be denoted as a linear sum of the error rates of the nodes on the next layer.

The gradient vector is defined as the derivative of the error measure with respect to each parameter, so we have to apply the chain rule again to find the gradient vector. If α is a parameter of the i th node at layer l, we have

$$\frac{\partial^+ E_p}{\partial \alpha} = \frac{\partial^+ E_p}{\partial x_{l,i}} \frac{\partial f_{l,i}}{\partial \alpha} = \varepsilon_{l,i} \frac{\partial f_{l,i}}{\partial \alpha}, \tag{2.26}$$

The derivative of the overall error measure E with respect to α:

$$\frac{\partial^+ E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial^+ E_p}{\partial \alpha}, \tag{2.27}$$

at the same time, the generic parameter α can be updated below:

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha} , \tag{2.28}$$

where η is a learning rate that can be written as:

$$\eta = \frac{k}{\sqrt{\sum_{\alpha} \left(\frac{\partial E}{\partial \alpha}\right)^2}} \tag{2.29}$$

in which k is a step size, which can affect the convergence rate.

As a matter of fact, ANFIS use the batch learning mode(also called as off-line learning), to update the parameter α based on the formula (2.27) and the update action is performed only after the whole training data set has been offered, i.e., after each epoch.

The following example illustrates how to map these generic equations down to the specific equations of ANFIS, (consider a two-input 1st order Sugeno fuzzy model with 2 fuzzy if-then rules, with generalized bell membership functions). The error-propagation network of Fig. 2.6 (b) is constructed in Fig.2.7, where the output of node i is the error signal of this node in the original adaptive network. Then we have the following:

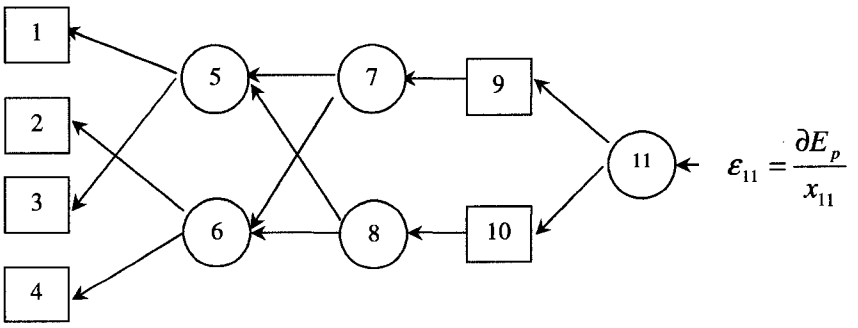


Fig. 2. 7 Error propagation model of Fig.2.6 (b)

$$\varepsilon_{11}=\frac{\partial E_p}{\partial x_{11}}=-2(d_{11}-x_{11})=-2\left(d_{11}-\sum_i \overline{w}_i f_i\right), i=1,2,$$

$$\varepsilon_{10}=\frac{\partial E_p}{\partial x_{10}}=\frac{\partial E_p}{\partial x_{11}}\frac{\partial f_{11}}{\partial x_{10}}=\varepsilon_{11}\frac{\partial f_{11}}{\partial x_{10}}=\varepsilon_{11}\frac{\partial\left(\sum_i \overline{w}_i f_i\right)}{\partial\left(\overline{w}_2 f_2\right)}=\varepsilon_{11}, i=1,2,$$

$$\varepsilon_{9}=\frac{\partial E_p}{\partial x_{9}}=\frac{\partial E_p}{\partial x_{11}}\frac{\partial f_{11}}{\partial x_{9}}=\varepsilon_{11}\frac{\partial f_{11}}{\partial x_{9}}=\varepsilon_{11}\frac{\partial\left(\sum_i \overline{w}_i f_i\right)}{\partial\left(\overline{w}_1 f_1\right)}=\varepsilon_{11}, i=1,2,$$

$$\varepsilon_{8}=\frac{\partial E_p}{\partial x_{8}}=\frac{\partial E_p}{\partial x_{10}}\frac{\partial f_{10}}{\partial x_{8}}=\varepsilon_{10}\frac{\partial f_{10}}{\partial x_{8}}=\varepsilon_{10}\frac{\partial\left(\overline{w}_2 f_2\right)}{\partial\left(\overline{w}_2\right)}=\varepsilon_{10} f_2=\varepsilon_{11} f_2,$$

$$\varepsilon_{7}=\frac{\partial E_p}{\partial x_{7}}=\frac{\partial E_p}{\partial x_{9}}\frac{\partial f_{9}}{\partial x_{7}}=\varepsilon_{9}\frac{\partial f_{9}}{\partial x_{7}}=\varepsilon_{9}\frac{\partial\left(\overline{w}_1 f_1\right)}{\partial\left(\overline{w}_1\right)}=\varepsilon_{9} f_1=\varepsilon_{11} f_1,$$

$$\varepsilon_{6}=\frac{\partial E_p}{\partial x_{6}}=\frac{\partial E_p}{\partial x_{8}}\frac{\partial f_{8}}{\partial x_{6}}+\frac{\partial E_p}{\partial x_{7}}\frac{\partial f_{7}}{\partial x_{6}}=\varepsilon_{8}\frac{\partial f_{8}}{\partial x_{6}}+\varepsilon_{7}\frac{\partial f_{7}}{\partial x_{6}}=\varepsilon_{8}\frac{\partial\left(\frac{w_2}{w_1+w_2}\right)}{\partial(w_2)}+\varepsilon_{7}\frac{\partial\left(\frac{w_1}{w_1+w_2}\right)}{\partial(w_2)}$$

$$=\varepsilon_{8}\frac{w_1}{(w_1+w_2)^2}+\varepsilon_{7}\frac{-w_1}{(w_1+w_2)^2}=\varepsilon_{11}\frac{w_1 f_2-w_1 f_1}{(w_1+w_2)^2},$$

$$\varepsilon_{5}=\frac{\partial E_p}{\partial x_{5}}=\frac{\partial E_p}{\partial x_{8}}\frac{\partial f_{8}}{\partial x_{5}}+\frac{\partial E_p}{\partial x_{7}}\frac{\partial f_{7}}{\partial x_{5}}=\varepsilon_{8}\frac{\partial f_{8}}{\partial x_{5}}+\varepsilon_{7}\frac{\partial f_{7}}{\partial x_{5}}=\varepsilon_{8}\frac{\partial\left(\frac{w_2}{w_1+w_2}\right)}{\partial(w_1)}+\varepsilon_{7}\frac{\partial\left(\frac{w_1}{w_1+w_2}\right)}{\partial(w_1)}$$

$$=\varepsilon_{8}\frac{-w_2}{(w_1+w_2)^2}+\varepsilon_{7}\frac{w_2}{(w_1+w_2)^2}=\varepsilon_{11}\frac{w_2 f_1-w_2 f_2}{(w_1+w_2)^2},$$

$$\varepsilon_{4}=\frac{\partial E_p}{\partial x_{4}}=\frac{\partial E_p}{\partial x_{6}}\frac{\partial f_{6}}{\partial x_{4}}=\varepsilon_{6}\frac{\partial f_{6}}{\partial x_{4}}=\varepsilon_{6}\frac{\partial\left(\mu_{A_2}\mu_{B_2}\right)}{\partial\left(\mu_{B_2}\right)}=\varepsilon_{6}\mu_{A_2},$$

$$\varepsilon_{3}=\frac{\partial E_p}{\partial x_{3}}=\frac{\partial E_p}{\partial x_{5}}\frac{\partial f_{5}}{\partial x_{3}}=\varepsilon_{5}\frac{\partial f_{5}}{\partial x_{3}}=\varepsilon_{5}\frac{\partial\left(\mu_{A_1}\mu_{B_1}\right)}{\partial\left(\mu_{B_1}\right)}=\varepsilon_{5}\mu_{A_1},$$

$$\varepsilon_{2}=\frac{\partial E_p}{\partial x_{2}}=\frac{\partial E_p}{\partial x_{6}}\frac{\partial f_{6}}{\partial x_{2}}=\varepsilon_{6}\frac{\partial f_{6}}{\partial x_{2}}=\varepsilon_{6}\frac{\partial\left(\mu_{A_2}\mu_{B_2}\right)}{\partial\left(\mu_{A_2}\right)}=\varepsilon_{6}\mu_{B_2},$$

$$\varepsilon_{1}=\frac{\partial E_p}{\partial x_{1}}=\frac{\partial E_p}{\partial x_{5}}\frac{\partial f_{5}}{\partial x_{1}}=\varepsilon_{5}\frac{\partial f_{5}}{\partial x_{1}}=\varepsilon_{5}\frac{\partial\left(\mu_{A_1}\mu_{B_1}\right)}{\partial\left(\mu_{A_1}\right)}=\varepsilon_{5}\mu_{B_1},$$

For the bell MF, $\mu_A(x)=\dfrac{1}{1+\left[\left(\frac{x-c_i}{a_i}\right)^2\right]^{b_i}}$ then the derivatives of an MF with respect to parameters are listed as follows:

$$\frac{\partial \mu_A}{\partial a_i} = \frac{2b_i}{a_i}\mu_A(1-\mu_A),$$

$$\frac{\partial \mu_A}{\partial b_i} = \begin{cases} 2\ln\left|\frac{x-c_i}{a_i}\right|\mu_A(1-\mu_A), & \text{if } x \neq c_i. \\ 0, & \text{if } x = c_i. \end{cases}$$

$$\frac{\partial \mu_A}{\partial c_i} = \begin{cases} \frac{2b_i}{x-c_i}\mu_A(1-\mu_A), & \text{if } x \neq c_i. \\ 0, & \text{if } x = c_i. \end{cases} \tag{2.30}$$

The gradient for the parameters a_1, b_1, c_1 in node 1, the parameters a_2, b_2, c_2 in node 2, the parameters a_3, b_3, c_3 in node 3, and the parameters a_4, b_4, c_4 in node 4 of the layer 1 can be obtained directly according to Eq. (2.30). Thus, for the node 1, we have

$$\frac{\partial E_p}{\partial a_1} = \frac{\partial E_p}{\partial x_1}\frac{\partial f_1}{\partial a_1} = \varepsilon_1 \frac{\partial f_1}{\partial a_1} = \varepsilon_1 \frac{\partial(\mu_{A_1})}{\partial(a_1)} = \varepsilon_1 \frac{2b_1}{a_1}\mu_{A_1}(1-\mu_{A_1}),$$

$$\frac{\partial E_p}{\partial b_1} = \frac{\partial E_p}{\partial x_1}\frac{\partial f_1}{\partial b_1} = \varepsilon_1 \frac{\partial f_1}{\partial b_1} = \varepsilon_1 \frac{\partial(\mu_{A_1})}{\partial(b_1)} = \begin{cases} 2\varepsilon_1 \ln\left|\frac{x-c_1}{a_1}\right|\mu_{A_1}(1-\mu_{A_1}), & \text{if } x \neq c_1. \\ 0, & \text{if } x = c_1. \end{cases}$$

$$\frac{\partial E_p}{\partial c_1} = \frac{\partial E_p}{\partial x_1}\frac{\partial f_1}{\partial c_1} = \varepsilon_1 \frac{\partial f_1}{\partial c_1} = \varepsilon_1 \frac{\partial(\mu_{A_1})}{\partial(c_1)} = \begin{cases} \frac{2b_1}{x-c_1}\varepsilon_1\mu_{A_1}(1-\mu_{A_1}), & \text{if } x \neq c_1. \\ 0, & \text{if } x = c_1. \end{cases}$$

For the node 2, we have

$$\frac{\partial E_p}{\partial a_2} = \frac{\partial E_p}{\partial x_2}\frac{\partial f_2}{\partial a_2} = \varepsilon_2 \frac{\partial f_2}{\partial a_2} = \varepsilon_2 \frac{\partial(\mu_{A_2})}{\partial(a_2)} = \varepsilon_2 \frac{2b_1}{a_1}\mu_{A_2}(1-\mu_{A_2}),$$

$$\frac{\partial E_p}{\partial b_2} = \frac{\partial E_p}{\partial x_2}\frac{\partial f_2}{\partial b_2} = \varepsilon_2 \frac{\partial f_2}{\partial b_2} = \varepsilon_2 \frac{\partial(\mu_{A_2})}{\partial(b_2)} = \begin{cases} 2\varepsilon_1 \ln\left|\frac{x-c_2}{a_2}\right|\mu_{A_2}(1-\mu_{A_2}), & \text{if } x \neq c_2. \\ 0, & \text{if } x = c_2. \end{cases}$$

$$\frac{\partial E_p}{\partial c_2} = \frac{\partial E_p}{\partial x_2}\frac{\partial f_2}{\partial c_2} = \varepsilon_2 \frac{\partial f_2}{\partial c_2} = \varepsilon_2 \frac{\partial(\mu_{A_2})}{\partial(c_2)} = \begin{cases} \frac{2b_2}{x-c_2}\varepsilon_2\mu_{A_2}(1-\mu_{A_2}), & \text{if } x \neq c_2. \\ 0, & \text{if } x = c_2. \end{cases}$$

For the node 3, we have

$$\frac{\partial E_p}{\partial a_3} = \frac{\partial E_p}{\partial x_3}\frac{\partial f_3}{\partial a_3} = \varepsilon_3 \frac{\partial f_3}{\partial a_3} = \varepsilon_3 \frac{\partial(\mu_{B_1})}{\partial(a_3)} = \varepsilon_3 \frac{2b_3}{a_3}\mu_{B_1}(1-\mu_{B_1}),$$

$$\frac{\partial E_p}{\partial b_3} = \frac{\partial E_p}{\partial x_3}\frac{\partial f_3}{\partial b_3} = \varepsilon_3 \frac{\partial f_3}{\partial b_3} = \varepsilon_3 \frac{\partial(\mu_{B_1})}{\partial(b_3)} = \begin{cases} 2\varepsilon_3 \ln\left|\frac{y-c_3}{a_3}\right|\mu_{B_1}(1-\mu_{B_1}), & \text{if } y \neq c_3. \\ 0, & \text{if } y = c_3. \end{cases}$$

$$\frac{\partial E_p}{\partial c_3}=\frac{\partial E_p}{\partial x_3}\frac{\partial f_3}{\partial c_3}=\varepsilon_3\frac{\partial f_3}{\partial c_3}=\varepsilon_3\frac{\partial(\mu_{B_1})}{\partial(c_3)}=\begin{cases}\dfrac{2b_3}{y-c_3}\varepsilon_3\mu_{B_1}(1-\mu_{B_1}), & \text{if } y\neq c_3.\\ 0, & \text{if } y=c_3.\end{cases}$$

For the node 4, we have

$$\frac{\partial E_p}{\partial a_4}=\frac{\partial E_p}{\partial x_4}\frac{\partial f_4}{\partial a_4}=\varepsilon_4\frac{\partial f_4}{\partial a_4}=\varepsilon_4\frac{\partial(\mu_{B_2})}{\partial(a_4)}=\varepsilon_4\frac{2b_4}{a_4}\mu_{B_2}(1-\mu_{B_2}),$$

$$\frac{\partial E_p}{\partial b_4}=\frac{\partial E_p}{\partial x_4}\frac{\partial f_4}{\partial b_4}=\varepsilon_4\frac{\partial f_4}{\partial b_4}=\varepsilon_4\frac{\partial(\mu_{B_2})}{\partial(b_4)}=\begin{cases}2\varepsilon_4\ln\left|\dfrac{y-c_4}{a_4}\right|\mu_{B_2}(1-\mu_{B_2}), & \text{if } y\neq c_4.\\ 0, & \text{if } y=c_4.\end{cases}$$

$$\frac{\partial E_p}{\partial c_4}=\frac{\partial E_p}{\partial x_4}\frac{\partial f_4}{\partial c_4}=\varepsilon_4\frac{\partial f_4}{\partial c_4}=\varepsilon_4\frac{\partial(\mu_{B_2})}{\partial(c_4)}=\begin{cases}\dfrac{2b_4}{y-c_4}\varepsilon_4\mu_{B_2}(1-\mu_{B_2}), & \text{if } y\neq c_4.\\ 0, & \text{if } y=c_4.\end{cases}$$

And then, the adaptive parameters can be updated using Eqs. (2.27), (2.28), (2.29).

2.3 Complex fuzzy theory review

When fuzzy sets were proposed in 1965 [12], a lot of researchers criticized and thought they had no practical value. Starting from that time, the application field of fuzzy logic has grown tremendously. The unique properties of fuzzy logic are capable of expressing subjective or linguistic knowledge so that fuzzy logic is mainly applied to simulate human decision making process. Mathematically, fuzzy sets are an extension of dual valued logic, extending the possible values of the membership function from {0, 1} to the unit interval [0, 1]. One of the most famous early applications of fuzzy logic is that of the Sendai Subway system in Sendai, Japan. This control of the Nanboku line, developed by Hitachi, used a fuzzy controller to run the train all day long. The fuzzy system controls acceleration, deceleration, and breaking of the train. Since its introduction, it not only reduced energy consumption by 10%, but the passengers hardly notice now when the train is changing its velocity. In the past neither conventional, nor human control could have achieved such performance [100]. A type-2 fuzzy set maps elements in a crisp domain to type-1 fuzzy numbers ranged from 0 to1. Since the value of

each point in a type-2 fuzzy set is given as a function, type-2 fuzzy sets are three-dimensional. After combining traditional fuzzy set with complex numbers, a novel complex fuzzy set characterized by complex-valued membership function is developed. This idea has resulted in the development of complex fuzzy sets [2] and complex fuzzy logic [1][3]. As suggested in [1], the complex fuzzy sets might be an appropriate model for "approximately periodic" phenomena since the complex fuzzy sets might be characterized by the sine function presented as a possible candidate in [1]. One of examples of these "approximately periodic" patterns given in [1] is that of traffic congestion happened in the major city. In the morning, there is heavy traffic in one direction while working people are rushed to work. The same thing occurs in the opposite direction when they go home from work. In between, traffic is lighter, and at night, the roads are nearly empty. Although this behaviour repeats from one day to the next, it is only approximately periodic since it never repeats itself exactly.

Complex fuzzy sets are different from the fuzzy complex numbers developed by Buckley in [25], [26] and [27]. A fuzzy complex number as developed firstly in [25] is a type-1 fuzzy set, whose members are elements of the complex plane (i.e. a fuzzy subset of the complex numbers). The differentiation and integration of fuzzy complex numbers was discussed in [26] and [27], respectively. Fuzzy complex numbers were employed in the solution of fuzzy relational equations in [28] and [29]. In [102], a complex fuzzy set was defined as a membership function mapping the complex plane into $[0,1]\times[0,1]$; this is very close to the complex fuzzy sets in [2], and quite similar to the formulation in [103]. Nguyen *et al.* investigated this literature and applied an optimization method to choose the best representation for fuzzy complex numbers in [104].

2.3.1 Complex fuzzy sets

A complex fuzzy set S characterized by a membership function $\mu_S(x)$ is defined on a universe of discourse U, in which any element belonging to U is mapped to a complex-valued grade of membership in the set S. The values from $\mu_S(x)$ correspond to the points in the unit disc in the complex plane and are represented in the form of

$$\mu_S(x) = r_S(x) \cdot e^{j\omega_S(x)}, \qquad j = \sqrt{-1}. \tag{2.31}$$

where $r_s(x)$ and $\omega_s(x)$ are both real valued and $r_s(x) \in [0,1]$. The complex fuzzy set S can be represented as the set of ordered pairs

$$S = \{(x, \mu_s(x)) \mid x \in U\}. \tag{2.32}$$

From equation (2.31), it is obvious that complex grades of membership consist of an amplitude term $r_S(x)$ and a phase term $\omega_S(x)$. Since complex fuzzy sets are extensions of ordinary fuzzy sets, it should be possible to express any ordinary complex set in complex form by setting the amplitude term and the phase term $\omega_s(x) = 0$ for all x. Without considering the phase term, the complex fuzzy set is converted into a traditional fuzzy set.

From the viewpoint of the amplitude term, it is apparent that the amplitude term is the same as the real-valued grade of membership, lying in the range of [0,1], and it may be regarded as representing the degree to which x is a member of the complex fuzzy set S. On the contrary, the phase term is a totally innovative parameter of membership, by which the complex fuzzy sets are distinguished from the ordinary fuzzy sets, and it is the phase term that supplies the framework of complex fuzzy logic with its unique properties and makes complex fuzzy logic differ from the conventional fuzzy logic. The

phase term allows for a new type of interaction to occur between membership functions. Complex fuzzy membership functions can be viewed as a wave, where the phase term allows them to constructively or destructively interfere with one another using a novel form of set aggregation termed vector aggregation [3].

2.3.2 Set theoretic operations

The derivation of complex fuzzy logic is basis on several mathematical properties of complex fuzzy sets, which include basic set theoretic operations such as complex fuzzy union and intersection, complex fuzzy relations and their composition, and a novel form of set aggregation-vector aggregation. The original definitions in [2] are described below.

Complex fuzzy complement:

The traditional fuzzy complement function c must satisfy the two following axioms:

1. $c(0) = 1$, and $c(1) = 0$ (boundary conditions)
2. $\forall x, y \in [0,1]$, if $x \leq y$, then $c(x) \geq c(y)$ (monotonicity)

It is also desirable for c to be continuous and involutive, $c(c(x)) = x$. There are three popular complement operations, the standard fuzzy complement, the step threshold complement and the Sugeno class complement.

Complex fuzzy union:

The set of axioms of traditional fuzzy set theoretic operations could not directly transplanted to that of complex fuzzy sets due to no closure for complex fuzzy sets while using the algebraic sum. However, the amplitude term of a complex-valued grade of membership is actually commensurate to the conventional real-valued grade of membership. Therefore, any axiomatic requirements applied to conventional fuzzy union are supposed to be applicable to the

amplitude term of complex fuzzy union, in other words, similar to the complex fuzzy complement, the complex fuzzy union u should apply traditional fuzzy unions to the amplitude term.

For $\mu_A(x) = r_A(x) \cdot e^{j\omega_A(x)}$ and $\mu_B(x) = r_B(x) \cdot e^{j\omega_B(x)}$, the membership union $A \cup B$ is represented as:

$$\mu_{A \cup B}(x) = [r_A(x) \times r_B(x)] \cdot e^{j\omega_{A \cup B}(x)}, \tag{2.33}$$

Now, the issue to resolve is how to obtain $\omega_{A \cup B}$. Here, we could not simply abandon the union operator for the phase term as we did in the complement operator. In the process of using the complex fuzzy union, the form of deriving the phase term of $A \bigcup B$ is dependent on the practical applications. That means, various applications of complex fuzzy union may need completely different form to calculate the phase term $\omega_{A \cup B}$. The authors [2][3] present several different possibilities in the paper:

a) Sum: $\omega_{A \cup B} = \omega_A + \omega_B$ (2.34)

b) Max: $\omega_{A \cup B} = \max(\omega_A, \omega_B)$ (2.35)

c) Min: $\omega_{A \cup B} = \min(\omega_A, \omega_B)$ (2.36)

d) Winner take all: $\omega_{A \cup B} = \omega_A$ *if* $r_A > r_B$ or $\omega_{A \cup B} = \omega_B$ *if* $r_B > r_A$ (2.37)

e) Weighted average: $\omega_{A \cup B} = \dfrac{r_A \omega_A + r_B \omega_B}{r_A + r_B}$ (2.38)

f) Average: $\omega_{A \cup B} = \dfrac{\omega_A + \omega_B}{2}$ (2.39)

g) Difference: $\omega_{A \cup B} = \omega_A - \omega_B$ (2.40)

Complex fuzzy intersection

The operation of complex fuzzy intersection is almost identical to that of complex fuzzy union. The traditional fuzzy axioms are used directly in the derivation of the amplitude terms, while treatment of the membership phase $\omega_{A\cap B}$ is almost completely application dependant. The operation of complex fuzzy intersection can be specified by:

$$\mu_{A\cap B}(x)=[r_A(x)\times r_B(x)]\cdot e^{j\omega_{A\cap B}(x)}, \tag{2.41}$$

and $\omega_{A\cap B}$ is application dependant and can be chosen from the same list above for the union (see Eqs. (2.34)~(2.40)) .

From the set operations above, the technique employed for defining the complex fuzzy complement, union and intersection was to treat the magnitude and phase of a complex number as separate entities; these operations are defined only based on the amplitude and the phase term is regarded as an entirely relative quantity, and it has a lower impact on the set theoretic operations. This idea was tested and formalized as rotational invariance in [1]. It was demonstrated that even under the more general criteria of rotational invariance, entirely disregarding the phase may limit the possible choices of the set-theoretic operations. More specifically, it was demonstrated that intuitively appealing choices for the complex fuzzy complement (negation) and the complex fuzzy intersection (the algebraic product) are exclude from consideration. Instead, the author in [1] suggested that the amplitude and phase of membership function should be coupled tightly in complex fuzzy set. Sine and cosine functions were also suggested as basic complex fuzzy membership functions.

2.3.3 Vector aggregation

Conventional aggregation operations on fuzzy sets provide a tool for incorporating several sets in order to generate a single fuzzy set, while vector aggregation

[2] provides a specially useful methods for performing complex fuzzy sets, given as follows:

Let $A_1, A_2, \cdots, A_n$ be complex fuzzy sets on U, vector aggregation on $A_1, A_2, \cdots, A_n$ is specified by a function v

$$v : \{a | a \in C, |a| \le 1\}^n \rightarrow \{b | b \in C, |b| \le 1\} \tag{2.42}$$

The function v generates an aggregation fuzzy set A by manipulating complex fuzzy sets $A_1, A_2, \cdots, A_n$ for each $x \in U$. For all $x \in U$, v may be represented by

$$\mu_A(x) = v(\mu_{A_1}(x), \mu_{A_2}(x), \cdots, \mu_{A_n}(x)) = \sum_{i=1}^{n} w_i \mu_{A_i}(x) \tag{2.43}$$

with $w_i \in \{a | a \in C, |a| \le 1\}$ for all i, and $\sum_{i=1}^{n} |w_i| = 1$.

From Eq. (2.43), vector aggregation generates a weighted sum of the grades of membership of x in sets $A_1, A_2, \cdots, A_n$. If the phases of the different grades of membership $\mu_{A_i}(x)$ are the same, the amplitude of the sum will be maximized. Otherwise, destructive inference will happen, in this case, the amplitude of the sum will become much smaller than the separate amplitudes of its arguments. Vector aggregation allows the aggregation of complex fuzzy sets in a manner that incorporates phase considerations.

2.3.4 Complex fuzzy relations

Complex fuzzy relations [2][3] and compositions affords the basis for the derivation of the complex fuzzy logic, they are summarized below. Let U and V to be two crisp sets. A complex fuzzy relation $R(U,V)$ is a complex fuzzy subset of the product space $U \times V$. As usual, $R(U,V)$ can be denoted as the set of ordered pairs

$$R(U,V) = \{((x,y), \mu_R(x,y)) | (x,y) \in U \times V\} \tag{2.44}$$

The complex-valued membership function $u_R(x)$ can be represented in the form of $r(x)\cdot e^{j\omega(x)}$, $j=\sqrt{-1}$, $r(x)$ is the amplitude of the grade of membership with $r(x)\in[0,1]$, and $\omega(x)$ is the phase of the grade of membership, both of which are real-valued.

Compositions of complex fuzzy relations on the same product space

The compositions of complex fuzzy relations on the same product space can be manipulated based on the complex fuzzy set theoretic operations. Let $R(U,V)$ and $S(U,V)$ be complex fuzzy relations, whose compositions are operated as:

a) The complex fuzzy intersection of $R(U,V)$ and $S(U,V)$, represented by $R\cap S(U,V)$, is characterized by the membership function

$$\mu_{R\cap S}(x,y)=\mu_R(x,y)*\mu_S(x,y) \tag{2.45}$$

b) The complex fuzzy union of $R(U,V)$ and $S(U,V)$, represented by $R\cup S(U,V)$, is characterized by the membership function

$$\mu_{R\cup S}(x,y)=\mu_R(x,y)\oplus\mu_S(x,y) \tag{2.46}$$

where $*$ and $\oplus$ refers to the set theoretic operations of complex fuzzy intersection and union.

Compositions of complex fuzzy relations on different product spaces

Let the membership functions of relations $R(U,V)$, $S(V,W)$ and the composition of these relations $R\circ S(U,W)$, be

$$u_R(x,y)=r_R(x,y)\cdot e^{j\omega_R(x,y)} \tag{2.47}$$

$$u_S(y,z)=r_S(y,z)\cdot e^{j\omega_S(y,z)} \tag{2.48}$$

$$u_{R\circ S}(x,z)=r_{R\circ S}(x,z)\cdot e^{j\omega_{R\circ S}(x,z)} \tag{2.49}$$

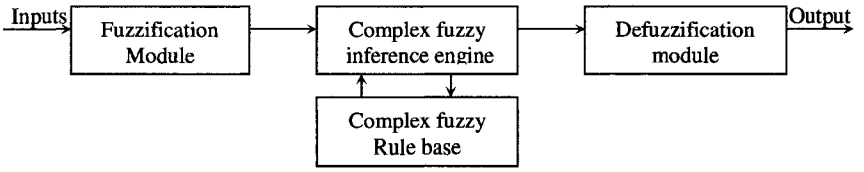


Fig. 2. 8 General scheme of a complex fuzzy logic system [3]

The amplitude term may be computed by using the sup-star composition and the phase term can be calculated using Eqs. (2.34)~(2.40) and the sup operation.

2.3.5 Complex fuzzy logic

Complex fuzzy logic [3] uses rules constructed with complex fuzzy sets to develop a complex fuzzy logic system shown in Fig. 2.8. As can be seen, a complex fuzzy system has the same architecture as a type-1 fuzzy system, but substitutes complex fuzzy sets for type-1 fuzzy sets.

The complex fuzzy implication relation is characterized by a complex-valued membership function and is expressed as $\mu_{A\to B}(x, y)$. This membership function contains an amplitude term $r_{A\to B}(x, y)$ and a phase term $\omega_{A\to B}(x, y)$. The amplitude term is equivalent to a real-valued grade of membership, and specifies the degree of truth of the implication relation. The phase term signifies the phase associated with the implication. Although the phase term is of little consequence by itself, it becomes important parameter where several implication relations are combined at the same time, as occurs in complex fuzzy systems.

The implication function employed in complex fuzzy logic is the product implication

$$\mu_{A\to B}(x, y) = \mu_A(x) \cdot \mu_B(y). \tag{2.50}$$

The amplitude and phase term are calculated using

$$r_{A\to B}(x,y) = r_A(x)\cdot r_B(y), \tag{2.51}$$

and

$$\omega_{A\to B}(x,y) = \omega_A(x) + \omega_B(y). \tag{2.52}$$

Complex fuzzy implication can be used to construct complex fuzzy inference rules, in the form of Generalized Modus Ponens.

Premise 1: " X is A^*";

Premise 2: "IF X is A, THEN Y is B".

Consequence: "Y is B^*."

where the sets A, B, A^* and B^* are all complex fuzzy sets.

The amplitude and phase term of the membership function of B^* are given by:

$$r_{B^*}(y) = \sup[r_{A^*}(x) \otimes r_{A\to B}(x,y)] = \sup[r_{A^*}(x) \otimes (r_A(x)\cdot r_B(y))] \tag{2.53}$$

$$\omega_{B^*}(y) = f[g(\omega_{A^*}(x), \omega_{A\to B}(x,y))] = f[g(\omega_A(x), (\omega_A(x) + \omega_B(y)))] \tag{2.54}$$

where g refers to any function utilized to compute the intersection of two membership phases, see Eqs. (2.34)~(2.40), and f is the membership phase equivalent of the sup operation.

Throughout section 2.3, we need to clearly indicate that these are Ramot's ideas [2][3], not ours. Our ideas often differ, where the magnitudes and phases of complex fuzzy sets are coupled tightly.

2.4 Chaotic Simulated Annealing Algorithm

Several chaotic simulated annealing algorithms solving unconstrained nonlinear problems have recently drawn much attention [11],[30]-[41]. In simulated annealing (SA)[42]-[48], the value of an objective to be minimized treated like heat energy in a

thermodynamic system. At high temperatures, SA allows function evaluations at faraway points and it is likely to accept a new point with higher energy. The disadvantage of SA lies in the slowly falling temperature process and dependence on the random number generator. Chaotic dynamical systems have a specific property, sensitivity to initial conditions, that means that each point in such a system is arbitrarily closely approximated by other points with significantly different future trajectories [49]-[51]. Thus, an arbitrarily small pertubation of the current trajectory of the chaotic map may result in significantly different future behavior, which can be applied to SA to allow algorithm to jump out of the local convergence. A transiently chaotic neural network (TCNN) with richer and more flexible dynamics [30][32] was proposed to solve combinatorial optimization problems by introducing simulated annealing to Hopfield neural network (HNN) [52][53]. Nonlinear dynamics approach [31] [54] used an additive chaotic force for determining the global minimum of a continuous, unconstrained or bound-constrained cost function. Chaotic simulated annealing (CSA) algorithm [11] introduced the concepts of chaotic initialization and chaotic sequences to SA.

Compared with traditional methods for optimization, hybridizing chaotic maps with derivative-free optimization algorithms is in its infancy, and so we will begin with a review of chaotic maps. In mathematics and physics, many one–dimensional maps exhibit sensitivity to initial conditions, with small changes in initial conditions leading to large changes in the long-term outcome (popularly known as the "butterfly effect"). Due to this sensitivity, the behavior of nonlinear chaotic maps seems to be random, as any measurement error in initial conditions are magnified exponentially through time.

However, these maps are deterministic, analytical functions (hence the term "deterministic chaos") [49].

2.4.1 The Logistic Map and Ulam-von Neumann Map

The Logistic differential equation in [50] was originally used by P.F. Verhulst in 1845 to model the surplus growth of the population biomass of species in a limited environment, such as food supply or disease, in which two casual effects are found: reproduction and starvation. The former denotes the increase of population which is proportional to the current population and the latter denotes a population decrease at a rate proportional to the value obtained by taking the theoretical "carrying capacity" of the environment less the current population. This differential equation is represented by

$$dx/dt = \mu x(1-x), \tag{2.55}$$

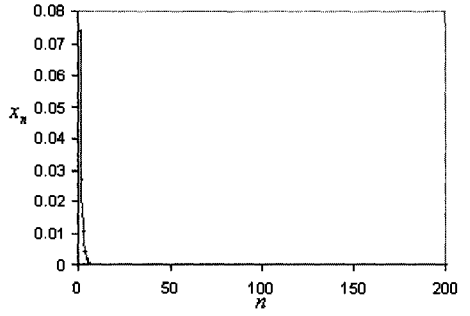
and its corresponding difference equation, the logistic map, is written as

$$x_{n+1} = \mu x_n(1-x_n),\ x_n \in [0,1], \tag{2.56}$$

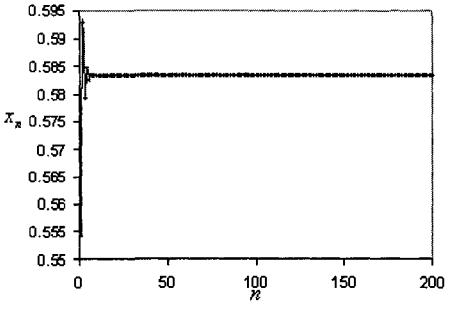
where x_n is a number between zero and one, and means the population at year n, and hence x_0 represents the initial population (at year 0) ; μ is a positive number, known as bifurcation parameter, and denotes a combined rate for reproduction and starvation. The probability density function of the Logistic map is expressed as

$$p_f(x) = \frac{1}{\pi\sqrt{x(1-x)}}. \tag{2.57}$$

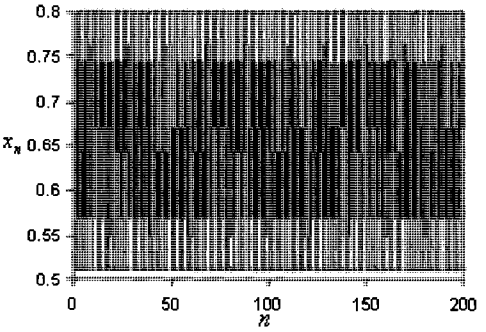
In Fig.2.9 (a)-(f), we present several runs of the logistic map for different values of μ. The map is iterated 200 times with the value of μ varying from 0.4 to 4 at different initial values generated by a random number generator.



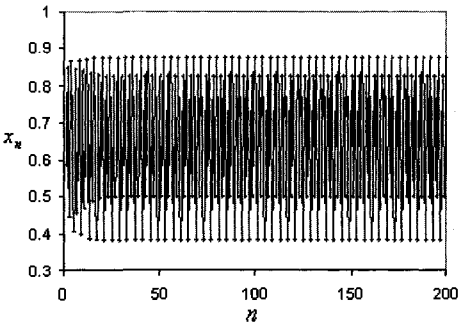
(a) Stable, at $\mu = 0.4, x = 0.242645$



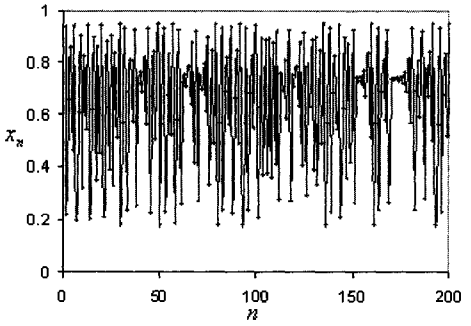
(b) Stable, at $\mu = 2.4, x = 0.362045$



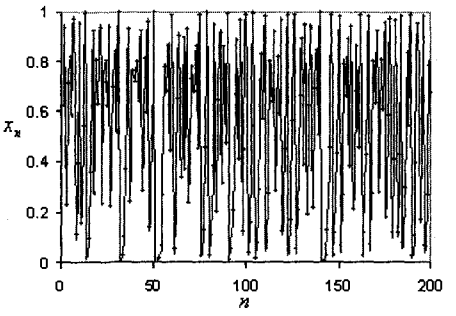
(c) Periodic, at $\mu = 3.2, x = 0.490295$



(d) Periodic, at $\mu = 3.5, x = 0.787796$



(e) chaotic, at $\mu = 3.8, x = 0.0.556549$



(f) Completely chaotic, at $\mu = 4.0, x = 0.806564$

Fig. 2. 9 The logistic map for different values of μ

Rather than continuing to describe the behavior of the logistic map for individual values of μ, we present a more global view of this chaotic model through a bifurcation diagram (Fig. 2.10), where μ varies smoothly from 2.0 to 4.0. Mathematically, a bifurcation occurs when the solution to a parameterized equation changes from being unique to there being two distinct solutions, due to a smooth change in the parameter. In

this graph, the map is iterated 200 times at each of many intervening values of μ, with the first 100 values dropped out to ensure that only the long-term behavior is plotted [50].

In Fig. 2.10, the first bifurcation occurs at $\mu = 3$. If a value of μ greater than 3 is chosen, the chaotic map becomes unstable, with higher values of μ leading to further bifurcations or even to chaotic behavior. The logistic map is known to be chaotic for most values of μ between about 3.57 and 4.

In addition, this map also displays a great sensitivity to initial conditions. As shown in Fig. 2.11, we choose $\mu = 4.0$, and select two very close initial values $x_{1,0} = 0.654321$, $x_{2,0} = 0.654320$. The map is then iterated 200 hundred times to generate two sequences, $\{x_{1,n}\}$ and $\{x_{2,n}\}, n = 1,2,\cdots,200$. We calculate the difference of these two sequences $\{x_{1,n} - x_{2,n}\}$, and plot that difference in Fig.2.11. This figure demonstrates how

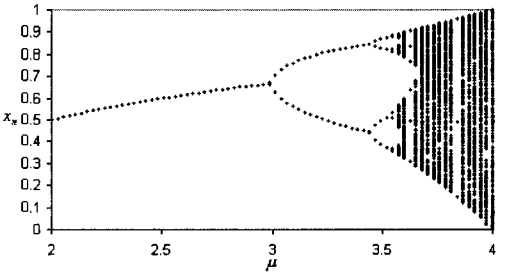


Fig. 2. 10 Bifurcation diagram of the logistic map, long-term values of x_n are plotted for $2 \le \mu \le 4$ [50].

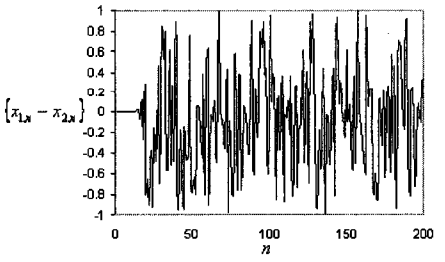


Fig. 2. 11 The sensitiveness of the logistic map to the initial condition at $\mu = 4.0$

tiny differences in initial conditions will be magnified through time in a chaotic map, until they cover the entire codomain of the map.

From Figs. 2.10 and 2.11, it would seem that at $\mu = 4.0$, the long-term behavior of the logistic map will fill much of the interval [0,1]; this is true, but with a caveat: the initial values must not be *fixed points*, which occur when $x_{n+k} = x_n$, where k denotes the order of the fixed point. The fixed points of order 1 are the solutions to the polynomial

$$f(x) = 4x(1-x) = x. \tag{2.58}$$

So the fixed points are $x_1^{(1)} = 0$ and $x_2^{(1)} = 0.75$. The fixed points of order two must meet $x_{n+2} = x_n$, and $x_{n+2} = x_{n+1}$. Thus we have

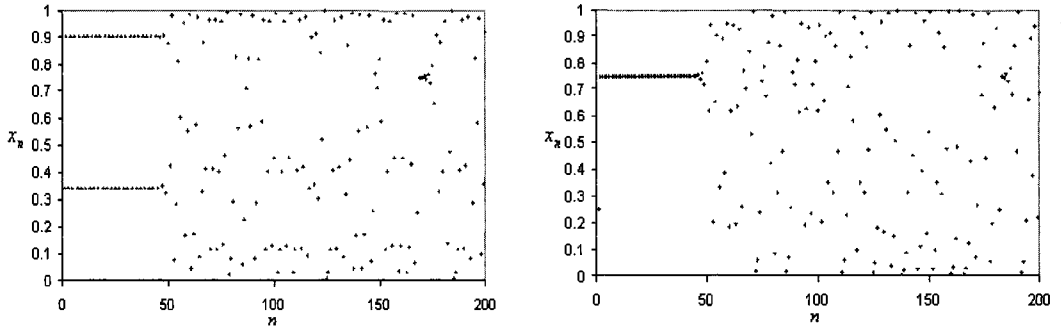
$$\begin{aligned} x_{n+2} &= 4x_{n+1}(1-x_{n+1}) \\ &= 4[4x_n(1-x_n)][1-4x_n(1-x_n)] \\ &= 4^2 x_n(1-x_n)[1-4x_n(1-x_n)] = x_n, \end{aligned} \tag{2.59}$$

and

$$\begin{aligned} x_{n+2} &= 4x_{n+1}(1-x_{n+1}) \\ &= 4[4x_n(1-x_n)][1-4x_n(1-x_n)] \\ &= [4x_n(1-x_n)]. \end{aligned} \tag{2.60}$$

So the fixed points of order two are $x_1^{(2)} = 1$ and $x_2^{(2)} = 0.25$ and $x_3^{(2)} = \frac{5+\sqrt{5}}{8}$ and $x_4^{(2)} = \frac{5-\sqrt{5}}{8}$.

Similarly, the fixed points of order three are $x_1^{(3)} = 0.5$ and $x_2^{(2)} = \frac{2+\sqrt{3}}{4}$ and $x_3^{(2)} = \frac{2-\sqrt{3}}{4}$. Over order three, no more real valued fixed points exist. When we study



(a) At $x = \frac{5+\sqrt{5}}{8}$ (b) At $x = \frac{2+\sqrt{3}}{4}$

Fig. 2. 12 The logistic map for $\mu = 4.0$

the fixed points of the logistic map, an interesting thing happens for $\mu = 4.0$, at different initial value $x = \frac{5+\sqrt{5}}{8}$ and $x = \frac{2+\sqrt{3}}{4}$ (seen in Fig. 2.12). For Fig. (a) in Fig. 2.12, the values of x_n are supposed to be $x = \frac{5+\sqrt{5}}{8}$ every two iterations , and for Fig. (b) in Fig. 2.12, the values of x_n are supposed to be $x = 0.75$ after two iterations. However, the map becomes unstable after iterating about 50 times since $\sqrt{3}$ and $\sqrt{5}$ are irrational numbers that cannot be represented by a digital computer. This is another instance of very small differences in the initial value causing large differences in long-time behavior.

The probability density distribution of the Logistic map is not a uniform distribution; the middle of the interval [0,1] is more sparsely populated than the ends. Table 2.3 shows a distribution of the Logistic map by starting with a random value x_0 in the interval [0,1] for $\mu = 4.0$ after iterating 100 times, 1000 times and 10000 times, separately. When we develop a chaotic optimization algorithm, we need to take this special distribution into consideration; one option is to compress the search space.

Table 2. 4 Density distribution of the logistic map

Number of Iteration	$0 \le x \le 0.3$	$0.3 < x \le 0.7$	$0.7 < x \le 1$
100	37	25	38
1000	374	262	364
10000	3678	2620	3702

Ulam and von Neumann studied the chaotic behavior of the nonlinear self mapping of the interval [-1,1] in 1947 [56], defined by

$$y_{n+1} = 1 - r y_n^2, \; y_n \in [-1,1], \tag{2.61}$$

with probability density function

$$p_f(y) = \frac{1}{\pi\sqrt{1-y^2}}. \tag{2.62}$$

A bifurcation diagram of this map is presented in Fig. 2.13. Again, we plot a series of values for y_n as a function of r obtained by starting with a random value y_0, iterating 200 times, and discarding the first 100 points. In this diagram the map fills much of the interval [-1,1] at $r = 2$.

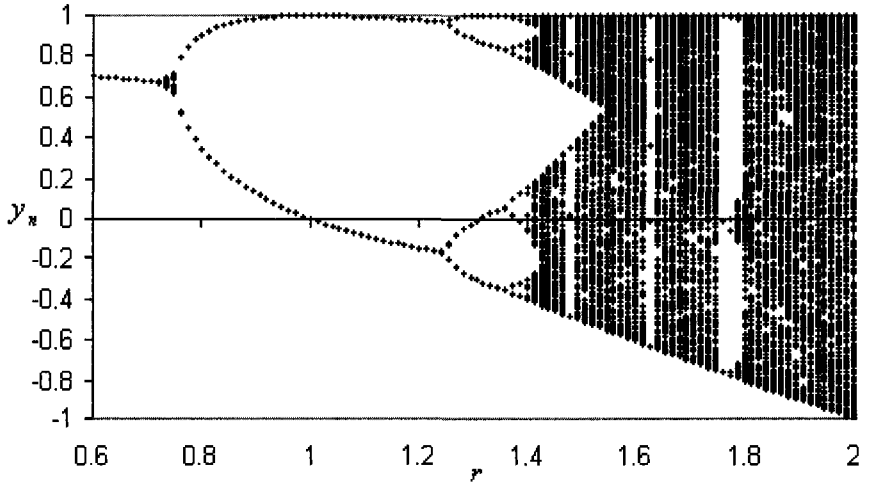


Fig. 2. 13 Bifurcation diagram of the Ulam-von Neumann map, long-term values of y_n are plotted for $0.6 \le r \le 2$.

Actually, the Ulam-von Neumann map is the conversion form of the Logistic map. Let

$$y = 2x - 1, \tag{2.63}$$

and substitute into Eq. (2.61), the Ulam-von Neumann map is converted into the Logistic map. Therefore, the Ulam-von Neumann has all chaotic behaviors which the Logistic map possesses.

2.4.2 Existing chaotic simulated annealing methods

The Transiently Chaotic Neural Network (TCNN) [30] was created to overcome the problem of local minima in Hopfield neural networks [45][46]. TCNN introduces a chaotic simulated annealing algorithm into the Hopfield neural network. Chaotic dynamics are used in place of random variates to search for a global optimum point in weight space; they are implemented via temporary feedback connections. Over time, the chaotic dynamics are reduced and finally disappear during training, using a decreasing *bifurcation parameter* that plays the role of the temperature parameter in the standard simulated annealing algorithm. In other words, TCNN is gradually converted to the traditional Hopfield neural network and converges to a stable equilibrium point. Unlike stochastic simulated annealing, chaotic simulated annealing has completely deterministic dynamics; this does imply that TCNN is not guaranteed to settle down to a global minimum no matter how slowly the annealing parameter (self-coupling) is reduced [41].

A transiently chaotic neuron [30] is defined as:

$$x_i(t) = \frac{1}{1 + e^{-y_i(t)/\varepsilon}} \tag{2.64}$$

$$y_i(t+1) = k y_i(t) + \alpha \left(\sum_{j=1, j \neq i}^{n} \omega_{ij} x_j(t) + I_i \right) - z_i(t)(x_i(t) \tag{2.65}$$

$$z_i(t+1) = (1-\beta) z_i(t) \qquad (2.66)$$

$$(i = 1, \ldots, n)$$

where $y_i(t)$ is the internal state of neuron i at time t,

ω_{ij} is connection weight from neuron j to neuron i, let $\omega_{ij} = \omega_{ji}$, $\omega_{ii} = 0$,

$\sum_{j=1, j\neq i}^{n} \omega_{ij} x_j(t) + I_i = -\partial E / \partial x_i$, $x_i(t)$ is the output of neuron i at time t,

E is the system error,

β is the damping factor of the time-dependent $z_i(t)$ ($0 \leq \beta \leq 1$),

I_i is the input bias of neuron i,

ε is a "steepness" parameter of the neuron output ($\varepsilon > 0$),

α is a scaling parameter for the inputs ($\alpha > 0$),

$z_i(t)$ is the feedback connection weight or refractory strength ($z_i(t) \geq 0$).

Since the "temperature" $z_i(t)$ evolves with time according to $z_i(t) = z_i(0) e^{(1-\beta)t}$, Eq. (2.65) eventually shrinks to the continuous-time Hopfield neural network without self-feedback connections, namely $\omega_{ii} = 0$. With exponential damping of $z(t)$, the neuron output $x(t)$ gradually transitions from chaotic behavior to a fixed point through reversed period-doubling bifurcations.

The nonlinear dynamics approach [31] is an additive chaotic forcing scheme for solving optimization problems. The gradients of the global optimization problem are viewed as a system of ordinary differential equations, a chaotic function is added, and the

entire system is then discretized to the Euler form. Normally, the gradients of a given cost function $f(x)$ can be treated as a system of ordinary differential equations:

$$\frac{dx_i}{dt} = -\frac{\partial f}{\partial x_i}, i = 1...n \tag{2.67}$$

with appropriate initial conditions: $x_i(t=0) = x_{i0}, i = 1,\ldots,n.$ For global optimization problems, Eq. (2.67) will have a few stable equilibrium points depending on the choice of initial conditions. However, finding a set of the initial conditions guaranteeing an approach to the equilibrium point corresponding to the global minimum of the given objective function $f(x)$ is of course difficult. The nonlinear dynamics approach is to convert the system of equations to a dynamic system [54], expressed as

$$\frac{dx_i}{dt} = -\frac{\partial f}{\partial x_i} + k_i C(t), i = 1...n, \tag{2.68}$$

where $C(t)$ is the chaotic function, and the scalar k_i controls the magnitude of added chaos. The explicit Euler technique is then used to solve the set of Eq. (2.68). Thus, the continuous-time dynamical system Eq. (2.68) is replaced by its equivalent discrete dynamical system

$$x_i^{j+1} = x_i^j - h\frac{\partial f^j}{\partial x_i} + k_i C^j, i = 1...n, \tag{2.69}$$

where h is the step size constant, indices i and j refer to the system variables and the time step, respectively, and the logistic map, Lorenz or Rossler systems can be used as the chaotic function. For example, the logistic map is represented as

$$C'^{m+1} = 4C'^{m}\left(1 - C'^{m}\right);\ C^{j+1} = C'^{m} - 0.5;\ -0.5 \le C^{j+1} \le 0.5. \tag{2.70}$$

The scaling factor k controls the rate at which Eq. (2.69) is brought into the vicinity of the global optimum. Plainly, this scalar is quite similar to the temperature variable in simulated annealing; however, annealing was never considered in [31].

The Chaotic Simulated Annealing (CSA) algorithm [11] introduced two different chaotic systems into the simulated annealing algorithm. The first chaotic system is the well-known one-dimensional logistic map defined by

$$z_{k+1} = \mu z_k (1 - z_k), z_k \in [0,1], k = 0,1,\ldots, \tag{2.71}$$

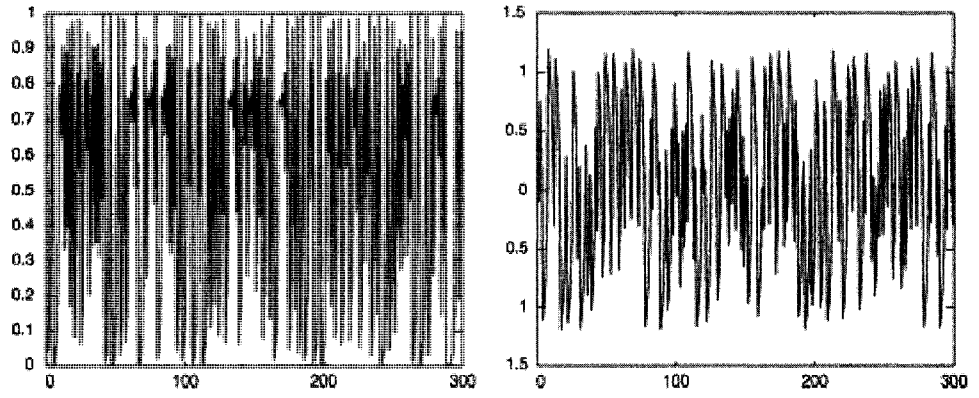
where z_k is the value of the variable z at the kth iteration.

The other chaotic system was developed for chaotic neurons [57][58], and consists of the map

$$z_{k+1} = \eta z_k - 2\tanh(\gamma z_k)\exp(-3z_k^2), k = 0,1,\ldots, \tag{2.72}$$

where z_k is the internal state of the neuron, η is a damping factor of the nerve membrane ($0 \leqslant \eta \leqslant 1$) and γ is an adjustable parameter depending on the practical application.

The output of equation (2.71) and equation (2.72) are shown in Fig. 2.14. From Fig. 2.13, the output of the one-dimensional logistic map is limited to the interval [0,1], while the output of chaotic neuron is not. These maps were used to replace the random number generators in a standard simulated annealing algorithm. The algorithm was first proposed to sample the input space and find a near optimal solution **x** to an objective function $f(\cdot)$ to be minimized. Similar to simulated annealing, an objective function $f(\cdot)$ is analogous to the energy in a thermodynamic system, expressed as $E = f(\mathrm{x})$, and the annealing schedule specifies how rapidly the temperature T is lowered from high to low values. The algorithm consists of three steps:



(a) (b)

Fig. 2. 14 (a) Logistic mapping at , μ=4,
(b) Chaotic neuron mapping at z_0=0.01, γ=5, η=0.9, k=300 [11].

Step 1: Initialization.

Set the iteration count of chaotic system k to 0, and initialize the chaotic variable $z_{0,i}$ to generate the next chaotic variables $z_{1,i}$, $i=1,\ldots,n$ by Eq. (2.71) or (2.72), i is the index of the input vector **x** of the objective function, n is the length of input vector **x**. The initial solution to the objective function $f(\cdot)$, $x_1=(x_{1,1},x_{1,2},\ldots,x_{1,n})^T$, is produced by the formula $x_{1,i}=a_i+(b_i-a_i)\times z_{1,i}$, $i=1,\ldots,n$, where a_i and b_i are the minimum and maximum of the i th dimension of the input vector x_1, separately. Set the temperature T to a high starting value $T_{\max}$, and choose a minimum temperature $T_{\min}$ which stops executing algorithm, and define the maximum iteration count $L_{\max}$ at each temperature. Set the iteration count m to 1 at each temperature. Let $x^*=x_1$ and $f^*=f(x^*)=f(x_1)$, where x^* is the current best solution to the objective function, f^* is the current minimum value of the objective function at **x**= x^*.

Step 2: Search

while ($T > T_{\min}$) do

(a) while $(m \le L_{\max})$ do

(1) Generate a new solution to the objective function, $y_m = (y_{m,1}, y_{m,2}, \ldots, y_{m,n})^T$, by $y_{m,i} = x_{m,i} + \alpha \times (b_i - a_i) \times z_{m,i}$; α is a variable, which decreases by $\alpha = \alpha \times e^{-\beta}$ in each iteration, where β is a constant between 0 and 1, a_i and b_i are the minimum and maximum of the ith dimension of the input vector x_m of the objective function.

(2) Evaluate the change in energy level $\Delta E^* = f(y_m) - f^*$ and $\Delta E = f(y_m) - f(x_m)$.

(3) If $\Delta E^* \le 0$ update the best solution $x^* = y_m$ and the best value $f^* = f(x^*)$.

(4) If $\Delta E \le 0$ update current state with new state, $x_{m+1} = y_m$.

(5) If $\Delta E > 0$ update current state with new state with probability $e^{-\frac{\Delta E}{T}}$.

(6) $m=m+1$.

End while

(b) $L_{\max} = L_{\max} + d, m = 0$, where d is a positive integer, which allows algorithm to search for more feasible solutions.

(c) Decrease temperature T according to annealing schedule by formula $T = \delta \times T$, where δ is a constant between 0 and 1.

end while

Step 3: Output the best solution x^* and the best value f^*.

CSA with Eq. (2.71) is called CSA1, and CSA with Eq. (2. 72) is named CSA2. The disadvantage of CSA1 is that the next candidate solution to the objective function is actually generated on a single side of the current optimum, not in the whole of the neighborhood. This results in slower computation and has a high impact on the quality of the approximate optimum point. The disadvantage of CSA2 is that the algorithm does not make full use of each trial solution and is inefficient because the values of chaotic sequences generated by Eq. (2. 72) exceed the range (-1,1), so that many trial solutions do not satisfy the nonlinear constraint function.

2.5 Ant colony optimization

Ant colonies are a community. Despite the small size and limited intelligence of each individual member of a colony, this highly organized society can accomplish complex tasks, that could not be accomplished by a single ant. Ant colony optimization (ACO) [8]-[10] is a metaheuristic algorithm derived from the observation of real ant's foraging behavior. The first framework was presented by Marco Dorigo in his PhD thesis, and is a probabilistic technique for solving computational problems, which can be reduced to find good paths through graphs. Now ant colony optimization mehtod has became one of the most succesful and widely recognized algorithmic techniques due to its ability to find shortest paths.

In the real world, when ants start to seach for food, they wander at random. Once they find food, they come back to their colony while laying down pheromone trails. If other ants find such a path, they tend not to keep traveling randomly, but rather to follow the trail instead, and the pheromone trails will be reinforced back and forth between the food source and their colony. In other words, positive feedback through the pheromone trails

eventually leads to all ants following a single path. Over time, however, the pheromone trail begins to evaporate, which decreases its attraction. The more time it takes for an ant to find the food source and return again, the less pheromones remain on the path due to evaporation. Compared to a long path, ants spend less time on a short path, keeping the pheromone density high since the production rate of the pheromone on the path is almost the same as its evaporation rate or larger. The advantage of ant colony optimization using pheromone evaporation is to avoid convergence to a locally optimal solution. Assuming that there were no evaporation at all, the paths selected by the first ants would be excessively attractive to the following ones. In that case, the exploration of the solution space would be limited, and thus the algorithm would probably not find the global optimal solution. In the following, we will describe the implementation of the ACO algorithm for function optimization problems.

2.5.1 Initialization

Define the number of the outer iterations, N_outer, and the number of the inner iterations, N_inner, the number for the feasible solution in the neighborhood, N_search, the number of the ant colony, M, evaporation coefficient, ρ, and the initial neighborhood D_i of the optimization variables is defined by

$$D_i = 0.1 \times (b_i - a_i), \tag{2.73}$$

where a_i, b_i are the lower and upper bounds of the optimization variables, and the neighborhood D_i tunes the step size for each variable of the objective function to optimize.

At the beginning of the ACO algorithm, a group of trial solutions to the nonlinear constraint objective function are randomly generated according to the equation

$$s_i = a_i + (b_i - a_i) \times r, \tag{2.74}$$

where a_i, b_i are the lower and upper bounds of the optimization variables. r is a random number in the interval (0,1). We select the best one satisfying the constraint functions as the current optimal solution, $S_{current}$. Similarly, another set of feasible solutions satisfying the constraint requirements, an ant colony, are built randomly.

2.5.2 Update Probability

The algorithm explores the optimal solution based on the ant colony, in which the probability of the ant i updated by the ant j depends on the combination of two values: the attractiveness η_{ij}, expressed by heuristic values from the objective function and indicating the priori desirability of that update; and the pheromone trail τ_j, indicating the posteriori desirability of that update and how proficient it has been in the past to make that particular update.

The update probability between the ant i and the ant j is defined as:

$$p_{ij} = \begin{cases} \dfrac{\tau_j{}^{\alpha} \eta_{ij}{}^{\beta}}{\sum_{k=1}^{M} \tau_k{}^{\alpha} \eta_{ik}{}^{\beta}} & \eta_{ij} > 0 \\ 0 & \eta_{ij} < 0 \end{cases}, i = 1,2,\ldots,M,\ j = 1,2,\ldots,M, \tag{2.75}$$

where M is the number of the ant colony; η_{ij} is the difference of the values of the objective function between the ant i and the ant j; η_{ik} is the difference of the values of the objective function between the ant i and the ant k, and only the positive value of

the η_{ik} is chosen in the calculation; α and β are the user defined parameters, here we set them to 1, separately.

If $p_{ij} > p_{transition}$, ($p_{transition}$ is called transition probability, which is set to 0.1 in the algorithm), then the ant i is replaced by the ant j. Other wise, the N_search new trial solutions are generated in the neighborhood of the ant j to find a local optimal solution according to Eq. (2.76):

$$X_{m+1} = X_m + Du, m = 1,2,\ldots N\_search, \tag{2.76}$$

where u is a vector of random numbers in the range (-1,1), and D is a matrix that controls the step size distribution. If the value of the objective function of the ant i is less than that of the ant j, then the ant i will be replaced by the ant j. When all ants' impacts on the ant i are evaluated, the pheromone trails are updated, until M ants build in parallel their solutions. The update of the pheromone trail is achieved by:

$$\tau_i(t+1) = \rho\tau_i(t) + \Delta\tau_i(t), \tag{2.77}$$

$$\Delta\tau_i(t) = \sum_{j=1}^{M} \frac{1}{f(X_j)}, j = 1,2,\ldots,M, \tag{2.78}$$

where $f(X_i)$ is the value of the objective function of the ant i which has been updated by the ant j. Intuitively, the small function value will result in higher levels of pheromone deposited on the paths. $\Delta\tau_i(t)$ indicates the sum of the contributions of all ants to the ant i. And ρ is pheromone evaporation rate.

Pheromone evaporation is an important concept in ACO, by means of which the pheromone trail intensity on the components decrease over time. During executing the

algorithm, pheromone evaporation is required to avoid quickly converging to a sub-optimal region. Here we set ρ to 0.7.

2.5.3 Output of Solutions

After M ants obtain in parallel their solutions, these solutions are compared with the $S_{current}$ to find the best one and store as $S_{current}$.

2.5.4 Adjustment of the Step Size

The exponential cooling scheme is used as the step size decrement rule, represented as:

$$D_{k+1} = vD_k \tag{2.79}$$

where v is set to 0.5 in the algorithm.

Repeat these procedures mentioned above until the termination criteria is satisfied.

2.6 Complex-valued neural networks

The ANCFIS architecture we propose is naturally a relative of the complex-valued neural network (CVNN) architectures, which have been studied for over 15 years. CVNNs generally accept complex-valued inputs and outputs, and their neuron weights and biases may also be complex-valued. Previous CVNN models [61]-[65] have generalized the Hopfield model, backpropagation and the perceptron learning rule to deal with complex inputs. Noest [61] introduced an associative memory network with local variables assuming one of q equidistant positions on the unit circle (q-state phasors) in the complex plane. Leung and Haykin [63] extended real-valued backpropagation networks to complex-valued backpropagation networks, for general radar signal processing and communications problems in which a complex-valued representation of signals is

required. In [63], the neuron activation function is expressed as $y_j = 1/(1 = \exp(-\sum w_{ji} x_i))$ with complex variables, and the dynamics with partial derivatives in real and imaginary parts are analyzed. Benvenuto & Piazza [64] considered a CVNN whose activation is expressed separately in real and imaginary parts as $f(z) = sigmoid(\mathrm{Re}(z)) + i\, sigmoid(\mathrm{Im}(z))$, namely the real-imaginary –type activation function, which can be regarded as one of the simplex extensions of a real sigmoid activation function. Dynamics of CVNNs with real-imaginary-type activations were analyzed when they are applied to complex-plane transform in [66][67]. Kim & Adali [68][69] discussed the characteristics of activation functions in details. Akira Hirose [70] proposed a developmental learning architecture based on the complex-valued coherent neural network. The input signal x_m, output signal y_n and weight ω_{nm} are all complex numbers with amplitude and phase. The neural connection weight ω_{nm} was expressed by $|\omega_{nm}|\exp[i2\pi f\tau_{nm}]$, where $i \equiv \sqrt{-1}$, $|\omega_{nm}|$ is the connection amplitude, τ_{nm} is the delay time and f is the carrier frequency which is used as the modulation parameter of the behavioral mode, that means, if the parameter value of f is fixed, the behavioral mode is also fixed. In the complex-valued coherent neural network, an amplitude-phase-type neuron activation function is expressed by $A\tanh(gs_n)\exp[i\beta_n]$, where A and g are real numbers, $s_n \exp[i\beta_n]$ is the complex-valued input summation $\sum_m \omega_{nm} x_m$ with amplitude s_n, phase β_n.

2.7 Complex-valued ANFIS

To date there has been only one attempt to develop an inductive learning architecture using complex fuzzy sets. This architecture, named CANFIS [71] was used to handle complex-valued input-output pairs, and to model a simple lead-lag compensator transfer function of form $k(1+TS)/(1+T'S)$, where $S = j\omega$. Their architecture is a hybrid of complex fuzzy sets that processes the output of a complex-valued single-layer neural network. The real-valued adaptive parameters of the complex membership functions for each rule were updated by a steepest descent algorithm, and the bias and weight parameters of complex neurons were updated by a complex least square estimator. In this work, the complex fuzzy sets follow the basic concept laid down by Ramot [2]: the phase and magnitude of the complex fuzzy sets are treated as uncoupled quantities. In [1], we argue that this undermines the fundamental nature of complex memberships. A complex membership is not merely a two-dimensional membership function; it is a vector, a *unitary* quantity. Without that coupling, it is natural to ask why complex fuzzy sets are required; a two-dimensional membership function (i.e. for complex input/output values) could simply be created from two type-1 fuzzy sets. That is precisely the approach taken in an earlier architecture, also dubbed CANFIS, developed by Li and Jang [72]. This is a straightforward extension of the ANFIS architecture to complex-valued inputs and outputs. Each complex-valued input was divided into its real and imaginary components, and two Gaussian type-1 fuzzy sets were associated with each component of each input. For three complex inputs, this architecture leads to 64 rules. As in the original ANFIS architecture, the consequent parameters were adjusted using the least-squares algorithm, while the premise parameters were adjusted via gradient descent. The ANCFIS system

we propose in the current paper is the first, and so far only, complex fuzzy system that couples phase and magnitude. As will be seen, this approach also results in a very parsimonious network structure, an advantage not enjoyed by the architectures in [71] and [72].

3 Complex fuzzy inference system

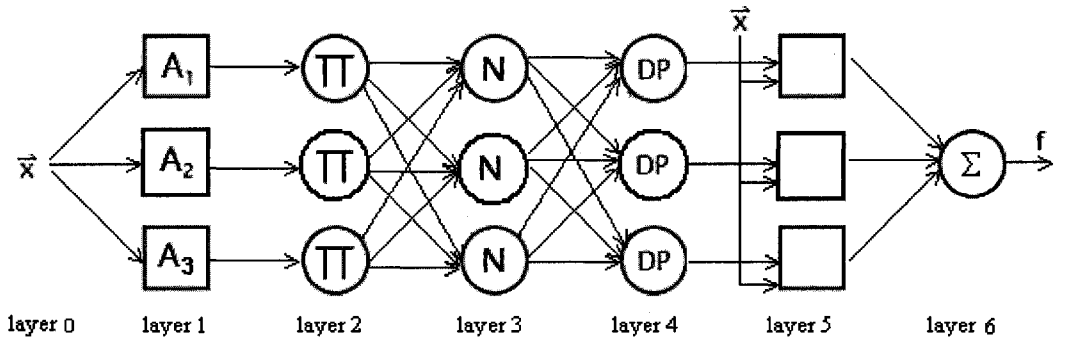


Fig. 3. 1 An ANCFIS architecture

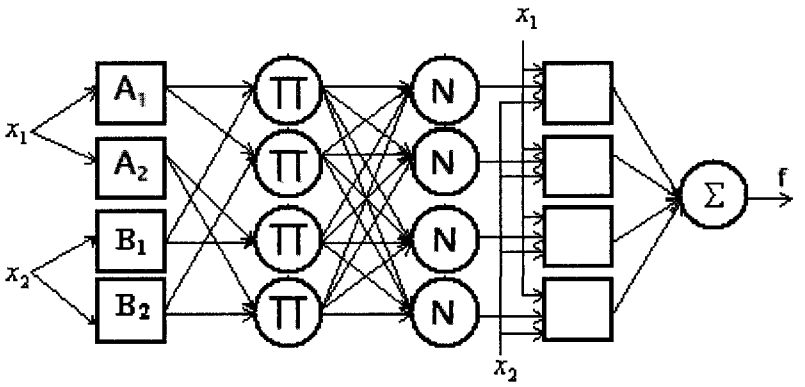


Fig. 3. 2 ANFIS architecture for a two-input Sugeno fuzzy model with four rules from [5]

We have developed the ANCFIS architecture to inductively determine a complex fuzzy inferential rule base, and implement complex fuzzy inference. The signature capability of a complex fuzzy inference system is *rule interference*, which is the property of rules to reinforce or contradict each other (constructive or destructive interference). This mechanism was first postulated in [3]; to the best of the author's knowledge, ANCFIS represents the first realization of this mechanism in a machine-learning algorithm. ANCFIS is a six-layer adaptive neural network, with adaptive nodes in layers one and five. An example of the ANCFIS architecture, assuming one input and three rules, is depicted in Fig. 3.1.

ANCFIS is based on the well-known ANFIS architecture developed by Jang [6], with substantial modifications. A two-input ANFIS network with two fuzzy sets per input is depicted in Fig. 3.2 for comparison. The most obvious modification is the addition of an additional layer; this is the dot-product layer in ANCFIS, which implements the rule-interference property. Less obvious, but crucial to the success of ANCFIS, is the use of complex fuzzy sets in Layer 1, rather than type-1 fuzzy sets. ANCFIS is intended to be a parsimonious, effective time series forecasting algorithm; this property flows directly from the incorporation of complex fuzzy sets. The generalized *modus ponens* rule [4] requires inputs to be compared to rule antecedents, to determine a *firing degree* for each rule. In the original ANFIS, a time series must be converted to orthogonal features (usually a *lagged representation*) before it is presented to the network. The inputs are then compared to type-1 fuzzy sets using the algebraic product. In ANCFIS, by contrast, complex fuzzy sets may be directly compared to a segment of the time series, using the complex-valued convolution operator (chosen after empirical comparisons against real-valued convolution and the L2 norm in [73]). Thus, while ANFIS may need several inputs to capture a segment of a univariate time series (leading to a combinatorial explosion in the number of inferential rules), ANCFIS needs only a single input consisting of an a window of the time series. ANCFIS thus side-steps the "curse of dimensionality" by requiring a lower input dimensionality than conventional machine learning algorithms. In multivariate time series, ANCFIS will require one input per variate.

As in the original ANFIS architecture, ANCFIS uses the hybrid learning rule [6].

In the forward pass, the parameters of layer five (the linear consequent function) are determined via least-squares optimization. In the backward pass, the parameters of layer one (the antecedent complex fuzzy sets) must be determined. In ANFIS, this was accomplished via gradient descent. However, the partial derivatives of the parameters of the complex fuzzy sets in ANCFIS do not have a closed-form solution. Thus, we turn to a derivative-free optimization technique to update the complex fuzzy set parameters of layer one. The precise choice of technique to employ will be determined via empirical comparison between ACO, CSA2, and our new VNCSA algorithm.

3.1 Complex Fuzzy Sets

As discussed, inputs in ANCFIS are windows of a time series, to be combined with a complex fuzzy set via complex-valued convolution. We thus need to specify a parameterized functional form for these complex fuzzy sets that is compatible with this architecture, and will also contribute to our overall objective of time series forecasting. Sinusoids are one obvious choice; we know that under very general conditions, an essentially arbitrary function can be represented by a Fourier series, the sum of a series of sine and cosine terms. The complex fuzzy sets we introduce in this paper are of the form

$$r(\theta) = d\sin(a\theta + b) + c \tag{3.1}$$

where $r(\theta)$ is the radial distance from the origin, and θ is the counterclockwise angle from the positive real axis (i.e. polar coordinates). The parameters $\{a,b,c,d\}$ allow us to manipulate the sine function; a increases or decreases the frequency of a sine wave, while b introduces a phase shift, c shifts the sine wave, and d changes the amplitude of the wave. Since the amplitude of complex fuzzy memberships are limited to the unit disc of the

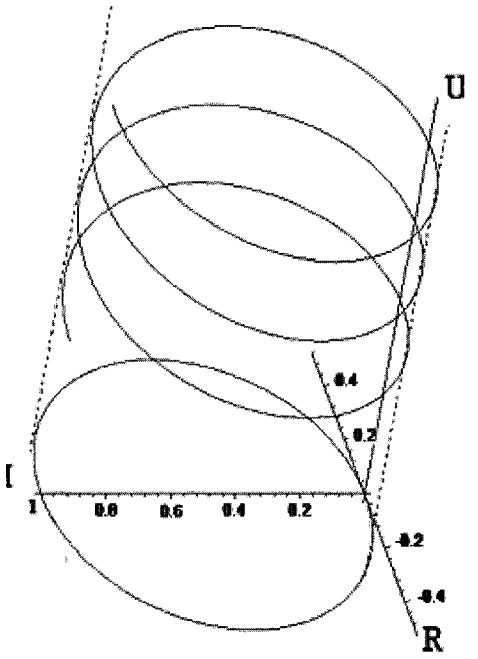


Fig. 3. 3 Visualization of a complex fuzzy set, $r_S(\theta) = \sin(\theta)$

complex plane, the membership function must satisfy two constraints: $0 \le d + c \le 1$, $1 \ge c \ge d \ge 0$. Importantly, this membership function tightly couples the phase and magnitude of the complex fuzzy set.

The membership function of our complex fuzzy set can be visualized by placing the complex plane $R \times I$ at right angles to the universe of discourse U. The complex membership function then forms a trajectory within the cylinder formed by projecting the unit disc $\boldsymbol{D}$ along U. For example, consider the function $r_S(\theta) = d\sin(a\theta + b) + c$ with $d = 1, a = 1, b = 0, c = 0$ in Fig. 3.3. An interesting property of this sine function is that, if you project the trajectory of this complex fuzzy set back to the complex plane, you get a circle of radius 0.5 centered at $0 + 0.5j$. We have previously speculated [1] that

membership functions whose projections lead to closed contours (especially convex contours) are likely to be an important class of complex fuzzy sets.

3.2 ANCFIS architecture

Layer 1: Premise parameters

The parameter set for layer 1 is $\{a_i, b_i, c_i, d_i\}$, $i = 1,2,\cdots,n\_CMF$, where n_CMF is the number of complex membership functions (henceforth, the premise parameters). The first layer computes the convolution of each membership function and the input vector. Firstly, the membership function is sampled by

$$r_k(\theta_k) = d\sin(a\theta_k + b) + c \quad , \tag{3.2}$$

$$\theta_k = \frac{2\pi}{n}(k + iteration\_count) \, , \tag{3.3}$$

where n is the length of input vector, and k represents the element index of the complex samples, $k = 1,2,\ldots,n$. $iteration\_count$ is an index on the training data patterns. Then the sampled points are transformed from polar to rectangular coordinates using the well-known transforms

$$x_k = r_k \cos(\theta_k) \, , \tag{3.4}$$

$$y_k = r_k \sin(\theta_k) \, . \tag{3.5}$$

These complex-valued samples are convolved with the original real-valued input vector. If the functions involved in convolution are continuous, the convolution is an integral that represents the amount of overlap of one function as it is shifted over another function. The reader will notice that the input vector presented to the ANCFIS and the

sampled points generated from the complex membership function via Eqs. (3.2), (3.3), (3.4), (3.5) are both discrete time series, not continuous functions. Therefore, in this case, the convolution is a sum instead of an integral. Let f represents the input vector, and g represents sampled points vector, $m = length(f)$, and $n = length(g)$, then h is the vector of the length $m+n-1$, whose k th element is

$$h(k) = \sum_j f(j)g(k+1-j), \tag{3.6}$$

$$g(k+1-j) = x_{k+1-j} + iy_{k+1-j} = r_{k+1-j}\cos(\theta_{k+1-j}) + ir_{k+1-j}\sin(\theta_{k+1-j}), \tag{3.7}$$

the sum is over all the values of j which lead to legal subscripts for $f(j)$ and $g(k+1-j)$, specifically $j = \max(1, k+1-n) : \min(k, m)$. Assuming that both of two vectors have the same length, $m = n$, this gives

$$h(1) = f(1) * g(1)$$
$$h(2) = f(1) * g(2) + f(2) * g(1)$$
$$\ldots$$
$$h(n) = f(1) * g(n) + f(2) * g(n-1) + \ldots + f(n) * g(1)$$
$$\ldots$$
$$h(2n-1) = f(n) * g(n)$$

Thus we have

$$convolution\ sum = \sum_{k=1}^{2n-1} h(k) = \sum_{k=1}^{2n-1} \sum_{j=\max(1,k+1-n)}^{\min(k,n)} f(j)g(k+1-j), \tag{3.8}$$

The following example demonstrates how to calculate the summation of the results of convolution between the input vector and the complex samples from CMF. For example, assume that we have two membership functions, and the length of input vector is 2. The number of points of sampled membership function should be equivalent to the length of input vector, namely, both of them have the same dimension. From Eqs. (3.2), (3.3), we sample 2 points from each complex membership function,

Y1=[2+3i, 3+4i] ------ complex samples from MF1,

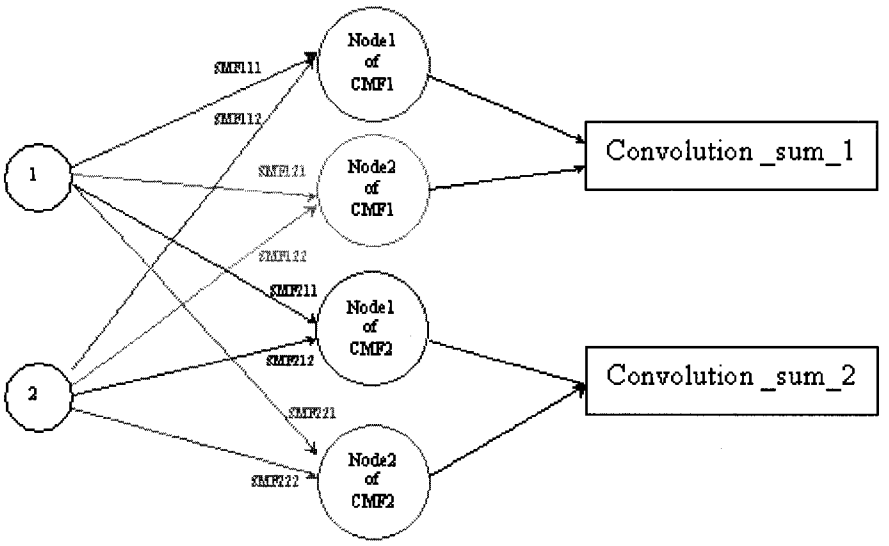


Fig. 3. 4 Illustration of Implicit structure in the convolution of input vector and sampled points generated from complex membership function

($SMF111 = SMF112$, $SMF121 = SMF122$, $SMF211 = SMF212$, $SMF221 = SMF222$)

Y2=[1+2i, 4+7i] ------ complex samples from MF2,

and X=[3, 2] ------ input vector,

where $x_1 = 3, x_2 = 2$

$2+3i = SMF_{111}$, $2+3i = SMF_{112}$, $3+4i = SMF_{121}$, $3+4i = SMF_{122}$;

$1+2i = SMF_{211}$, $1+2i = SMF_{212}$, $4+7i = SMF_{221}$, $4+7i = SMF_{222}$,

sum (Conv(X,Y1))= $SMF_{111} * x_1 + SMF_{112} * x_2 + SMF_{121} * x_1 + SMF_{122} * x_2$,

sum (Conv(X,Y2))= $SMF_{211} * x_1 + SMF_{212} * x_2 + SMF_{221} * x_1 + SMF_{222} * x_2$,

From the formula of the summation of the convolution between the complex samples of the complex membership functions and the elements of the input vector of presented to the neural network, it is interesting to notice that this summation of the convolution can

be viewed as the implicit neural network, which may be illustrated in Fig. 3.4, where SMF_{ikj} can be regarded as the weight connecting the j th element of the input vector presented to ANCFIS to the k th sample point of the i th membership function in the first layer. The different elements of input vector connected to the same sample point from the membership function share the same weight. In other words, the weights connecting the same sample point from the complex membership function to different elements of the input vector are always the same.

From Fig. 3.4, since the convolution sum can be viewed in the form of a neural network, and this provides essential guidance in deriving the gradient equations for the ANCFIS learning algorithm. In other words, these complex-valued weights connected the elements of the input vector and the nodes generated from the complex membership function can be updated using gradient descent method. Thus, Eq. (3.8) can be rewritten

$$convolution\_sum = \sum_{j=1}^{n}\left(O_{0,j} \times \sum_{k=1}^{n} SMF_{ikj} \right), \tag{3.9}$$

where n is the length of input vector, $O_{0,j}$, $j = 1,2,\ldots n$, denotes the j th element of input vector presented to the inference system.

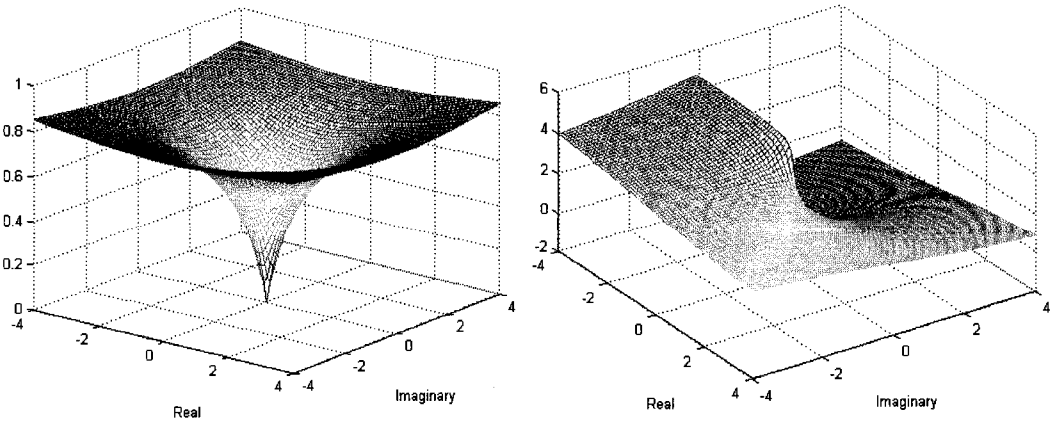


Fig. 3. 5 The Elliott function, (Left) Magnitude. (Right) Phase

In order to keep this value within the complex unit circle, and keep its phase the same, we need to find an appropriate activation function and apply it to the sum of all components of the convolution. We choose Elliot function [65] to process real and imaginary components of complex number jointly,

$$f(z)=\frac{z}{1+|z|}, \tag{3.10}$$

where z is a complex number. Fig. 3.5 shows the magnitude and phase of the joint-complex activation function, which is suitable for processing information meaning in rotation around the origin of coordinate.

Substituting Eq. (3.9) into Eq. (3.10), the output of the nodes in the first layer is represented by

$$O_{1,i}=\frac{\sum_{j=1}^{n}\left(O_{0,j}\times\sum_{k=1}^{n}SMF_{ikj}\right)}{1+\left|\sum_{j=1}^{n}\left(O_{0,j}\times\sum_{k=1}^{n}SMF_{ikj}\right)\right|}, i=1,2,\cdots,n\_CMF. \tag{3.11}$$

In the following, the complex-valued output of node i in layer l is denoted $O_{l,i}$.

Layer 2: Firing strength

Each node labeled $\prod$ in layer 2 is a fixed node that is responsible for multiplying the incoming signals and delivering the product to the next layer.

$$O_{2,i}=\prod_{i}O_{1,i},\ i=1,2,\cdots,n\_CMF. \tag{3.12}$$

In Fig. 3.1, we only have one input vector to the network $\bar{x}$, and the output of layer 2 is the same as layer 1. The output from every node in layer 2 represents the firing strength of a fuzzy rule. Although it is not shown here, this layer allows for interaction between

multiple input vectors, in the case of multivariate time series. The algebraic product is also the complex fuzzy intersection derived in [1].

Layer 3 Normalized firing strength

The output of each node labeled N in layer 3 represents the i_{th} rule's normalized firing strength:

$$O_{3,i} = \overline{w}_i = \frac{w_i}{\sum_{j=1}^{n\_CMF} |w_j|}, \; i = 1,2,\cdots,n\_CMF, \tag{3.13}$$

where $\sum_{j=1}^{n\_CMF} |w_j|$ refers to the summation of the magnitude of each weight w_j. This layer only normalizes the magnitude of the weights; phases are unchanged.

Layer 4 Dot product

The output of each node labeled DP in layer 4 is the dot product of each normalized firing strength and the sum of the outputs of all nodes in the previous layer. Note that the outputs of layer 4 are always real valued, which is desirable as the predicted output for time series data should be real valued.

$$O_{4,i} = w_i^{DP} = \overline{w}_i \bullet \sum_{i=1}^{n\_CMF} \overline{w}_i, \; i = 1,2,\cdots,n\_CMF, \tag{3.14}$$

where $\sum_{i=1}^{n\_CMF} \overline{w}_i$ is the complex sum. If all the phases of the various arguments are aligned, the amplitude of the sum will be maximized (constructive interference) If, however, the phases of all the arguments are not equal, destructive interference may occur, and the weights of some rules might be drastically reduced.

Layer 5 Consequent parameters

Each node i in layer 5 is an adaptive node. The output of each node i is

calculated by:

$$O_{5,i} = w_i^{DP} f_i = w_i^{DP}(\vec{p}_i \vec{x} + r_i)$$

$$= w_i^{DP}\left(p_{i,1}x_1 + p_{i,2}x_2 + \cdots + p_{i,j}x_j + \cdots + p_{i,n}x_n + r_i\right), j = 1,2,\ldots,n, \quad (3.15)$$

where w_i^{DP} is the i-th output of the 4th layer, x_j is the j-th value in the input vector and n is the length of the input vector, and $\{\vec{p}_i, r_i\}$ is the parameter set for the linear output function. $\vec{p}_i$ is a vector of the same length as the input vector $\vec{x}$, while r_i is a constant. Parameters in this layer are referred to as consequent parameters, and are identified in the forward pass using a linear least squares estimator [24][6][74].

Layer 6

In this layer, there is only a single node that calculates the overall output as the summation of all incoming signals.

$$O_{6,1} = \text{overall output} = \sum w_i^{DP} f_i . \quad (3.16)$$

3.3 Backpropagation for ANCFIS

This section introduces a backprogation learning rule for ANCFIS structure to update the sampled points generated from complex membership functions, which is in essence the simple steepest descent method. The main part of this basic learning rule involves how to recursively produce a gradient vector in which each element is defined as the derivative of an error measure with respect to a sampled point of complex membership function. This can be realized using the chain rule.

3.3.1 Derivative of complex function

Before deriving a gradient vector in ANCFIS structure, we need to seek a formula for computing the derivative $f'(z)$ of the complex function $f(z)=u(x,y)+iv(x,y)$ in terms of the partial derivatives of $u(x,y)$ and $v(x,y)$.

Let

$$f(z)=u(z)+iv(z)=u(x,y)+iv(x,y), \tag{3.17}$$

where $z=x+iy, \bar{z}=x-iy,$

we define

$$dz=dx+idy\ , \tag{3.18}$$

then the total derivative of f with respect to z can be calculated as follows:

$$x=\frac{z+\bar{z}}{2}\ , \tag{3.18}$$

$$y=\frac{z-\bar{z}}{2i}, \tag{3.19}$$

therefore

$$\frac{dx}{dz}=\frac{1}{2}\text{and}\ \frac{dx}{d\bar{z}}=\frac{1}{2}\ , \tag{3.20}$$

$$\frac{dy}{dz}=\frac{1}{2i}=-\frac{1}{2}i, \frac{dy}{d\bar{z}}=-\frac{1}{2i}=\frac{1}{2}i, \tag{3.21}$$

and

$$\frac{df}{dz}=\frac{\partial f}{\partial x}\frac{\partial x}{\partial z}+\frac{\partial f}{\partial y}\frac{\partial y}{\partial z}=\frac{1}{2}\left(\frac{\partial f}{\partial x}-i\frac{\partial f}{\partial y}\right). \tag{3.22}$$

According to u and v, Eq. (3.29) can be expressed by

$$\frac{df}{dz}=\frac{1}{2}\left[\left(\frac{\partial u}{\partial x}+i\frac{\partial v}{\partial x}\right)-i\left(\frac{\partial u}{\partial y}+i\frac{\partial v}{\partial y}\right)\right]=\frac{1}{2}\left[\left(\frac{\partial u}{\partial x}+i\frac{\partial v}{\partial x}\right)+\left(-i\frac{\partial u}{\partial y}+\frac{\partial v}{\partial y}\right)\right]$$

$$= \frac{1}{2}\left[\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) + \left(-i\frac{\partial u}{\partial y} + i\frac{\partial v}{\partial x}\right)\right]. \tag{3.23}$$

So, the partial derivatives of complex-valued Elliot function , $\frac{\partial u}{\partial x}, \frac{\partial v}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial v}{\partial y}$, can be expressed as follows:

$$u_x = \begin{cases} \dfrac{\left(y^2 + |z|\right)}{|z|(1+|z|)^2} & \text{if } |z| \neq 0 \\ 1 & |z| = 0 \end{cases} \tag{3.24}$$

$$u_y = \begin{cases} \dfrac{-xy}{|z|(1+|z|)^2} & \text{if } |z| \neq 0 \\ 0 & |z| = 0 \end{cases} \tag{3.25}$$

$$v_x = \begin{cases} \dfrac{-xy}{|z|(1+|z|)^2} & \text{if } |z| \neq 0 \\ 0 & |z| = 0 \end{cases} \tag{3.26}$$

$$v_y = \begin{cases} \dfrac{\left(x^2 + |z|\right)}{|z|(1+|z|)^2} & \text{if } |z| \neq 0 \\ 1 & |z| = 0 \end{cases} \tag{3.27}$$

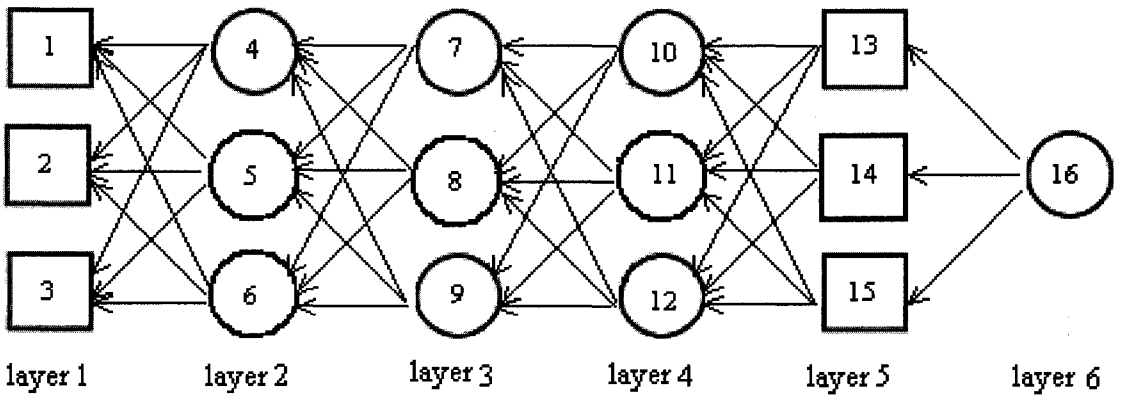


Fig. 3. 6 ANCFIS backpropagation network

Based on the knowledge of differentiating complex functions, we will derive the procedures of this backpropagation learning rule in the rest of this section. By reversing the links of the forward pass network and supplying the error signals at the output layer as inputs to the new network. Thus, the error propagation network used in the backward pass is obtained, in which the errors propagate from the last output node towards the inputs. The backpropagation network is shown in Fig. 3.6, where for the sake of clarity, a unique number is assigned to each node.

3.3.2 Gradient-based Learning Rule

The sum of the squared error is used as the error measure for the *pth* $(1 \le p \le P)$ entry of the training data:

$$E_p = \sum_{q=1}^{N(L)} \left(d_q - x_{L,q}\right)^2, \tag{3.28}$$

where L refers to the number of the layers, d_q is the *qth* component of the *pth* desired output vector, and $x_{L,q}$ is the *qth* component of the actual output vector generated by supplying the *pth* input vector to the network. We can obtain the p th error measure E_p from each training pair in our training data. The goal of the ANCFIS is to minimize the total error measure

$$E = \sum_p E_p \,. \tag{3.29}$$

This can be achieved by changing the parameters in the node functions. In order to facilitate the discussion the 'error signal', $\varepsilon_{l,i}$ is defined as the derivative of the error

measure E_p with respect to the output of node i in layer l, taking both direct and indirect paths into account:

$$\varepsilon_{l,i} = \frac{\partial^+ E_p}{\partial x_{l,i}} \tag{3.30}$$

The error signal for the i th output node (at layer L) can be calculated directly

$$\varepsilon_{L,i} = \frac{\partial^+ E_p}{\partial x_{L,i}} = \frac{\partial E_p}{\partial x_{L,i}} = -2(d_i - x_{L,i}). \tag{3.31}$$

For every internal (non output) node at the i th position in layer l, the error signal can be represented as a linear combination of error signals from layer $l+1$. This means that all error signals can be calculated iteratively from the output towards the input layer.

$$\varepsilon_{l,i} = \frac{\partial^+ E_p}{\partial x_{l,i}} = \sum_{l=1}^{N(m+1)} \frac{\partial^+ E_p}{\partial x_{l+1,m}} \frac{\partial f_{l+1,m}}{\partial x_{l,i}} = \sum_{l=1}^{N(m+1)} \varepsilon_{l+1,m} \frac{\partial f_{l+1,m}}{\partial x_{l,i}}, \tag{3.32}$$

where $1 \le l \le L-1$,and m refers to the index of node in the layer $l+1$.

Once all of the $\varepsilon_{l,i}$'s have been obtained, it is straightforward to calculate the gradient for a generic weight *SMF* sampled from the membership function in node i of the first layer. Suppose that the output $x_{1,i}$ of the neuron i in the first layer of the network is

$$x_{1,i} = f(z_{1,i}) = u_{1,i} + iv_{1,i},\ z_{1,i} = x^{1,i} + iy^{1,i} = \sum_{j=1}^{n} \left(x_{0,j} \times \sum_{k=1}^{n} SMF_{ikj} \right), \tag{3.33}$$

In what follows, the subscripts R and I indicate the real and imaginary parts, respectively. We note the following partial derivatives:

$$\frac{\partial x^{1,i}}{\partial SMF_{ikjR}} = x_{0,j}, \frac{\partial y^{1,i}}{\partial SMF_{ikjR}} = 0, \frac{\partial x^{1,i}}{\partial SMF_{ikjI}} = 0, \frac{\partial y^{1,i}}{\partial SMF_{ikjI}} = x_{0,j}. \tag{3.34}$$

In order to use the chain rule to find the gradient of the error function E_p with respect to the real part of SMF_{ikj}, we have to observe the variable dependencies: the real function E_p is a function of both $u_{1,i}(x^{1,i}, y^{1,i})$ and $v_{1,i}(x^{1,i}, y^{1,i})$, and $x^{1,i}$ and $y^{1,i}$ are both functions of SMF_{ikjR} and SMF_{ikjI}. Thus, we can rewrite the gradient of the error function with respect to the real part of SMF_{ikjR} as

$$\frac{\partial^+ E_p}{\partial SMF_{ikjR}} = \frac{\partial^+ E_p}{\partial u_{1,i}}\left(\frac{\partial u_{1,i}}{\partial x^{1,i}}\frac{\partial x^{1,i}}{\partial SMF_{ikjR}} + \frac{\partial u_{1,i}}{\partial y^{1,i}}\frac{\partial y^{1,i}}{\partial SMF_{ikjR}}\right)$$

$$+\frac{\partial^+ E_p}{\partial v_{1,i}}\left(\frac{\partial v_{1,i}}{\partial x^{1,i}}\frac{\partial x^{1,i}}{\partial SMF_{ikjR}} + \frac{\partial v_{1,i}}{\partial y^{1,i}}\frac{\partial y^{1,i}}{\partial SMF_{ikjR}}\right)$$

$$=\left(\frac{\partial^+ E_p}{\partial u_{1,i}}\frac{\partial u_{1,i}}{\partial x^{1,i}} + \frac{\partial^+ E_p}{\partial v_{1,i}}\frac{\partial v_{1,i}}{\partial x^{1,i}}\right)x_{0,j}, \tag{3.35}$$

where $\frac{\partial^+ E_p}{\partial u_{1,i}}$ and $\frac{\partial^+ E_p}{\partial v_{1,i}}$ are the real part and the imaginary part of the error signals of node i in the first layer, separately. $\frac{\partial u_{1,i}}{\partial x^{1,i}}$ is the partial derivative of the real part of the Ellitott function with respect to the real part of summation of convolution. $\frac{\partial v_{1,i}}{\partial x^{1,i}}$ is the partial derivative of the imaginary part of the activation function with respect to the real part of summation of convolution. Likewise, the gradient of the error function with respect to the imaginary part SMF_{ikjI} is

$$\frac{\partial^+ E_p}{\partial SMF_{ikjI}} = \frac{\partial^+ E_p}{\partial u_{1,i}}\left(\frac{\partial u_{1,i}}{\partial x^{1,i}}\frac{\partial x^{1,i}}{\partial SMF_{ikjI}} + \frac{\partial u_{1,i}}{\partial y^{1,i}}\frac{\partial y^{1,i}}{\partial SMF_{ikjI}}\right)$$

$$+\frac{\partial^+ E_p}{\partial v_{1,i}}\left(\frac{\partial v_{1,i}}{\partial x^{1,i}}\frac{\partial x^{1,i}}{\partial SMF_{ikjI}}+\frac{\partial v_{1,i}}{\partial y^{1,i}}\frac{\partial y^{1,i}}{\partial SMF_{ikjI}}\right)$$

$$=\left(\frac{\partial^+ E_p}{\partial u_{1,i}}\frac{\partial u_{1,i}}{\partial y^{1,i}}+\frac{\partial^+ E_p}{\partial v_{1,i}}\frac{\partial v_{1,i}}{\partial y^{1,i}}\right)x_{0,j}\ , \tag{3.36}$$

combining (3.35) and (3.36), we can write the gradient of error function E_p with respect to the complex sample SMF_{ikj} as

$$\frac{\partial^+ E_p}{\partial SMF_{ikj}}=\frac{\partial^+ E_p}{\partial SMF_{ikjR}}+i\frac{\partial^+ E_p}{\partial SMF_{ikjI}}$$

$$=\left(\frac{\partial^+ E_p}{\partial u_{1,i}}\frac{\partial u_{1,i}}{\partial x^{1,i}}+\frac{\partial^+ E_p}{\partial v_{1,i}}\frac{\partial v_{1,i}}{\partial x^{1,i}}\right)x_{0,j}+i\left(\frac{\partial^+ E_p}{\partial u_{1,i}}\frac{\partial u_{1,i}}{\partial y^{1,i}}+\frac{\partial^+ E_p}{\partial v_{1,i}}\frac{\partial v_{1,i}}{\partial y^{1,i}}\right)x_{0,j} \tag{3.37}$$

Once all of the $\varepsilon_{l,i}$'s have been obtained, it is straightforward to calculate the update for a generic weight SMF_{ikj},

$$\frac{\partial^+ E_p}{\partial SMF_{ikj}}=\varepsilon_{1,i}\frac{\partial f_{1,i}}{\partial SMF_{ikj}}, \tag{3.38}$$

$$\frac{\partial^+ E}{\partial SMF_{ikj}}=\sum_{p=1}^{P}\frac{\partial^+ E_p}{\partial SMF_{ikj}}\ . \tag{3.39}$$

where $f_{1,i}$ denotes the function of node i of the first layer. By using the steepest descent, the update formula for the generic complex sample is

$$\Delta SMF_{ikj}=-\eta\frac{\partial^+ E}{\partial SMF_{ikj}}=-\eta\sum_{p=1}^{P}\frac{\partial^+ E_p}{\partial SMF_{ikj}}$$

$$=-\frac{k}{\sqrt{\sum_{SMF_{ikj}}\left(\left(\mathrm{Re}\left(\frac{\partial^+ E}{\partial SMF_{ikj}}\right)\right)^2+\left(\mathrm{Im}\left(\frac{\partial^+ E}{\partial SMF_{ikj}}\right)\right)^2\right)}}\sum_{p=1}^{P}\frac{\partial^+ E_p}{\partial SMF_{ikj}}\ , \tag{3.40}$$

where η is the learning rate, k is the step size, the length of each transition along the gradient direction in the parameter space. Generally speaking, when the step size is updated, the speed of convergence can be modified.

Usually we can change the step size to adjust the convergence speed, such as using the following heuristic rules[5][6]:

1. If the parameter undergoes m consecutive reductions, increases k by

$$k = k * increase\ rate$$

namely, increase the step size after m downs

2. If the parameter varies consecutively n times of combinations of one reduction and one increase, decreases k by

$$k = k * decrease\ rate$$

namely, decrease the step size after n combinations of 1 up and 1 down.

Thus, we have

$$SMF_{ikj\,new} = SMF_{ikj\,old} + \Delta SMF_{ikj} = SMF_{ikj\,old} - \eta \frac{\partial^+ E}{\partial SMF_{ikj}}$$

$$= SMF_{ikj\,old} - \frac{k}{\sqrt{\sum_{SMF_{ikj}} \left(\left(\mathrm{Re}\left(\frac{\partial^+ E}{\partial SMF_{ikj}} \right) \right)^2 + \left(\mathrm{Im}\left(\frac{\partial^+ E}{\partial SMF_{ikj}} \right) \right)^2 \right)}} \sum_{p=1}^{P} \frac{\partial^+ E_p}{\partial SMF_{ikj}} \tag{3.41}$$

For simplicity and clarity, taking an ANCFIS with two CMFs as an example(see Fig. 3.7), we start with the calculation of the derivative of the error measure in (3.31) at node 11.

From the output to layer 6:

$$\varepsilon_{11} = \frac{\partial^+ E_p}{\partial x_{11}} = -2(d_{11} - x_{11}) = -2\left(d_{11} - \sum w_i^{DP} f_i\right),\ i = 1,2 \tag{3.42}$$

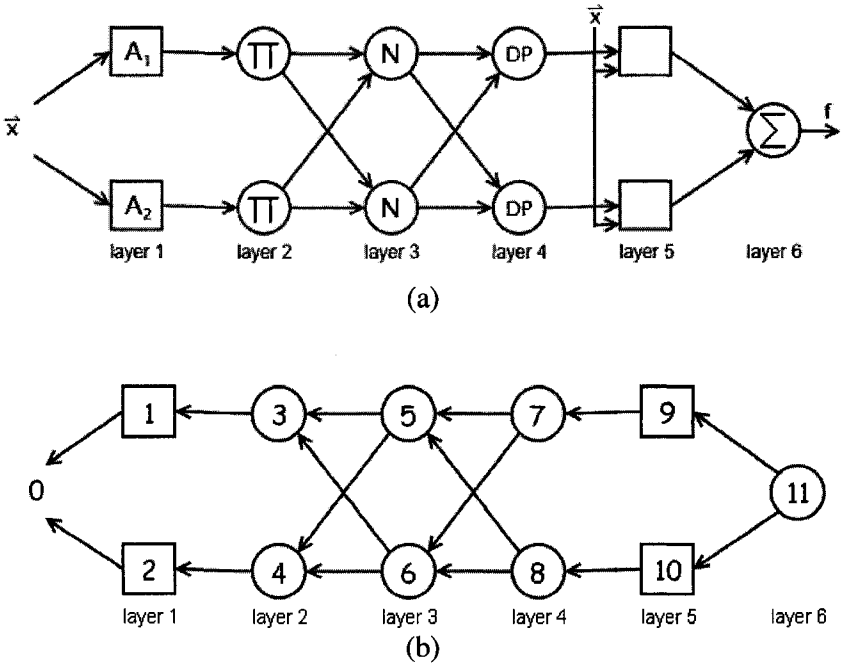


Fig. 3. 7 A six-layer ANCFIS structure with 2 complex membership functions (a) forward pass; (b) ANCFIS backpropagation network

From layer 6 to layer 5:

$$\varepsilon_{10} = \frac{\partial E_p}{\partial x_{10}} = \frac{\partial E_p}{\partial x_{11}} \frac{\partial f_{11}}{\partial x_{10}} = \varepsilon_{11} \frac{\partial f_{11}}{\partial x_{10}} = \varepsilon_{11} \frac{\partial\left(\sum w_i^{DP} f_i\right)}{\partial\left(\overline{w}_2 f_2\right)} = \varepsilon_{11},\ i = 1,2 \tag{3.43}$$

$$\varepsilon_9 = \frac{\partial E_p}{\partial x_9} = \frac{\partial E_p}{\partial x_{11}} \frac{\partial f_{11}}{\partial x_9} = \varepsilon_{11} \frac{\partial f_{11}}{\partial x_9} = \varepsilon_{11} \frac{\partial\left(\sum w_i^{DP} f_i\right)}{\partial\left(\overline{w}_1 f_1\right)} = \varepsilon_{11},\ i = 1,2 \tag{3.44}$$

From layer 5 to layer 4:

$$\varepsilon_8 = \frac{\partial E_p}{\partial x_8} = \frac{\partial E_p}{\partial x_{10}} \frac{\partial f_{10}}{\partial x_8} = \varepsilon_{10} \frac{\partial f_{10}}{\partial x_8} = \varepsilon_{10} \frac{\partial\left(w_2^{DP} f_2\right)}{\partial\left(w_2^{DP}\right)} = \varepsilon_{10} f_2 = \varepsilon_{11} f_2 \tag{3.45}$$

$$\varepsilon_7 = \frac{\partial E_p}{\partial x_7} = \frac{\partial E_p}{\partial x_9} \frac{\partial f_9}{\partial x_7} = \varepsilon_9 \frac{\partial f_9}{\partial x_7} = \varepsilon_9 \frac{\partial\left(w_1^{DP} f_1\right)}{\partial\left(w_1^{DP}\right)} = \varepsilon_9 f_1 = \varepsilon_{11} f_1 \tag{3.46}$$

From layer 4 to layer 3:

$$\begin{aligned}\varepsilon_6 &= \frac{\partial E_p}{\partial x_6} = \frac{\partial E_p}{\partial x_8}\frac{\partial f_8}{\partial x_6} + \frac{\partial E_p}{\partial x_7}\frac{\partial f_7}{\partial x_6} = \varepsilon_8 \frac{\partial f_8}{\partial x_6} + \varepsilon_7 \frac{\partial f_7}{\partial x_6} = \varepsilon_8 \frac{\partial w_2^{DP}}{\partial(\overline{w}_2)} + \varepsilon_7 \frac{\partial w_1^{DP}}{\partial(\overline{w}_2)} \\ &= \varepsilon_8 \frac{\partial[\overline{w}_2 \bullet (\overline{w}_1 + \overline{w}_2)]}{\partial(\overline{w}_2)} + \varepsilon_7 \frac{\partial[\overline{w}_1 \bullet (\overline{w}_1 + \overline{w}_2)]}{\partial(\overline{w}_2)}\end{aligned} \tag{3.47}$$

$$\begin{aligned}\varepsilon_5 &= \frac{\partial E_p}{\partial x_5} = \frac{\partial E_p}{\partial x_8}\frac{\partial f_8}{\partial x_5} + \frac{\partial E_p}{\partial x_7}\frac{\partial f_7}{\partial x_5} = \varepsilon_8 \frac{\partial f_8}{\partial x_5} + \varepsilon_7 \frac{\partial f_7}{\partial x_5} = \varepsilon_8 \frac{\partial w_2^{DP}}{\partial(\overline{w}_1)} + \varepsilon_7 \frac{\partial w_1^{DP}}{\partial(\overline{w}_1)} \\ &= \varepsilon_8 \frac{\partial[\overline{w}_2 \bullet (\overline{w}_1 + \overline{w}_2)]}{\partial(\overline{w}_1)} + \varepsilon_7 \frac{\partial[\overline{w}_1 \bullet (\overline{w}_1 + \overline{w}_2)]}{\partial(\overline{w}_1)}\end{aligned} \tag{3.48}$$

Assume that $\overline{w}_1 = x_1 + iy_1$, $\overline{w}_2 = x_2 + iy_2$. The derivative of complex functions in (3.47) yields:

$$\begin{aligned}\varepsilon_6 &= \varepsilon_8 \frac{\partial(\overline{w}_2 \bullet \overline{w}_2 + \overline{w}_2 \bullet \overline{w}_1)}{\partial \overline{w}_2} + \varepsilon_7 \frac{\partial(\overline{w}_1 \bullet \overline{w}_1 + \overline{w}_1 \bullet \overline{w}_2)}{\partial \overline{w}_2} \\ &= \varepsilon_8 \frac{\partial[(x_1x_2 + y_1y_2) + (x_2x_2 + y_2y_2)]}{\partial(x_2 + iy_2)} + \varepsilon_7 \frac{\partial[(x_1x_2 + y_1y_2) + (x_1x_1 + y_1y_1)]}{\partial(x_2 + iy_2)} \\ &= \frac{1}{2}\varepsilon_8((x_1 + 2x_2) + i(y_1 + 2y_2)) + \frac{1}{2}\varepsilon_7(x_1 - iy_1) \\ &= \frac{1}{2}\varepsilon_8(\overline{\overline{w}_1} + \overline{\overline{w}_2}) + \frac{1}{2}\varepsilon_8\overline{\overline{w}_2} + \frac{1}{2}\varepsilon_7\overline{\overline{w}_1}\end{aligned} \tag{3.49}$$

where $\overline{\overline{w}_i}$ indicates the complex conjugate of the normalized weight. The derivative of complex functions in (3.48) yields:

$$\begin{aligned}\varepsilon_5 &= \varepsilon_8 \frac{\partial(\overline{w}_2 \bullet \overline{w}_1 + \overline{w}_2 \bullet \overline{w}_2)}{\partial \overline{w}_1} + \varepsilon_7 \frac{\partial(\overline{w}_1 \bullet \overline{w}_1 + \overline{w}_1 \bullet \overline{w}_2)}{\partial \overline{w}_1} \\ &= \varepsilon_8 \frac{\partial[(x_2x_1 + y_2y_1) + (x_2x_2 + y_2y_2)]}{\partial(x_1 + iy_1)} + \varepsilon_7 \frac{\partial[(x_1x_1 + y_1y_1) + (x_1x_2 + y_1y_2)]}{\partial(x_1 + iy_1)} \\ &= \frac{1}{2}\varepsilon_8(x_2 - iy_2) + \frac{1}{2}\varepsilon_7((2x_1 + x_2) - i(2y_1 + y_2)) \\ &= \frac{1}{2}\varepsilon_8(\overline{\overline{w}_2}) + \frac{1}{2}\varepsilon_7(\overline{\overline{w}_1} + \overline{\overline{w}_2}) + \frac{1}{2}\varepsilon_7\overline{\overline{w}_1}\end{aligned} \tag{3.50}$$

From layer 3 to layer 2:

$$\varepsilon_4 = \frac{\partial E_p}{\partial x_4} = \frac{\partial E_p}{\partial x_6}\frac{\partial f_6}{\partial x_4} + \frac{\partial E_p}{\partial x_5}\frac{\partial f_5}{\partial x_4} = \varepsilon_6 \frac{\partial f_6}{\partial x_4} + \varepsilon_5 \frac{\partial f_5}{\partial x_4} = \varepsilon_6 \frac{\partial \overline{w}_2}{\partial (w_2)} + \varepsilon_5 \frac{\partial \overline{w}_1}{\partial (w_2)}$$
$$= \varepsilon_6 \frac{\partial \left[w_2 \Big/ \left(|w_1| + |w_2| \right) \right]}{\partial (w_2)} + \varepsilon_5 \frac{\partial \left[w_1 \Big/ \left(|w_1| + |w_2| \right) \right]}{\partial (w_2)} \tag{3.51}$$

$$\varepsilon_3 = \frac{\partial E_p}{\partial x_3} = \frac{\partial E_p}{\partial x_6}\frac{\partial f_6}{\partial x_3} + \frac{\partial E_p}{\partial x_5}\frac{\partial f_5}{\partial x_3} = \varepsilon_6 \frac{\partial f_6}{\partial x_3} + \varepsilon_5 \frac{\partial f_5}{\partial x_3} = \varepsilon_6 \frac{\partial \overline{w}_2}{\partial (w_1)} + \varepsilon_5 \frac{\partial \overline{w}_1}{\partial (w_1)}$$
$$= \varepsilon_6 \frac{\partial \left[w_2 \Big/ \left(|w_1| + |w_2| \right) \right]}{\partial (w_1)} + \varepsilon_5 \frac{\partial \left[w_1 \Big/ \left(|w_1| + |w_2| \right) \right]}{\partial (w_1)} \tag{3.52}$$

Assume that $w_1 = x_1 + iy_1$, $w_2 = x_2 + iy_2$. The use of derivative formula of complex function in (3.51) yields:

$$\varepsilon_4 = \varepsilon_6 \frac{\partial \left[\dfrac{(x_2 + iy_2)}{\sqrt{x_1^2 + y_1^2} + \sqrt{x_2^2 + y_2^2}} \right]}{\partial (x_2 + iy_2)} + \varepsilon_5 \frac{\partial \left[\dfrac{(x_1 + iy_1)}{\sqrt{x_1^2 + y_1^2} + \sqrt{x_2^2 + y_2^2}} \right]}{\partial (x_2 + iy_2)} \tag{3.53}$$

The partial derivatives can be calculated separately as follows:

let $\dfrac{x_2 + iy_2}{M} = u_2 + iv_2$, where $u_2 = \dfrac{x_2}{M}$ and $v_2 = \dfrac{y_2}{M}$ and $M = \sqrt{x_1^2 + y_1^2} + \sqrt{x_2^2 + y_2^2}$,

then,

$$\frac{\partial u_2}{\partial x_2} = \frac{M - \dfrac{x_2^2}{\sqrt{x_2^2 + y_2^2}}}{M^2}, \quad \frac{\partial u_2}{\partial y_2} = \frac{-x_2 y_2}{M^2 \sqrt{x_2^2 + y_2^2}} = \frac{\partial v_2}{\partial x_2} \text{ and } \frac{\partial v_2}{\partial y_2} = \frac{M - \dfrac{y_2^2}{\sqrt{x_2^2 + y_2^2}}}{M^2}.$$

$$\frac{\partial \left[\dfrac{(x_2 + iy_2)}{\sqrt{x_1^2 + y_1^2} + \sqrt{x_2^2 + y_2^2}} \right]}{\partial (x_2 + iy_2)} = \frac{1}{2}\left[\left(\frac{\partial u_2}{\partial x_2} + \frac{\partial v_2}{\partial y_2} \right) + \left(-i\frac{\partial u_2}{\partial y_2} + i\frac{\partial v_2}{\partial x_2} \right) \right].$$

And let $\frac{x_1+iy_1}{M}=u_1+iv_1$ where $u_1=\frac{x_1}{M}$, $v_1=\frac{y_1}{M}$ and $M=\sqrt{x_1^2+y_1^2}+\sqrt{x_2^2+y_2^2}$,

then,

$$\frac{\partial u_1}{\partial x_2}=\frac{-x_1x_2}{M^2\sqrt{x_2^2+y_2^2}},\ \frac{\partial u_1}{\partial y_2}=\frac{-x_1y_2}{M^2\sqrt{x_2^2+y_2^2}},\ \frac{\partial v_1}{\partial x_2}=\frac{-y_1x_2}{M^2\sqrt{x_2^2+y_2^2}}\ \text{and}$$

$$\frac{\partial v_1}{\partial y_2}=\frac{-y_1y_2}{M^2\sqrt{x_2^2+y_2^2}}.$$

$$\frac{\partial\left[\dfrac{(x_1+iy_1)}{\sqrt{x_1^2+y_1^2}+\sqrt{x_2^2+y_2^2}}\right]}{\partial(x_2+iy_2)}=\frac{1}{2}\left[\left(\frac{\partial u_1}{\partial x_2}+\frac{\partial v_1}{\partial y_2}\right)+\left(-i\frac{\partial u_1}{\partial y_2}+i\frac{\partial v_1}{\partial x_2}\right)\right].$$

Continuing with this, we can obtain

$$\frac{\partial u_2}{\partial x_1}=\frac{-x_1x_2}{M^2\sqrt{x_1^2+y_1^2}},\ \frac{\partial u_2}{\partial y_1}=\frac{-y_1x_2}{M^2\sqrt{x_1^2+y_1^2}},\ \frac{\partial v_2}{\partial x_1}=\frac{-x_1y_2}{M^2\sqrt{x_1^2+y_1^2}},\ \frac{\partial v_2}{\partial y_1}=\frac{-y_1y_2}{M^2\sqrt{x_1^2+y_1^2}}$$

and

$$\frac{\partial u_1}{\partial x_1}=\frac{M-\dfrac{x_1^2}{\sqrt{x_1^2+y_1^2}}}{M^2},\ \frac{\partial u_1}{\partial y_1}=\frac{-x_1y_1}{M^2\sqrt{x_1^2+y_1^2}}=\frac{\partial v_1}{\partial x_1},\ \frac{\partial v_1}{\partial y_1}=\frac{M-\dfrac{y_1^2}{\sqrt{x_1^2+y_1^2}}}{M^2}.$$

Finally, we have

$$\varepsilon_4=\varepsilon_6\frac{1}{2}\left[\left(\frac{\partial u_2}{\partial x_2}+\frac{\partial v_2}{\partial y_2}\right)+i\left(-\frac{\partial u_2}{\partial y_2}+\frac{\partial v_2}{\partial x_2}\right)\right]+\varepsilon_5\frac{1}{2}\left[\left(\frac{\partial u_1}{\partial x_2}+\frac{\partial v_1}{\partial y_2}\right)+i\left(-\frac{\partial u_1}{\partial y_2}+\frac{\partial v_1}{\partial x_2}\right)\right] \quad (3.54)$$

$$\varepsilon_3=\varepsilon_6\frac{1}{2}\left[\left(\frac{\partial u_2}{\partial x_1}+\frac{\partial v_2}{\partial y_1}\right)+i\left(-\frac{\partial u_2}{\partial y_1}+\frac{\partial v_2}{\partial x_1}\right)\right]+\varepsilon_5\frac{1}{2}\left[\left(\frac{\partial u_1}{\partial x_1}+\frac{\partial v_1}{\partial y_1}\right)+i\left(-\frac{\partial u_1}{\partial y_1}+\frac{\partial v_1}{\partial x_1}\right)\right] \quad (3.55)$$

$$\varepsilon_2=\frac{\partial E_p}{\partial x_2}=\frac{\partial E_p}{\partial x_4}\frac{\partial f_4}{\partial x_2}=\varepsilon_4\frac{\partial f_4}{\partial x_2}=\varepsilon_4\frac{\partial(w_2)}{\partial(w_2)}=\varepsilon_4 \quad (3.56)$$

$$\varepsilon_1 = \frac{\partial E_p}{\partial x_1} = \frac{\partial E_p}{\partial x_3}\frac{\partial f_3}{\partial x_1} = \varepsilon_3 \frac{\partial f_3}{\partial x_1} = \varepsilon_3 \frac{\partial(w_1)}{\partial(w_1)} = \varepsilon_3 \tag{3.57}$$

Once we get all of the ε_i's, we can straightforwardly calculate the gradient for a generic weight SMF_{ikj} in node i according to Eq. (3.38) and 3.39)

3.3.3 Mathematical Model of Premise Parameter Identification

How to update the real valued premise parameters $\{a,b,c,d\}$ of the complex membership function is a unique problem. In ANFIS, we simply add the negative gradient of the overall error measure E with respect to adaptive parameters of membership functions multiplied by the learning rate to the parameters to be updated. In ANCFIS, since the derivative of the overall error measure E with respect to the adaptive parameters $\{a,b,c,d\}$is complex, while adaptive parameters are real-valued, the update of adaptive parameters of complex membership functions could not be directly achieved. In terms of Eq. (3.4) and (3.5), after the magnitude of phase of each weight $SMF_{ikj\,new}$ are calculated, we have a set of m magnitude-phase data pairs for each complex membership function, $m = 1,2,\ldots,n^2$. Now what we need to do is to find an optimization method for fitting a set of magnitude-phase data pairs related to the adaptive parameters. That means the unknown adaptive parameters $\{a,b,c,d\}$ can be indirectly identified using the optimization method.

We hope to optimize a model by minimizing a squared error measure between magnitudes of the updated weights and the fitting value of complex membership function at given phases corresponding to the magnitudes of the updated weights, defined by:

$$E(x_1,x_2,x_3,x_4) = \sum_{m=1}^{n^2} (magnitude_m - (x_4 * \sin(x_1 * phase_m + x_2) + x_3))^2 , \tag{3.58}$$

where x_1, x_2, x_3, x_4 refer to the premise parameters a, b, c, d, respectively, n is the length of input vector to the system, $Magnitude_m$ and $phase_m$ are the magnitude-phase data pair of the updated weight $SMF_{ikj_{new}}$, $k = 1,2,\ldots,n$, $j = 1,2,\ldots,n$, $m = 1,2,\ldots,n^2$, and n^2 is the total number of the weights.

Seen from Eq. (3.58), this optimization objective function belongs to the class of nonlinear least square problems, which may be solved using Levenberg-Marquardt method [75], since this optimization technique can handle well ill-conditioned matrices. However, this method needs to derive the Jacobian matrix of the difference between the desired output, $magnitude_m$, and the model's output, $x_4 * \sin(x_1 * phase_m + x_2) + x_3$. In addition, this method is deterministic in the sense that it inevitably leads to convergence to the nearest local minimum close to the initial points. In practice, however, knowing good starting points is nearly impossible. We have elected instead to employ derivative-free optimization. Therefore, we will propose a derivative-free optimization technique, VNCSA, to identify adaptive parameters of complex membership functions in the first layer of ANCFIS architecture. In the next section, we describe the novel VNCSA optimization technique, which we developed specifically for this application.

3.4 Variable neighborhood chaotic simulated annealing algorithm

The development of a novel derivative-free algorithm aims to solve nonlinear constrained optimization problems. The structure of the most constrained optimization problems is essentially contained in the following:

minimize $f(S)$ $\quad S \in R^n$,

subject to $c_i(S)=0,\ i\in E,$

$$c_i(S)\geq 0,\ i\in I, \tag{3.59}$$

where $f(S)$ is the objective function, S is the control variable, $c_i(S)$ are additional constraint functions, $i=1,2,\ldots,p$. E is the index set of equations or equality constraints in the problem. I is the set of inequality constraints, and both of these sets are finite.

3.4.1 VNCSA algorithm

VNCSA is based on the Simulated Annealing (SA) algorithm. We hybridize the SA algorithm with two 1-dimensional chaotic maps and a variable neighborhood strategy for finding global minima. VNCSA still simulates the cooling of a physical system whose possible energies correspond to the values of the objective function to be minimized, and allows solution candidates of worse quality than the current solution (uphill moves) in order to escape from local minima. The probability of an uphill move is reduced over time during the search. The algorithm starts by producing a group of feasible solutions satisfying all given constraints, by iterating the Logistic map from a random initial point with the parameter μ=4.0 (fully chaotic behavior). As the Logistic map has a domain and codomain of [0,1], we will in general have to execute a linear transform from [0,1] to the domain of the individual control variables s_i. Candidate solutions that do not satisfy the constraints are simply rejected, leaving only feasible solutions, of which the candidate having the lowest value of the objective function is selected as the initial solution. The annealing temperature parameter is also initialized from the objective values of the feasible candidate solutions. Then the following steps are repeated until the stopping condition is met: 1) A solution S_{new} from the variable neighborhood $N(S)$ of the solution

$S_{current}$ is generated using the Ulam-von Neumann map; 2) S_{new} is checked for feasibility, and discarded if it does not satisfy the constraints; 3) $f(S_{new}), f(S_{current})$ and temperature T are employed to determine if S_{new} is accepted as the new current solution.

3.4.1.1 Generating initial solution population

In order to avoid choosing the fixed points of the Logistic map as initial solutions, we use a random generator to generate N uniform variates in the interval (0,1) as the initial values of a set of chaotic variables, $x_{i,0}$, $i = 1,2,\ldots,N$, where N is the number of variables of the objective function. We then substitute $x_{i,0}$ into the Logistic map (Eq. (2.56)) with $\mu = 4.0$ to obtain a new set of chaotic variables, $x_{i,j+1}$, expressed by

$$x_{i,j+1} = 4.0 \times x_{i,j} \times (1 - x_{i,j}), \tag{3.60}$$

$$x_{i,j} = x_{i,j+1}, \tag{3.61}$$

where i is the index of variables of the objective function, and j is the iteration count of logistic map.

We convert the chaotic variables $x_{i,j+1}$ into the variables $s_{i,j+1}$ of the objective function using a linear transform:

$$s_{i,j+1} = a_i + (b_i - a_i) \times x_{i,j+1}, \tag{3.62}$$

where a_i, b_i are the lower and upper bounds of the variables $s_{i,j+1}$ of the objective function, respectively. Thus, a legal $s_{i,j+1}$ between a_i and b_i is generated through Eq. (3.62). This process is repeated until j reaches the maximum iteration count M, then set the current solution $S_{current}$ to be the particular solution having the best value of the objective function.

3.4.1.2 Initial temperature and final temperature

The initial temperature is selected so that the probability for an uphill-move is quite high at the beginning of the algorithm. An appropriate initial temperature T_0 is one that leads to an average increase of acceptance probability p_0 of about 0.8 [76]. This value of T_0 will be completely dependent on the scaling of objection function $f(S)$; it can be estimated by calculating the average objective value increase observed on the initial solution population Δf^+, giving us

$$T_0 = -\Delta f^+ / \ln(p_0), \tag{3.63}$$

In VNCSA, the final temperature is defined as a fixed value, 0.01.

3.4.1.3 Neighbor selection

VNCSA gnerates new trial solutions S^{new} based on the current feasible solution $S^{current}$ by introducing small changes through the Ulam-von Neumann map, defined by.

$$S_{i,j+1}^{new} = S_i^{current} + D_i y_{i,j+1}, \tag{3.64}$$

$$y_{i,j+1} = 1 - 2y_{i,j}^2, \tag{3.65}$$

$$y_{i,j} = y_{i,j+1}, \tag{3.66}$$

where $y_{i,j}$ is the chaotic variable of the Ulam-von Neumann map starting from random initial values in (0,1) [55], i is the index of variable of objective function, $i = 1,2,\ldots,N$, N is the number of variables of the objective function, j is the iteration count of the Ulam-von Neumann map, $j = 0,1,\ldots M$, M is the maximum iteration count of the Ulam-von Neumann map, and D_i defines the maximum change allowed in each control variable (the neighborhood). The initial neighborhood is defined by

$$D_i = 0.1 \times (b_i - a_i), \tag{3.67}$$

where a_i, b_i are the lower and upper bounds of the control variables. The neighborhood D_i tunes the step size in Eq. (3.64) for each variable of the objective function. After $L_{\max} \times M$ trials (L is the maximum value of iteration count l at each temperature, and M is the maximum iterations of the Ulam-von Neumann map at each iteration count l), to ensure a more efficient search space, the neighborhood D [43] is updated by

$$D_i^{new} = (1-\alpha) D_i^{current} + \alpha \omega R_i, \tag{3.68}$$

where R_i is the magnitude of the successful change made to each control variable, α is the damping constant controlling the rate at which information from R_i is folded into D_i with weighting ω. The values of α control the behavior of the VNCSA algorithm.

Generally speaking, the determination of the optimal values of two parameters $L_{\max}$ and M is application dependent, especially on the dimensions and constraint intervals of the variables of the objective functions to be minimized. The best way to select their values is by doing some trial-and-error experiments. To enable this algorithm to converge to near optimal solutions for large-scale multi-peak nonlinear constraint optimization problems, one way to do this is to make the parameters $L_{\max}$ and M random variables within appropriate ranges, such that the algorithm can explore a wider area of the input domain based on the chaotic series between -1 and 1 generated by Ulam-von-Neumann chaotic map. If the number of variables and the intervals between the upper bounds and lower bounds of variables are lager, the maximum value of the iteration count l, $L_{\max}$, and the value of the iteration count of Ulam-von Neumann chaotic map, M,

need to be initialized to larger values to ensure the algorithm to evaluate the more feasible solutions.

3.4.1.4 Temperature update

The annealing schedule specifies the degree of uphill movement allowed during the search. The simplest and most often used temperature decrement rule is the exponential cooling scheme [5][42] , expressed by

$$T_{k+1} = \beta T_k , \tag{3.69}$$

where β is a constant close to, but less than,1. It is to be noticed that the value of objective function is generally improved with larger β at the cost of expensive computational effort.

3.4.1.5 The procedures on VNCSA algorithm

Begin

$S_{current}$:= GenerateInitialSolutionPopulationAtLogisticMap() ***(Section 3.4.1.1)***

T_0 := SetInitializationTemperature() ***(Section 3.4.1.2)***

While ($T_k < 0.01$) do

While($l \le L_{\max}$) do

While($m < M$)

S_{new} := PickNeighborAtUlam_von_NeumannMap() ***(Section 3.4.1.3)***

if $f(S_{new}) < f(S_{current})$ **then**

$S_{current} := S_{new}$

else

accept S_{new} as new solution with probability

$$\exp(-(f(S_{new}) - f(S_{current}))/T_k)$$

end if

end while

end while

UpdateNeighborhood(D) ***(Section 3.4.1.3)***

UpdateTemperature(T_k) ***(Section 3.4.1.4)***

end while

$S_{best} := S_{current}$

output: S_{best} viewed as optimization solution for x

End

3.4.2 Parameter Selection and Comparison on Benchmark Test Functions

In this section, we test the VNCSA algorithm on several benchmark optimization problems. Our purpose is twofold: first, we want to determine if VNCSA performs as well as CSA2 [11] and ACO on known benchmark problems; second, we wish to determine good values for the parameters α and ω in Eq. (3.68). These values will then be used in our comparative experiments in Section 5; VNCSA will either be selected or rejected for ANCFIS using these parameter settings. All of the problems selected are minimization problems; wherever necessary, the original maximization problems can be converted into minimizing problems by negation. Six test problems are presented here.

3.4.2.1 Unconstrained optimization

The first three problems are test functions for unconstrained global optimization: the Goldstein-Price (GP) Function, Hartman's function 1 (HN1), and Hartman's function 2

Table 3. 1 Parameters of VNCSA and CSA2 used in test functions

f	T_{max}	T_{min}	α (CSA2)	β	L_{max}	α (VNCSA)	ω (VNCSA)	M
GP	10	0.01	1	0.94	2	0.99	0.95	400
HN1	10	0.01	1	0.88	2	0.99	0.95	400
HN2	10	0.01	1	0.95	2	0.99	0.95	400

Table 3. 2 Parameters of ACO used in test functions

f	N_outer	N_inner	N_search	M	α	β	$P_{transtion}$	γ	ρ
GP	10	10	20	10	1	1	0.1	Exp(-1.005)	0.7
HN1	5	5	10	5	1	1	0.1	Exp(-1.005)	0.7
HN2	10	5	10	10	1	1	0.1	Exp(-1.005)	0.7

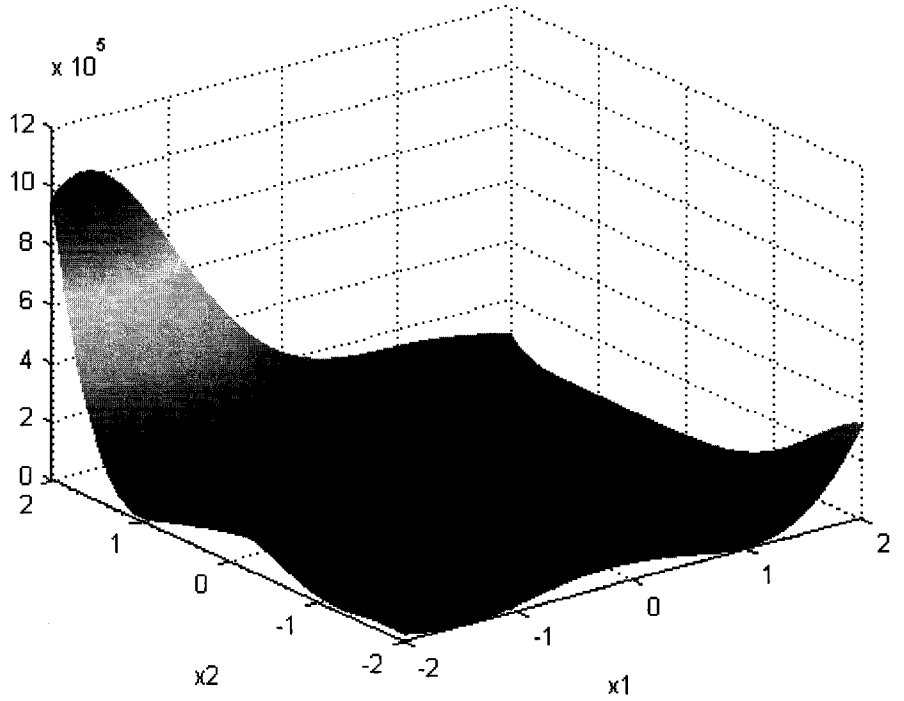


Fig. 3. 8 The graph of Goldstein-Price function

(HN2) [11]. To ensure a fair comparison, the VNCSA algorithm uses the same parameter values as CSA2; these are listed in Table 3.1, along with the values of α and ω in VNCSA. The parameters used in ACO are listed in Table 3.2.

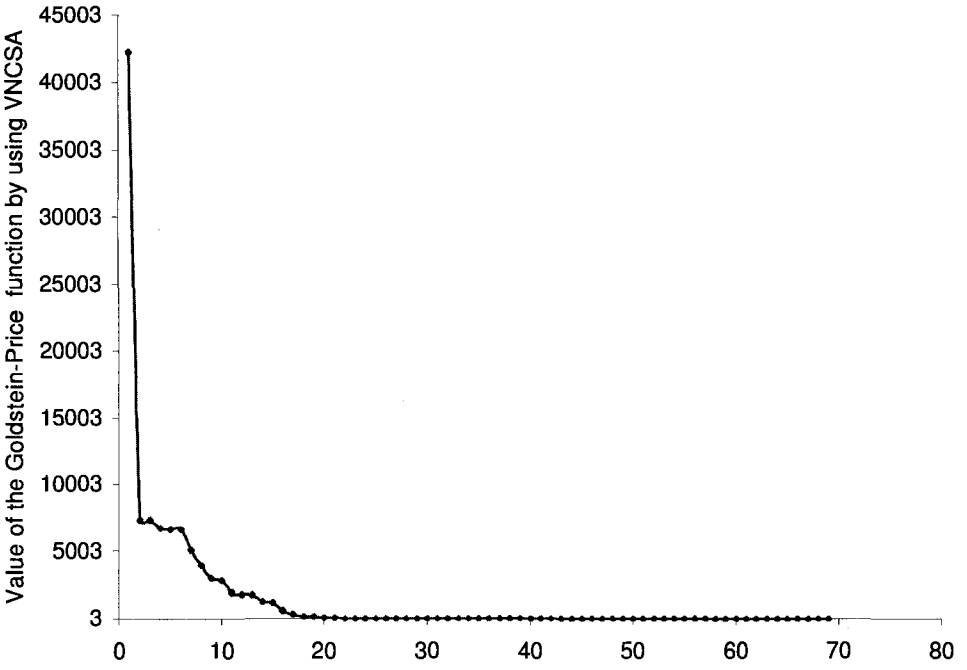


Fig. 3. 9 The iteration process using VNCSA algorithm

The first is the Goldstein-Price Function with several local minima,

$$f_{18}(X)=\left[1+(x_1+x_2+1)^2\left(19-14x_1+3x_1^{\,2}-14x_2+6x_1x_2+3x_2^{\,2}\right)\right]$$

$$\times\left[30+(2x_1-3x_2)^2\left(18-32x_1+12x_1^{\,2}+48x_2-36x_1x_2+27x_2^{\,2}\right)\right],\ |x_i|\le 2,\quad (3.70)$$

whose global minimum is $\min\left(f_{18}(X^*)\right)=f_{18}(0,-1)=3$. The graph of the Goldstein function is shown in Fig. 3.8 and its iteration process is shown in Fig. 3.9 using VNCSA algorithm.

The second is the Hartman's function 1 with 4 local minima,

$$f_{19}(X)=-\sum_{i=1}^{4}c_i\exp\left[-\sum_{j=1}^{3}a_{ij}\left(x_j-p_{ij}\right)^2\right],\ 0\le x_j\le 1. \qquad (3.71)$$

where $(a_{ij}) = \begin{bmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 0.1 & 10 & 35 \end{bmatrix}, (c_i) = (1 \quad 1.2 \quad 3 \quad 3.2), (p_{ij}) = \begin{bmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{bmatrix}$.

The global minimum is $\min(f_{19}(X^*)) = f_{19}(0.114, 0.556, 0.852) = -3.86$. Fig. 3.10 shows the iteration process of Hartmann's function 1 using VNCSA algorithm.

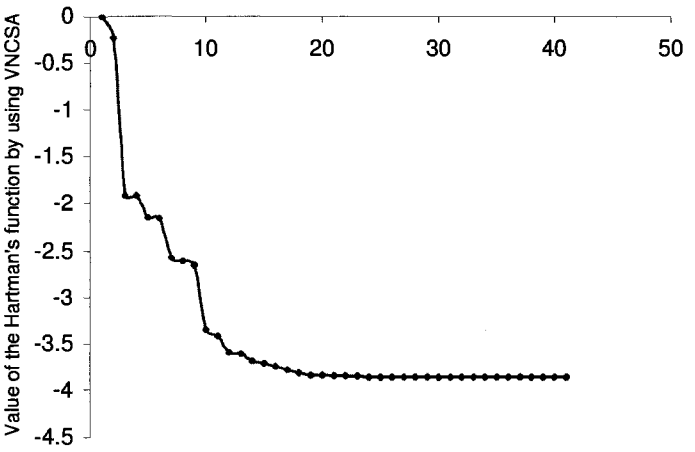


Fig. 3. 10 The iteration process for Hartman's function 1 using VNCSA algorithm

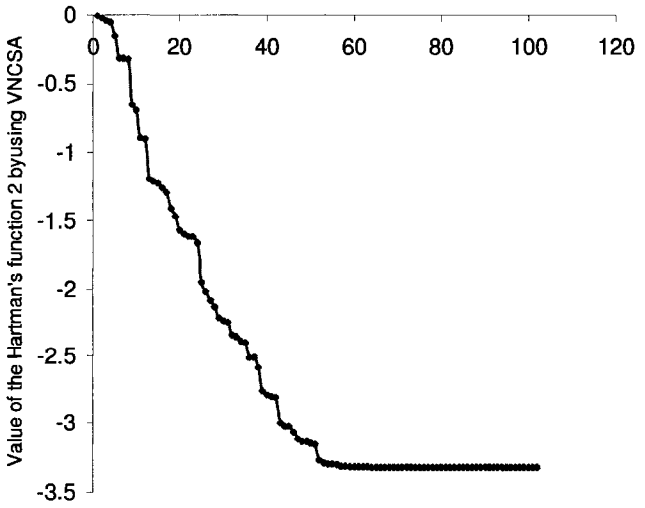


Fig. 3. 11 The iteration process for Hartman's function 2 using VNCSA algorithm

The third is the Hartman's function 2 with 6 local minima,

$$f_{20}(X) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij}\left(x_j - p_{ij}\right)^2\right],\ 0 \le x_j \le 1 \tag{3.72}$$

where $(a_{ij}) = \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}, (c_i) = (1 \quad 1.2 \quad 3 \quad 3.2),$

$$(p_{ij}) = \begin{bmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1415 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{bmatrix},$$

The global minimum is $\min(f_{20}(X^*)) = f_{20}(0.201, 0.15, 0.477, 0.275, 0.311, 0.657) = -3.32$.

Fig. 3.11 shows the iteration process of Hartmann's function 2 using VNCSA algorithm.

Table 3. 3 Testing results on unconstrained functions

f	VNCSA	CSA2	ACO
GP	$X^* = (0,-1)$ $f_{18}(X^*) = 3$ number of f evaluated: 89602	$X^* = (0,-1)$ $f_{18}(X^*) = 3$ number of f evaluated: 88682	$X^* = (0,-1)$ $f_{18}(X^*) = 3$ number of f evaluated: 2393127
HN1	$X^* = (0.114851 \quad 0.555653 \quad 0.852542)$ $f_{19}(X^*) = -3.862782$ number of f evaluated: 43623	$X^* = (0.120624 \quad 0.555664 \quad 0.852517)$ $f_{19}(X^*) = -3.862760$ number of f evaluated: 42220	$X^* = (0.114622 \quad 0.555644 \quad 0.852544)$ $f_{19}(X^*) = -3.862782$ number of f evaluated: 2225474
HN2	$X^* = (0.201708 \quad 0.146781 \quad 0.476745$ $0.275342 \quad 0.311652 \quad 0.657276)$ $f_{20}(X^*) = -3.321995$ number of f evaluated: 107604	$X^* = (0.201635 \quad 0.147548 \quad 0.476804$ $0.275567 \quad 0.311628 \quad 0.657635)$ $f_{20}(X^*) = -3.321984$ number of f evaluated: 105840	$X^* = (0.201728 \quad 0.146850 \quad 0.476699$ $0.275335 \quad 0.311669 \quad 0.657286)$ $f_{20}(X^*) = -3.321995$ number of f evaluated: 2215046

Table 3. 4 Parameters of VNCSA and CSA2 used in testing constraint functions

f	T_{max}	T_{min}	α (CSA2)	β	L_{max}	α (VNCSA)	ω (VNCSA)	M
G4	1.01	0.01	1	0.98	12	0.99	0.95	4100
G9	1.01	0.01	1	0.98	12	0.99	0.95	4100
G10	1.01	0.01	1	0.98	12	0.99	0.95	4100

Table 3. 5 Parameters of ACO used in test constraint functions

f	N_outer	N_inner	N_search	M	α	β	$P_{transtion}$	γ	ρ
G4	20	50	50	20	1	1	0.1	0.5	0.7
G9	20	50	50	20	1	1	0.1	0.5	0.7
G10	20	50	50	20	1	1	0.1	0.5	0.7

The results of our experiments using VNCSA, and CSA2 and ACO are shown in Table 3.3. For each test case, we list the type of function f, the best solution reached X^*, and the minimum of function $f(X^*)$. Form the Table 3.3, we can see that the VNCSA algorithm for unconstrained functions can find the more exact optimal solutions than the CSA2 algorithm under the same parameters. In addition, the ACO algorithm is much less efficient than the VNCSA algorithm since more feasible solutions are evaluated, which results in very expensive computation.

3.4.2.2 Constrained optimization Problems

We also employed three difficult benchmark constrained functions to test the performance of the VNCSA algorithm, denoted G_4, G_9, G_{10}, respectively [77]. As mentioned before, to ensure a fair comparison, the parameters used in VNCSA algorithm are set to the same parameter values as CSA2, which are listed in Table 3.4, as well as the

values of α and ω in VNCSA. The parameters used in ACO are listed in Table 3.5. Again we only report the best parameter values after parameter exploration.

The first constrained problem can be defined by the function constraints

$$0 \le 85.334407 + 0.0056858x_2x_5 + 0.00026x_1x_4 - 0.0022053x_3x_5 \le 92,$$

$$90 \le 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 - 0.0021813x_3^2 \le 110,$$

$$20 \le 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 - 0.0019085x_3x_4 \le 25,$$

and the added variable constraints $78 \le x_1 \le 102$, $33 \le x_2 \le 45$, $27 \le x_i \le 45$, $i = 3,4,5$ with an objective function to be minimized :

$$G_4(X) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141, \qquad (3.73)$$

The best solution obtained in 10 runs of a genetic algorithm [78] was

$$G_4(80.49,35.07,32.05,40.33,33.34) = -30005.7,$$

Whereas a better solution [79] is $G_4(78.0,33.0,29.995,45.0,36.776) = -30665.7$.

The iteration process for G4 function is shown in Fig. 3.12 using VNCSA algorithm.

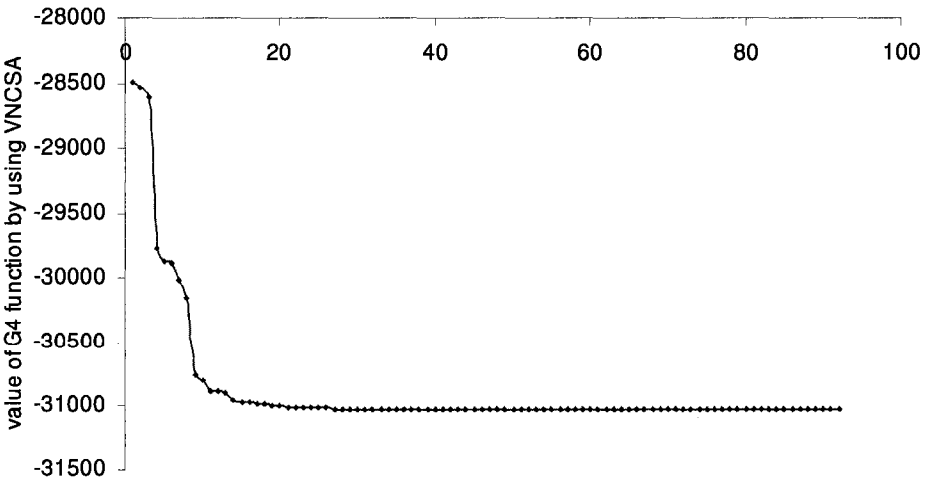


Fig. 3. 12 The iteration process for G4 using VNCSA algorithm

The second constrained function is the function with four nonlinear constraints examined by Michalewicz, represented by

$$G_9(X)=(x_1-10)^2+5(x_2-12)^2+x_3^{\,4}+3(x_4-11)^2$$
$$+10x_5^{\,6}+7x_6^{\,2}+x_7^{\,4}-4x_6x_7-10x_6-8x_7, \qquad (3.74)$$

subject to thefollowing constraints:

$$127-2x_1^{\,2}-4x_2^{\,4}-x_3-4x_4^{\,2}-5x_5\geq 0, \quad 282-7x_1-3x_2-10x_3^{\,2}-x_4+x_5\geq 0,$$

$$196-23x_1-x_2^{\,2}-6x_6^{\,2}+8x_7\geq 0, \qquad -4x_1^{\,2}-x_2^{\,2}-3x_1x_2-2x_3^{\,2}-5x_6+11x_7\geq 0,$$

and bounds: $-10.0\leq x_i\leq 10,\ i=1,2,\ldots 7.$

The global minimum is

$$\min G_9(X^*)=G_9(2.330499,1.951372,-0.4775414,4.365726,-0.6244870,1.038131,1.594227)$$
$$=680.6300573.$$

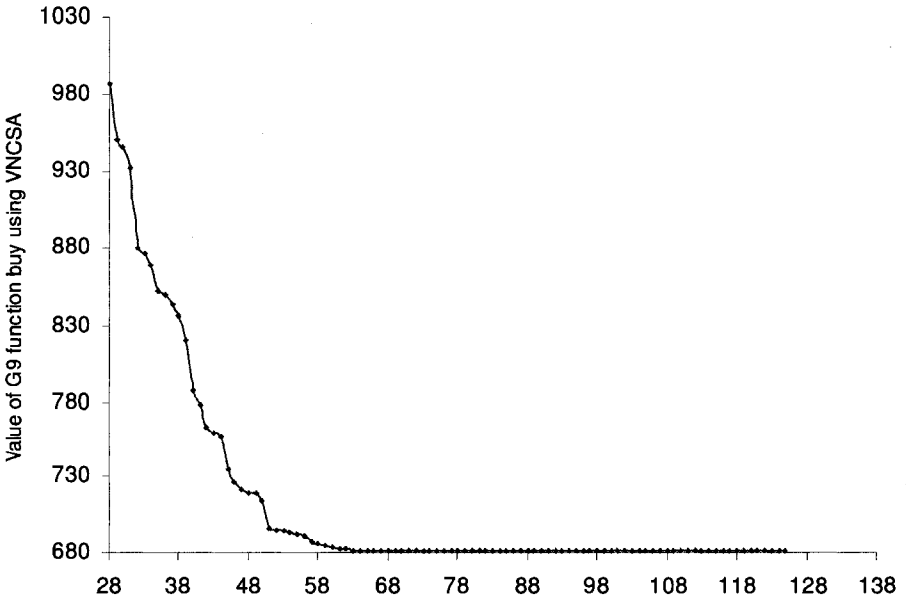


Fig. 3. 13 The iteration process for G9 by using VNCSA algorithm

The iteration process for G9 function is shown in Fig. 3.13 shows using VNCSA algorithm.

The third is the function with 3 linear and nonlinear constraints, whose global optimum is very hard to be found, since all six constraints are active at the global optimum.

$$G_{10}(X) = x_1 + x_2 + x_3, \tag{3.75}$$

subject to

$$1 - 0.0025(x_4 + x_6) \geq 0, \qquad 1 - 0.0025(x_5 + x_7 - x_4) \geq 0,$$

$$1 - 0.01(x_8 + x_5) \geq 0, \qquad x_1 x_6 - 833.33252 x_4 - 100 x_1 + 83333.333 \geq 0,$$

$$x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4 \geq 0, \quad x_3 x_8 - 1250000 - x_3 x_5 + 2500 x_5 \geq 0,$$

and bounds

$100 \leq x_1 \leq 10000$, $1000 \leq x_i \leq 10000$, $i = 2,3$, $10 \leq x_i \leq 1000$, $i = 4,5,\ldots,8$.

up to now, the best global solution is

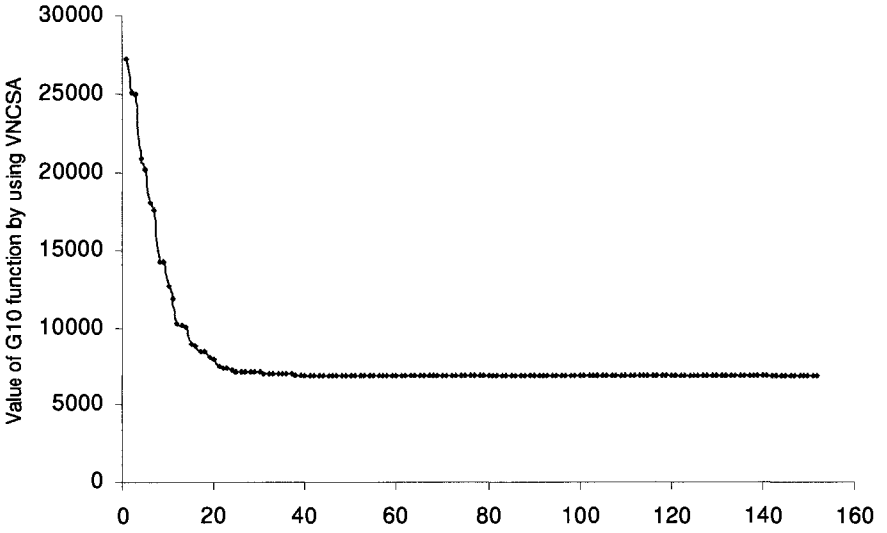


Fig. 3. 14 The iteration process for G10 by using VNCSA algorithm

Table 3. 6 Testing results on constrained functions

f	VNCSA	CSA2	ACO
G_4	X^*=(78, 33, 27.070997, 45, 44.969242) $G_4(X^*)$=-31025.560237	X^*=(78, 33, 27.071001, 45, 44.969231) $G_4(X^*)$=-31025.559909	X^*=(78, 33.014567, 27.070785, 44.999999, 44.945957) $G_4(X^*)$=-31024.901270
G_9	X^*=(2.330779, 1.951318, -0.476265, 4.365744, -0.624418, 1.037897, 1.594240) $G_9(X^*)$=680.630062	X^*=(2.327393, 1.953413, -0.480587, 4.361062, -0.621978, 1.044346, 1.593403) $G_9(X^*)$=680.631086	X^*=(2.430184, 1.901512, 0.602105, 4.413564, -0.510547, 1.512274, 1.969299) $G_9(X^*)$=685.910576
G_{10}	X^*=(569.083822, 1359.365096, 5120.876153, 181.157672, 295.164954, 218.842328, 285.992718, 395.164954) $G_{10}(X^*)$=7049.325070	X^*=(588.353199, 1317.481539, 5144.026145, 182.768374, 294.238954, 217.231446, 288.529420, 394.238954) $G_{10}(X^*)$=7049.860884	N/A

$X^* = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)$

where $\min G_{10}(X^*) = 7049.330923$. Fig. 3.14 shows the iteration process of G10 using VNCSA algorithm.

Our results for these three functions are summarized in Table 3.6, which presents the same information as Table 3.3. From Table 3.6, the minimum $G(X^*)$obtained by VNCSA is superior to the other two methods for all three functions. For function G_4, all three algorithms found better solutions than the present published solution. For function G_9, the minimum found by using VNCSA is almost the same as the present global minimum published. For function G_{10}, the ACO algorithm could not find a feasible solution. VNCSA, however, found a solution that is slightly better than the

current published solution. Based on these experimental results, we have selected the values α=0.99 and ω=0.95 for the experiments in Chapter 4.

4 Experimental Comparison of the Three Alternatives in ANCFIS

We have compared the three alternatives for the layer-1 optimization algorithm (VNCSA, CSA2, and ACO) under the ANCFIS architecture on six time-series forecasting problems. One is a synthetic dataset generated using two sinusoid functions. The five others are the well-known Mackey-Glass time series [5][6], Santa Fe dataset A (laser)[80][81], the commonly used annual Wölfer sunspot numbers[82]-[86], a wave-heights prediction problem [82][87], and predicting the brightness of a variable star [82]. The six datasets exhibit what we term "approximately periodic" behavior: the measurement of interest has a recurring pattern, but never actually repeats itself. The Mackey-Glass and Santa Fe A time series, in particular, are known to be chaotic in nature. The goal of these experiments is to determine which optimization technique will be employed in the ANCFIS architecture. Our experimental design for each dataset follows the usual forecasting methodology: a single-split, one-step ahead prediction design where the chronological ordering of the data points is preserved, and the training data are chronologically earlier than the test data. In all of the experiments, the root mean squared error (RMSE) (a commonly used error measure in forecasting) is reported [88]

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(desired\ output_j - predicted\ output_j\right)^2} \quad . \tag{4.1}$$

Since we apply the same mathematical optimization model (seen in Eq. (3.58)) with 4 variables , a,b,c,d , and the same constraints in ANCFIS for predicting all six different time series datasets, we use the same values of parameters used in VNCSA,

Table 4. 1 Parameters of VNCSA and CSA2 used in ANCFIS for time series data prediction

Algorithm	T_{max}	T_{min}	β	L_{max}	α	ω	M
VNCSA	100	0.01	0.98	2	0.99	0.95	400
CSA2	100	0.01	0.98	2	1		400

Table 4. 2 Parameters of ACO used in ANCFIS for time series data prediction

Algorithm	N_outer	N_inner	N_search	M	α	β	$P_{transtion}$	γ	ρ
ACO	10	10	50	10	1	1	0.1	0.95	0.6

CSA2, and ACO in all six time series forecasting experiments; these are shown in Table 4.1, and Table 4.2, separately; these are the best parameters found after exploration.

In ANCFIS, we use a heuristically select the input vector length by choosing the number of points that covers one "period" of the function. In addition, the format of input-output pairs should be like:

$$[x(t), x(t+1), \ldots, x(t+n-1); x(t+n)], \tag{4.2}$$

where n is the length of input vector presented to the network, and $x(t+n)$ is the value to predict.

For the comparison with the VNCSA-ANCFIS, we create a mapping for ANFIS from 4 points of the time series spaced 6 apart — that is, $[x(t), x(t+6), x(t+12), x(t+18)]$ to a predicted future value $x(t+19)$.

4.1 Synthetic dataset

The synthetic dataset is generated by the formula:

$$0.5\sin(0.2t+0.67)+0.3+0.25\sin(0.58t+0.1)+0.15 \tag{4.3}$$

sampled in intervals of 0.5 from [0, 750]. This gives a total of 1500 points. The normalized data are shown in Fig. 4.1.The dataset-specific parameters used for this

Table 4. 3 Synthetic dataset parameter exploration

Parameter	Value(s)
Input vector length	65
Step size	0.01
Step size decrease rate	0.8
Step size increase rate	1.1
Epochs	100
Number of membership functions	2

experiment are listed in Table 4.3. Note that the number of membership functions is fixed at 2 (in order to see if the two frequencies can be captured), and the length of the input vector is set to 65. The first 1000 points are taken as training data and the final 500 are used as testing data. This gives us 935 input-output data pairs as the training data set for ANCFIS while the remaining 500 pairs are the test data set.

We compare the results for VNCSA, CSA2, ACO, and ANFIS in Table 4.4. Plainly, the RMSEs of both VNCSA and CSA2 decreased to zero, which were far less than those of ACO and ANFIS. We provide further analysis of the VNCSA alternative in Figs. 4.2 & 4.3; in Fig. 4.2 , we plot the predicted and actual outputs in part (a), and the prediction errors in part (b). We plot the training RMSE for each epoch in Fig. 4.3. Plainly, the ANCFIS and CSA2 exactly track this synthetic time series.

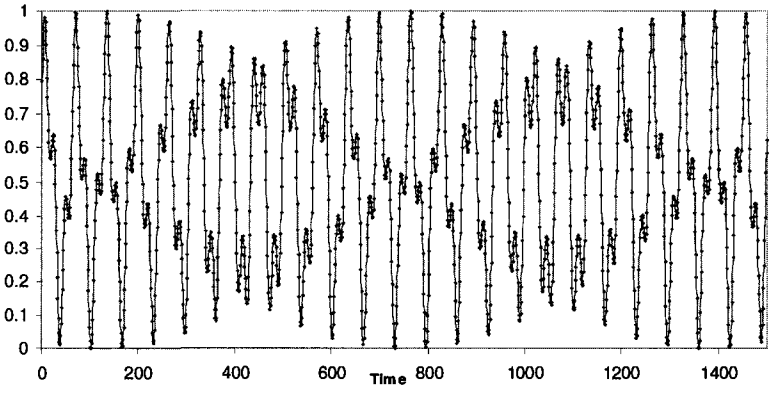


Fig. 4. 1 Synthetic dataset after normalization

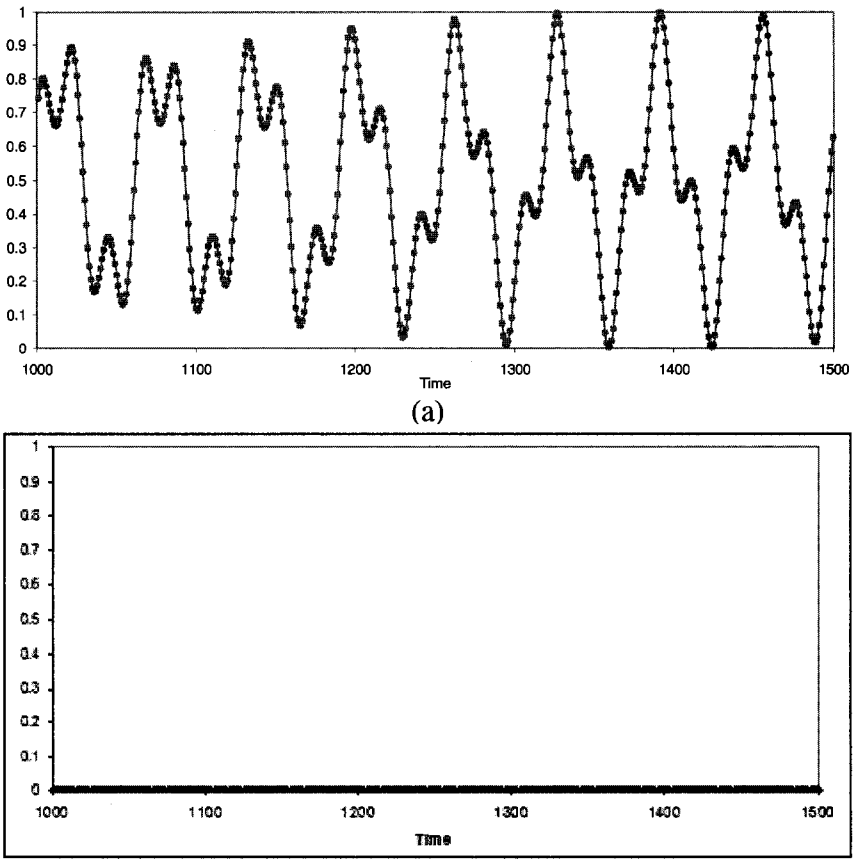


Fig. 4. 2 (a) synthetic time series test results and one-step-ahead prediction(desired values as dashed line, and predicted values as solid line); (b) prediction error.

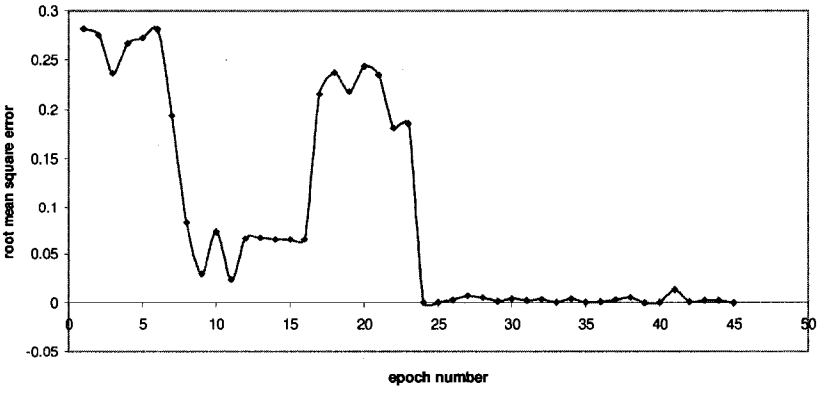


Fig. 4. 3 Training RMSE curves for synthetic time series

Table 4. 4 Comparison testing RMSE for synthetic time series

Method	Test data size	RMSE	Number of rules
ANCFIS using VNCSA	500	0	2
ANCFIS using CSA2	500	0	2
ANCFIS using ACO	500	0.000012	2
ANFIS	500	1.69355e-005	16

4.2 Mackey-Glass Time Series

This time series is produced by the Mackey-Glass differential delay equation [6] represented as

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t), \tag{4.4}$$

We use the same time series as Jang [5]: time step 0.1, initial condition $x(0)=1.2$ and $\tau=17$, with 1000 points; these run from t=124 to t=1123, to avoid initialization transients. We use a window of 44 data points as our input vector (approximately one period, see Fig. 4.4). This gives us 956 input-output data pairs. The first 456 pairs are used as the training data set for ANCFIS while the remaining 500 pairs are the test data set (in common with Jang's experiment [5][6]). The number of CMFs in Layer 1 was set to three, and the step size, the step size increase rate and step decrease rate are set to 0.001, 1.1 , 0.8, respectively. We compare the results for VNCSA, CSA2, ACO, and ANFIS in Table 4.5. Plainly, compared with CSA2 and ACO, VNCSA was superior, with an RMSE that was roughly half that of the CSA2 algorithm, and far less than ACO. Although ANFIS provided the best RMSE, its $RMSE_{trn}$ is larger than its $RMSE_{chk}$, and this is unusual. As Jang analyzed in [5][6], this special case is caused by two reasons: (1) the ANFIS has captured the essential components of the underlying dynamica; and (2)

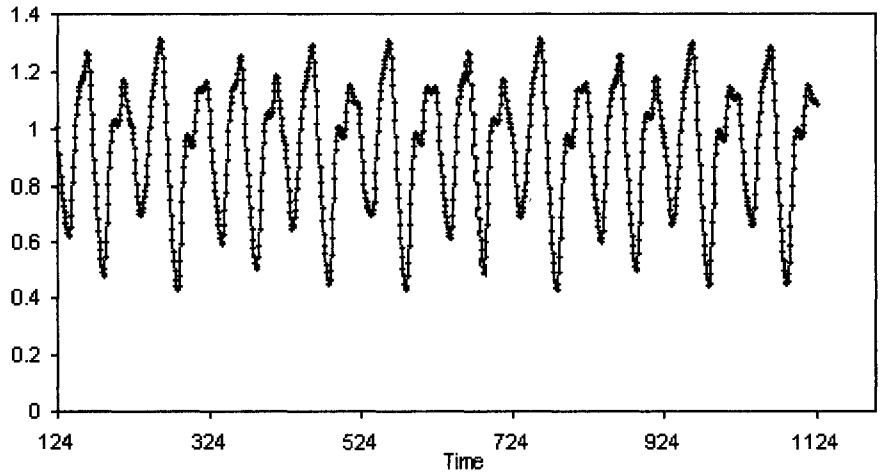
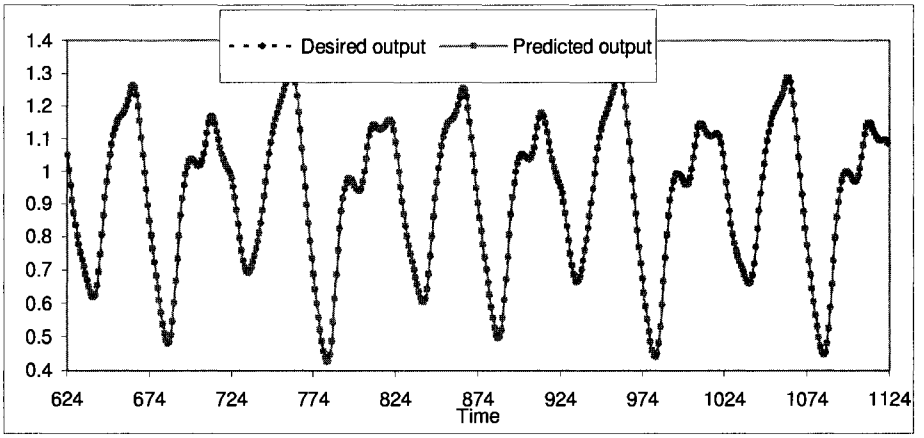
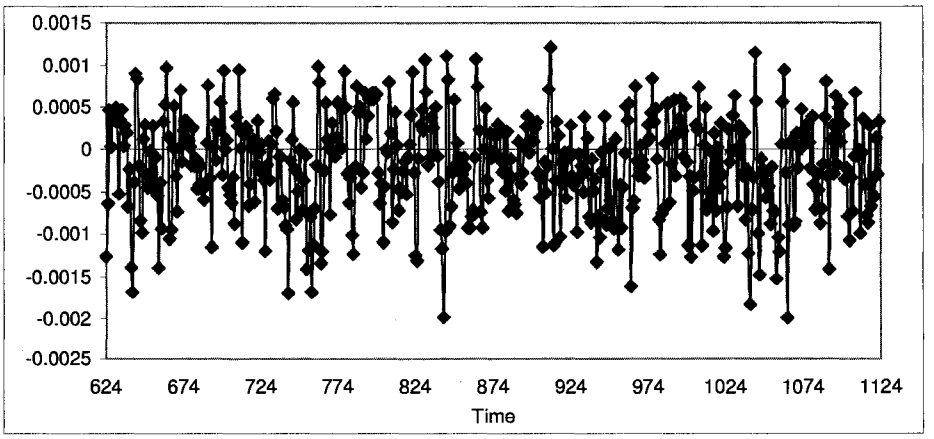


Fig. 4. 4 Mackey-Glass time series from $t = 124$ to 1123



(a)



(b)

Fig. 4. 5 (a) Mackey –Glass time series test results from t=624 to 1123 and one-step-ahead prediction(desired values as dashed line, and predicted values as solid line); (b) prediction error.

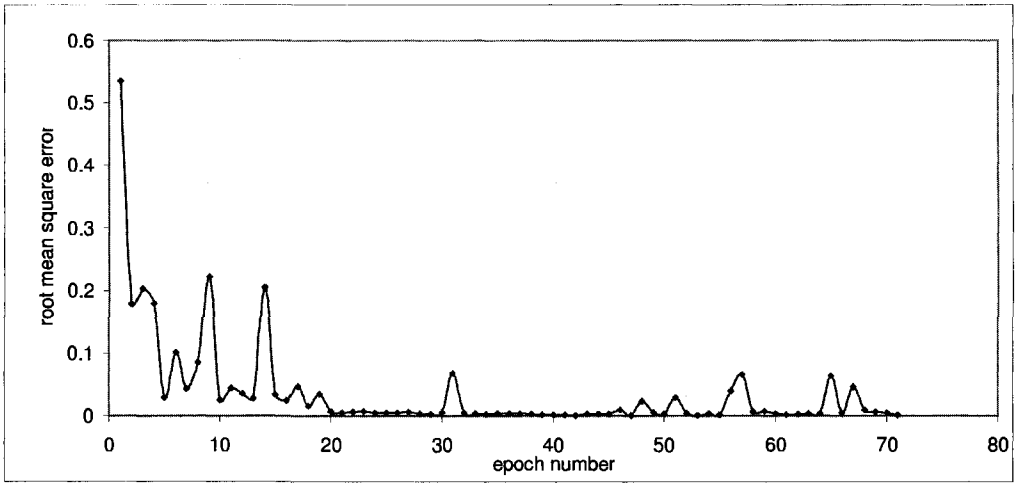


Fig. 4. 6 Training RMSE curves for Mackey-Glass series

Table 4. 5 Comparison of testing RMSE for Mackey-Glass

Method	Test data size	RMSE	Number of rules
ANCFIS using VNCSA	500	0.000688	3
ANCFIS using CSA2	500	0.001237	3
ANCFIS using ACO	500	0.009546	3
ANFIS	500	0.000545665	16

the training data contain the effects of the initial conditions ($x(t)$ is set to 0 for $t \leq 0$ in the integration), which might not be easily accounted for in the essential components identified by ANFIS. We provide further analysis of the VNCSA alternative in Figs. 4.5 & 4.6; in Fig. 4.5, we plot the predicted and actual outputs in part (a), and the prediction errors in part (b). We plot the training RMSE for each epoch in Fig 4.6. Plainly, the ANCFIS forecasts track this chaotic time series very closely.

We provide further analysis of the learning that takes place in Layer 1 in Table 4.6, where we present the three complex membership functions before and after the training of the network. As per usual practice, these parameters are initialized to small

Table 4. 6 Membership functions

	INITIAL CMFS	CMFS AFTER LEARNING
1	0.4081861768*sin(0.1628570557*t +0.5568542480)+0.5376982526;	0.087973*sin(0.998037 *t + 79.364839)+0.861512;
2	0.0952898986*sin(0.0535736084 *t +0.1979370117)+0.1264701374;	0.022990*sin(0.000226*t +49.514667)+0.173827;
3	0.0042810123*sin(0.2807769775*t +0.6464538574)+0.4306143397;	0.064255*sin(0.000002*t +85.287607)+0.463169;

random values prior to the start of the training process. The reader will note that extensive changes take place; in particular, there are major changes in the frequency and phase of the sine waves after training.

In order to demonstrate the parsimony of the VNCSA-ANCFIS network structure parsimony, we illustrate ANFIS architecture used in [5] and VNCSA-ANCFIS network structure, respecively. Fig. 4.7 is a VNCSA-ANCFIS architecture. Fig.4.8 illustrates an ANFIS architecture that is equivalent to a four-input first order Sugeno fuzzy model with sixteen rules. Assume that the length of input vector is also set to 44, the number of rules will reach to 2^{44}, while the number of rules in VNCSA-ANCFIS is only three. The VNCSA-ANCFIS network architecture is much more parsimonious, thanks to the windowed input format.

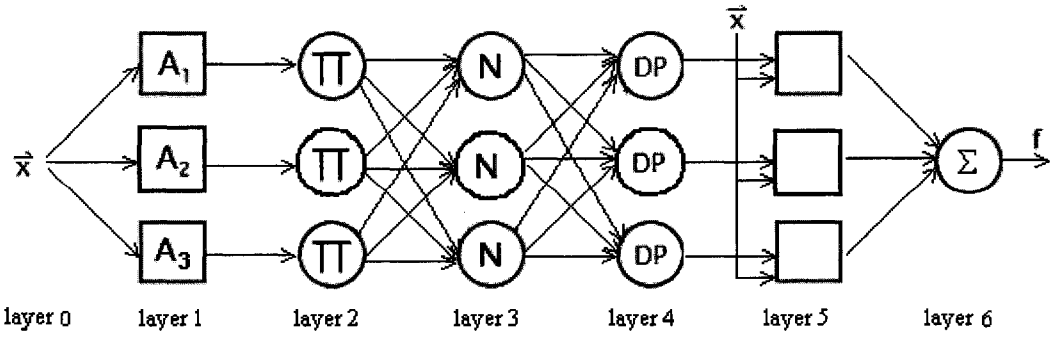


Fig. 4. 7 A VNCSA-ANCFIS architecture

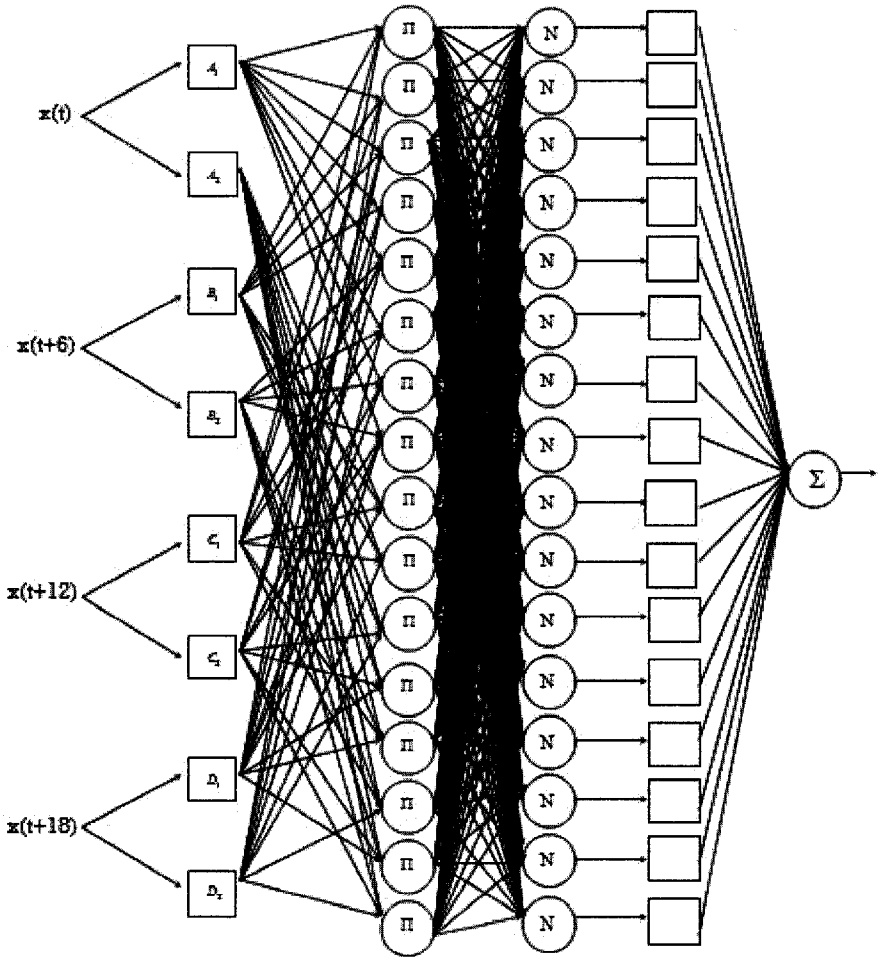


Fig. 4. 8 An ANFIS architecture with 4-input Sugeno fuzzy model with sixteen rules (The connections from inputs to layer 4 are not shown)

4.3 Santa Fe dataset A

This was the first in a series of six datasets chosen for the Santa Fe time series competition [89] in 1991. The dataset is composed of 1960 observations made on an 81.5-micron 14NH3 cw (FIR) laser. This dataset has been analyzed in [90]-[92].

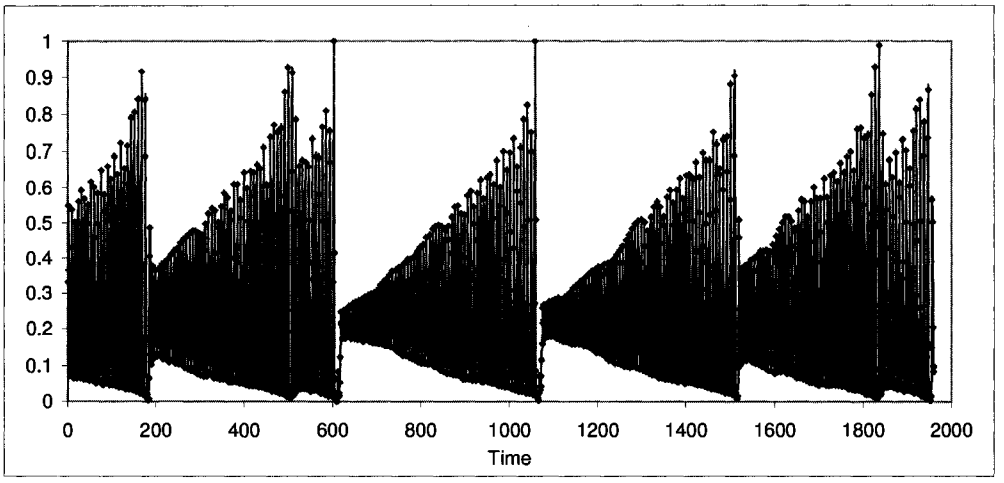
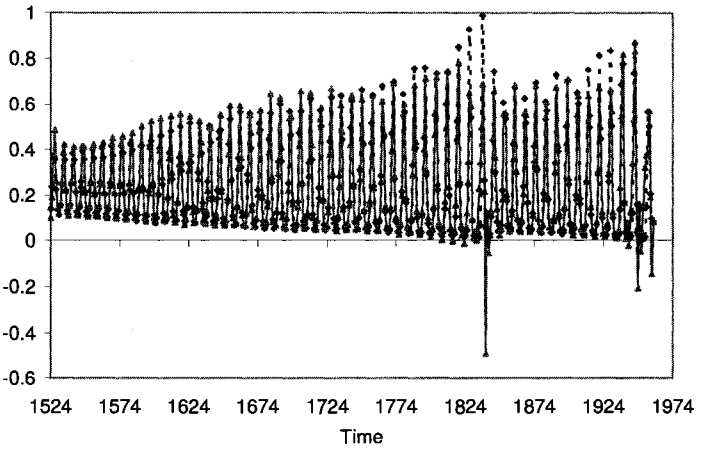
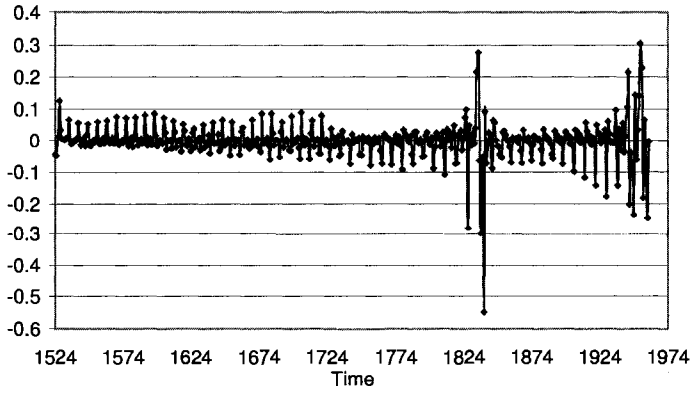


Fig. 4. 9 Santa Fe dataset A (laser) after normalization



(a)



(b)

Fig. 4. 10 (a) Santa Fe dataset A (laser)time series test results and one-step-ahead prediction(desired values as dashed line, and predicted values as solid line); (b) prediction error.

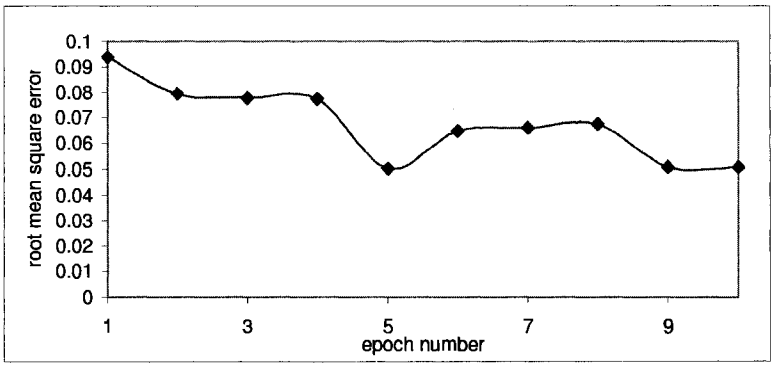


Fig. 4. 11 Training RMSE curves for Santa Fe dataset A (laser) series

Table 4. 7 Comparison of testing RMSE for Santa Fe dataset A (laser) series

Method	Test data size	RMSE	Number of rules
ANCFIS using VNCSA	437	0.060118	3
ANCFIS using CSA2	437	0.089928	3
ANCFIS using ACO	437	0.091036	3
ANFIS	437	0.0888076	16

The normalized laser data are shown in Fig. 4.9. The number of membership functions is fixed at 3, and the length of the input vector is set to 11, the step size, the step size increase rate and step decrease rate are set to 0.001, 1.1, 0.8, respectively. The first 1523 points are taken as training data and the final 437 are used as testing data. This gives us 1512 input-output data pairs as the training data set for ANCFIS while the remaining 437 pairs are the test data set. We compare the results for VNCSA, CSA2, ACO, and ANFIS in Table 4.7. Among these alternatives, VNCSA was superior, with an RMSE that was roughly two-thirds those of CSA2, ACO, and ANFIS.

We again provide further analysis of the VNCSA results in Figs. 4.10 and 4.11; we plot actual versus predicted outputs in Fig. 4.10(a), we also plot prediction errors in

Fig. 4.10(b). Fig. 4.11 shows training RMSE per epoch. Again, we see that ANCFIS closely tracks this dataset.

4.4 Sunspots

The annual sunspot data time series was chosen because it is a commonly cited time series dataset [83]-[86], [93]-[97]. The data contains the average number of sunspots per year as measured from 1700 to 1979 (see Fig. 4.12). In our experiments, the years 1700-1920 are used as training data. The remaining years up to 1979 are used as testing data. This is consistent with the experiments from [86]. Other papers [85][96] have used the first 180 points as training data with the remaining years up until 1979 as testing data.

The length of input vector is set to 12. Thus, the first 209 input-output pairs are used as the training data set for ANCFIS while the remaining 59 input-output pairs are the testing data set. The number of CMFs assigned to the input vector of the ANCFIS was set to three, and the step size, the step size increase rate and step decrease rate are set to 0.001, 1.1 , 0.8, separately.

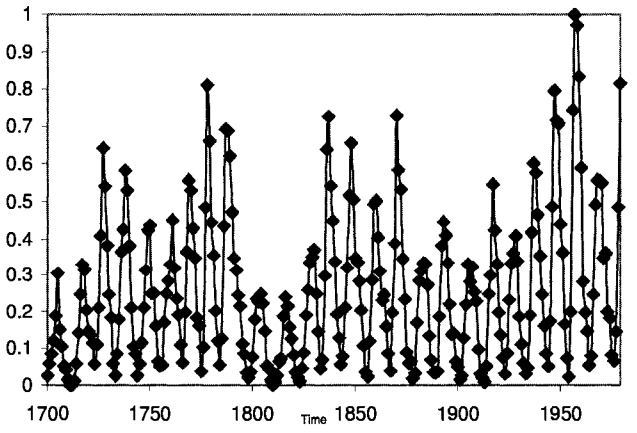
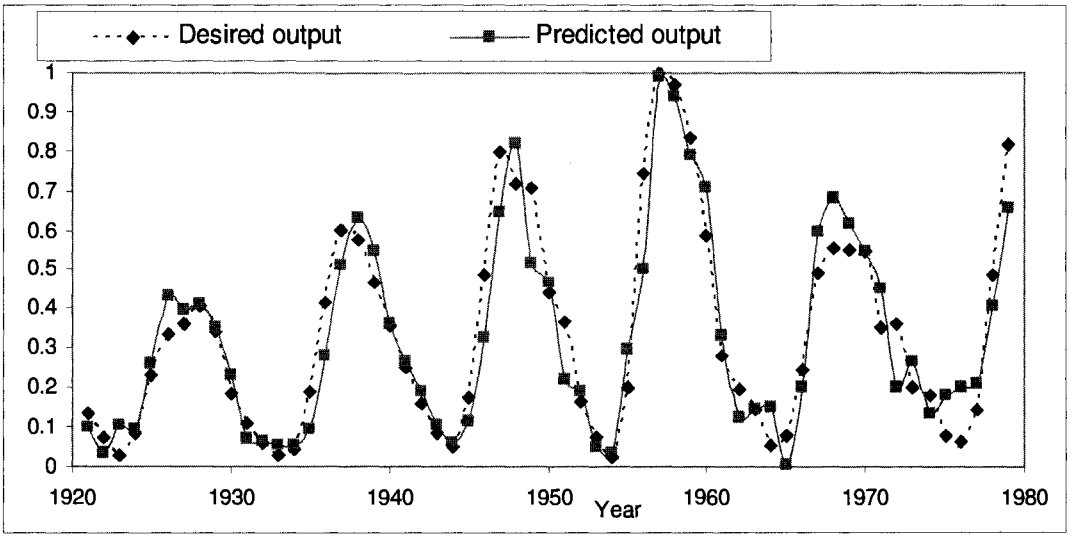
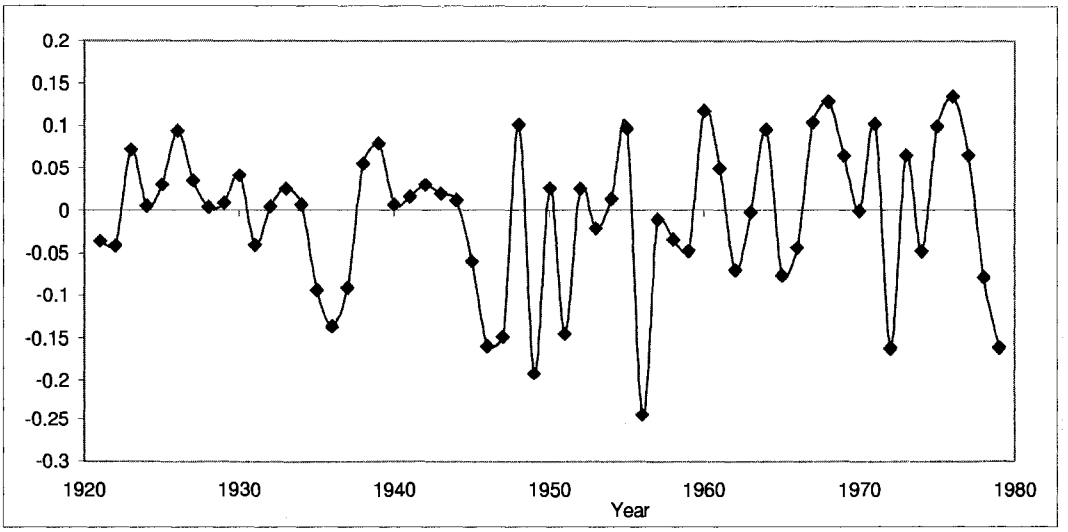


Fig. 4. 12 Sunspot series after normalization

The test RMSE results are listed in Table 4.8. Again, the VNCSA alternative provided a better prediction with lower RMSE. The desired and predicted values for testing data are very close in Fig. 4.13 (a); the prediction errors are plotted in Fig. 4.13(b) depicts the difference between them on a much finer scale. Fig.4.14 shows the training RMSE cureves.



(a)



(b)

Fig. 4. 13 (a) Sunspot time series test results from t=624 to 1123 and one-step-ahead prediction(desired values as dashed line, and predicted values as solid line); (b) prediction error.

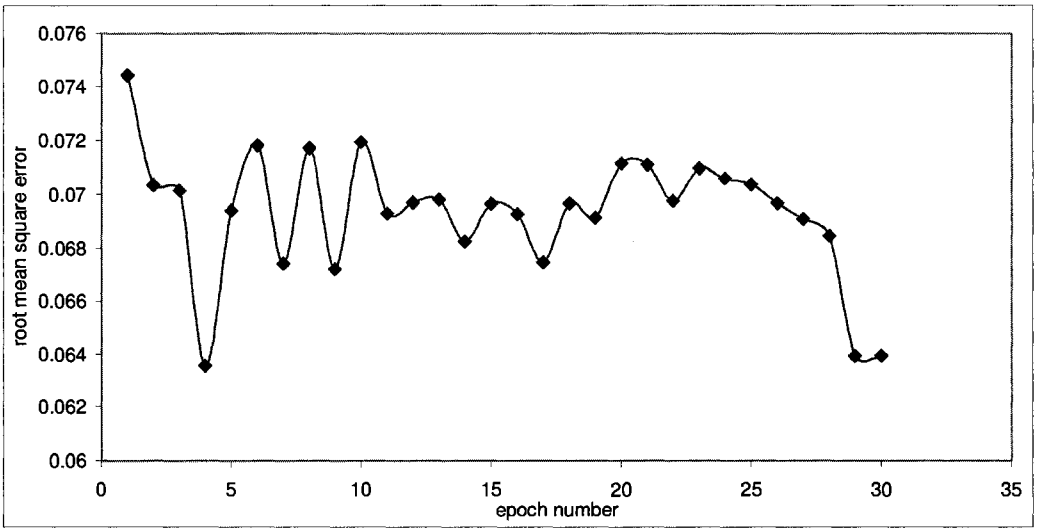


Fig. 4. 14 Training RMSE curves and testing RMSE curves for Sunspot series

Table 4. 8 Comparison of testing RMSE for sunspot series

Method	Test data size	RMSE	Number of rules
ANCFIS using VNCSA	59	0.086488	3
ANCFIS using CSA2	59	0.093512	3
ANCFIS using ACO	59	0.089405	3
ANFIS	59	0.29319	16

4.5 Predicting Stellar Brightness

This is a time series that records the daily brightness of a variable star on 600 successive midnights, see Fig. 4.15. The dataset was normalized so that all measurements fall in [0,1]. The first 480 data points are used as the training data set, while the remaining 120 data points constitute the test data. We use an input length of 27 (again, approximately one period), while the other parameters such as the number of complex membership functions, step size etc. are the same as those for predicting the Mackey-Glass time series. Thus, there are 453 input-output pairs in the training set. The test RMSE results are listed in Table 4.9. Again, the VNCSA alternative provided a lower

RMSE (although this time only by a small amount). We again provide further analysis of the VNCSA results in Figs. 4.16 and 4.17; actual versus predicted outputs are plotted in Fig. 4.16(a), while prediction errors are plotted in Fig. 4.16(b). Training RMSE per epoch is plotted in Fig. 4.17. Again, we see that ANCFIS closely tracks this dataset.

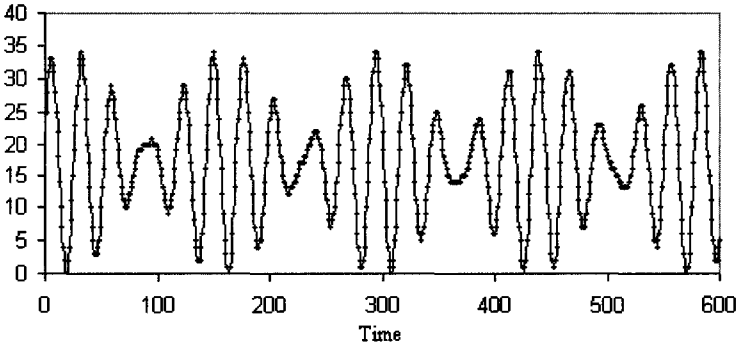


Fig. 4. 15 The Star time series data before normalizing

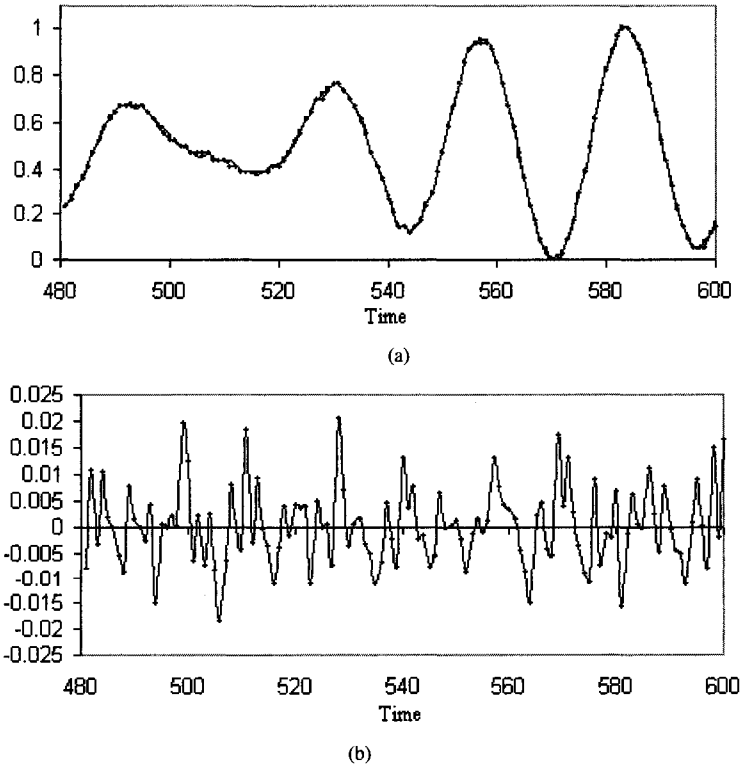


Fig. 4. 16 (a) the Star time series testing data from t=481 to 600(desired values as dashed line, and predicted values as solid line); (b) prediction error

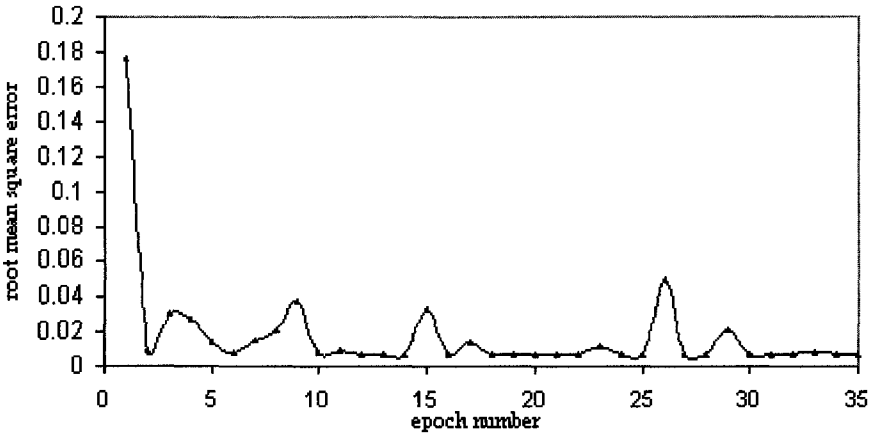


Fig. 4. 17 Training RMSE curves for Star series

Table 4. 9 Comparison of testing RMSE for star brightness

Method	Test data size	RMSE	Number of rules
ANCFIS using VNCSA	480	0.007578	3
ANCFIS using CSA2	480	0.007611	3
ANCFIS using ACO	480	0.007943	3
ANFIS	480	0.0113775	16

4.6 Predicting Wave Height

This time series records forces on a cylinder suspended in a tank of water, sampled every 0.15 seconds, and contains 320 data points as shown in Fig. 4.18. After normalizing, we choose the first 256 points as the training data set, and the remaining 64 points as the testing data. We set the length of input vector to 16, and the other parameters are the same as above, giving us 240 input-output pairs as our training set. The test RMSE results are presented in Table 4.10. Once again, the VNCSA alternative is the best. Further analysis is presented in Figs. 4.19 and 4.20; actual versus predicted outputs are plotted in Fig. 4.19(a), while predictions errors are plotted in Fig. 4.19(b). Training RMSE per epoch is plotted in Fig. 4.20.

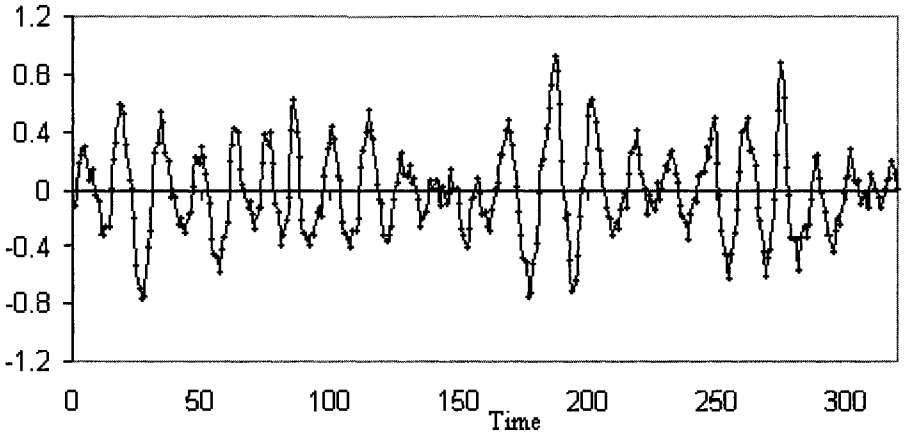


Fig. 4. 18 The waves time series data before normalizing

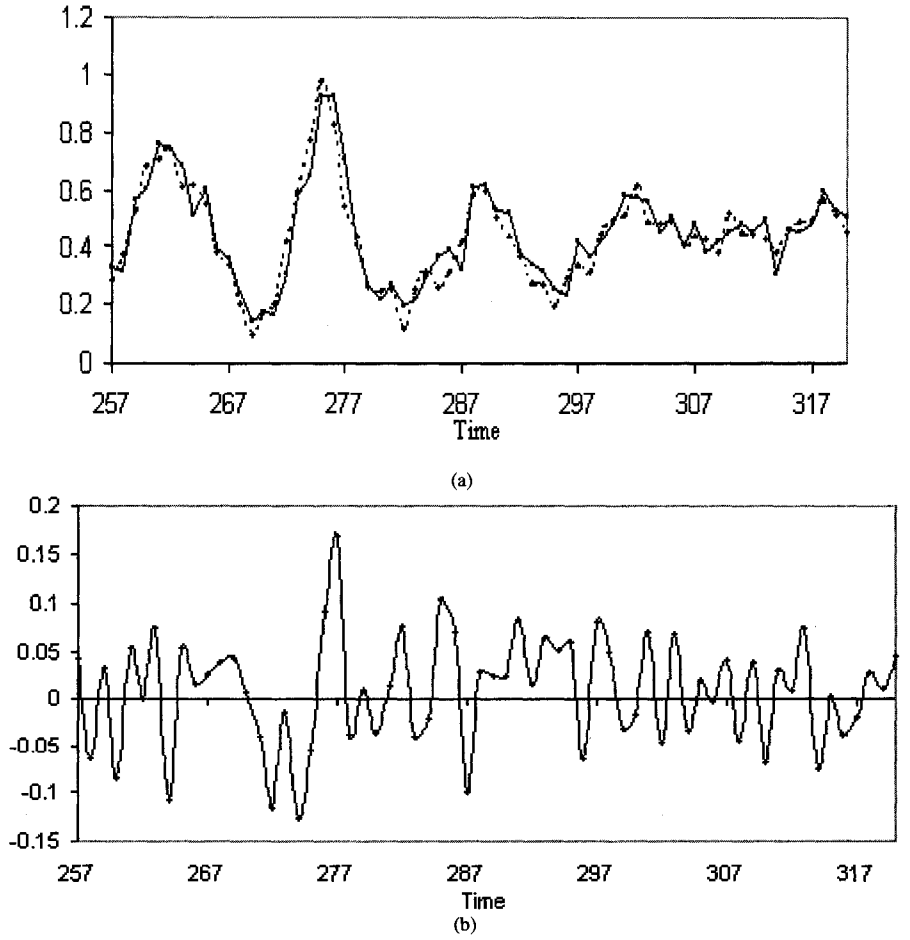


Fig. 4. 19 (a) the waves time series testing data from t=257 to 320 desired values as dashed line, and predicted values as solid line; (b) prediction error

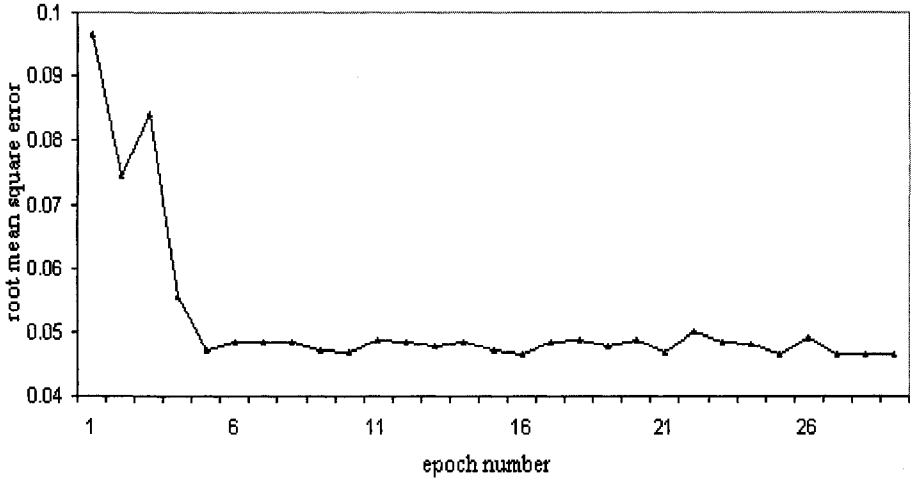


Fig. 4. 20 Training RMSE curves for Waves series

Table 4. 10 Comparison of testing RMSE for waves series

Method	Test data size	RMSE	Number of rules
ANCFIS using VNCSA	256	0.058765	3
ANCFIS using CSA2	256	0.061642	3
ANCFIS using ACO	256	0.062234	3
ANFIS	256	0.113619	16

Overall, our results showed that VNCSA appears to be the best alternative for the layer-1 optimization technique in ANCFIS. VNCSA was consistently superior to CSA2 or ACO, or ANFIS in all six datasets (except the testing RMSE on Mackey-Glass time series generated using ANFIS), and had a strong advantage in the synthetic time series. For the chaotic Mackey-Glass time series, the testing RMSE of ANCFIS using VNCSA is excellent, and very close to that of ANFIS. This also is another time series generated by a deterministic mathematical function. The Wave Height dataset, the Star Brightness dataset, and the sunspot dataset were the observations of physical phenomena, and thus vulnerable to measurement noise; note that we did not perform any noise reduction on

these datasets. The Santa Fe dataset is actually not an approximately periodic dataset, but ANCFIS with VNCSA is able to predict it more accurately than the other alternatives.

5 Summary and Future Work

In this thesis, we have described the development of the new ANCFIS neuro-fuzzy system. Inspired by the Fourier series, which tell us any signal can be approximated by a superposition of sinusoids, sine functions are selected as CMFs [1] to construct a novel architecture, called ANCFIS. This system is the first machine learning architecture to employ the new concept of complex fuzzy logic, as explored by Ramot [3] and Dick [1]. ANCFIS tightly couples phase and magnitude, and implements rule interference. As such, ANCFIS is intended to demonstrate the practical value of complex fuzzy logic, by enabling new solutions to practical machine learning problems. Based on the discussion of the phenomenon of *regularity* in [1], we have selected time series forecasting as the class of problems to solve. We provided an overview of the ANCFIS architecture and learning algorithm. We found that the learning algorithm could not be a pure gradient-descent technique, because there is no closed-form expression for the derivative of the error measure with respect to the complex fuzzy set parameters in ANCFIS. A derivative-free optimization technique is thus needed.

We selected three alternatives for the Layer-1 optimization technique: ant-colony optimization, a chaotic simulated-annealing algorithm from the literature, and a new chaotic simulated-annealing technique developed specifically for ANCFIS. We described this new technique, and performed several benchmarking experiments to validate it, as well as to determine appropriate parameter values for the algorithm. We then compared the performance of the ANCFIS architecture with all three of these alternatives on one

synthetic and five real-world time series datasets. We determined that our new algorithm, VNCSA, was the most effective Layer-1 optimization algorithm.

We also compared our results with those of classic ANFIS architecture in both performance and the number of rules. As the experiments demonstrated, the ANCFIS networks were considerably smaller than an ANFIS network for the same time-series problem. We indeed saw that ANCFIS could predict even a chaotic time series using only 3 rules; an ANFIS network for the same problem is expected to be considerably larger.

In future work, we must first complete a comparative performance analysis of the ANCFIS architecture across numerous time series datasets. We will also compare ANCFIS to different network architectures; radial basis function networks, for example, are a common choice in time series prediction tasks [49]. This work, if successful, will establish complex fuzzy logic as a viable and valuable addition to the field of Computational Intelligence. Additional items of future work include further empirical comparisons of VNCSA against other derivative-free optimization techniques, and the development of alternative machine-learning approaches for complex fuzzy logic (e.g. genetic algorithms).

6 References

[1] S. Dick, "Towards Complex Fuzzy Logic," *IEEE Transactions on Fuzzy Systems*, vol. 13, no. 3, June 2005, pp. 405 – 414.

[2] D. Ramot, R. Milo, M. Friedman, A. Kandel, "Complex Fuzzy Sets," *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 2, April 2002, pp. 171 – 186.

[3] D. Ramot, M. Friedman, G. Langholz, A. Kandel, "Complex Fuzzy Logic," *IEEE Transactions on Fuzzy Systems*, vol. 11, no. 4, Aug. 2003, pp. 450 – 461.

[4] G. J. Klir, B.Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*; Prentice Hall, Englewood Cliffs, NJ, 1995.

[5] R. Jang, C.T. Sun, E. Mizutani, *Neuro-Fuzzy and Soft Computing A Computational Approach to Learning and Machine Intelligence*, Prentice Hall, 1996.

[6] R. Jang, "ANFIS: Adaptive-Network-Based Fuzzy Inference System," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 3, May-June 1993, pp. 665 – 685.

[7] R. Jang, "Neuro-Fuzzy Modeling for Dynamic System Identification," in *Proceedings of the 1996 Asian Soft Computing in Intelligent Systems and Information Processing*, Dec 1996, pp. 320-325.

[8] M. Dorigo, V. Maniezzo and A. Colorni, "Ant System: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man and Cybernetics* - Part B, vol. 26, no.1, pp. 29-41, 1996.

[9] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the travelling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol.1, no.1, pp. 53-66, 1997.

[10] M. Dorigo, G. Di Caro and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol.5, no. 2, pp.137-172, 1999.

[11] M. Ji, H. Tang, "Application of chaos in simulated annealing," *Chaos, Solitons & Fractals*, vol. 21, no. 4, 2004, pp. 933-941.

[12] L. A. Zadeh, "Fuzzy sets," *Inform. Control*, vol. 8, 1965, pp. 338 – 353.

[13] J. M. Mendel, R. I. John and F. Liu, "Interval type-2 fuzzy logic systems made simple," *IEEE Transactions on Fuzzy Systems*, vol. 14, no. 6, Dec. 2006, pp.808-821.

[14] J. M. Mendel and R. I. John, "Type-2 fuzzy sets made simple," *IEEE Transactions on Fuzzy Systems*, vol. 10, April 2002, pp. 117-127.

[15] J. M. Mendel, *Uncertain Rule-Based Fuzzy Logic Systems*, Prentice-Hall, 2001.

[16] P. Melin, O. Castillo, *Hybrid Intelligent Systems for Pattern Recognition Using Soft Computing*; Springer Berlin / Heidelberg, 2005.

[17] J. M. Mendel, "Type -2 fuzzy sets and systems: an overview," *IEEE Transactions on Computational Intelligence*, vol. 2, no. 1, Feb. 2007, pp.20-29.

[18] H. Hagnis, "Type-2 FLCs: A new generation of fuzzy controllers," *IEEE Transactions on Computational Intelligence*, vol. 2, no. 1, Feb. 2007, pp.30-43.

[19] E. H. Mamdani, "Application of fuzzy logic to approximate reasoning using linguistic synthesis," *IEEE Transactions on Computers*, vol. 26, no. 12, Dec 1977, pp. 1182 – 1191.

[20] M. Sugeno, and T. Yasukawa, "A fuzzy-logic-based approach to qualitative modeling," *IEEE Transactions on Fuzzy Systems*, vol. 1, no. 1, February 1993, pp. 7 – 31.

[21] M. Sugeno and G.T. Kang, "Structure identification of fuzzy model," *Fuzzy Sets and Systems*, vol. 28, no. 1, pp. 15-23, 1988
[22] T.Takagi and M. Sugeno, " Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. on Systems, Man and Cybernetics*, vol.15, no. 1, pp. 116-132, 1985.
[23] Y. Tsukamoto, "An approach to fuzzy reasoning method," In Madan M. Gupta, Rammohan K, Ragade, and Ronald R. Yager, editors. *Advances in Fuzzy Set Theory and Applications*, pp. 137 – 149. North Holland, Amsterdam, 1979.
[24] T. C. Hsia. System identification: least-squares methods. *D.C. Heath and Company*, 1997.
[25] J.J. Buckley, " Fuzzy complex number," *Fuzzy sets Syst* ,vol.33, 1989, pp.333-345.
[26] —, " Fuzzy complex analysis I: Differentiation," *Fuzzy sets Syst.* , vol.41, 1991, pp.269-284.
[27] —, " Fuzzy complex analysis II: Integration," *Fuzzy sets Syst.*,vol.49, 1992, pp.171-179.
[28] J. J. Buckley and Y. Qu, "Solving linear and quadratic fuzzy equations," *Fuzzy Sets Syst.*, vol. 38, pp. 43–59, 1990.
[29] — , "Solving fuzzy equations: a new solution concept," *Fuzzy Sets Syst.*, vol. 39, pp. 291–301, 1991.
[30] L. Chen and K. Aihara, "Global searching ability of chaotic neural networks", *IEEE Trans. Circuits and Systems-I: Fundamental Theory and Applications*, vol. 46, no. 8, August, 1999, pp. 974-993.
[31] R. Konnur, "Additive chaotic forcing scheme for determination of the global minimum of functions," *Communications in Nonlinear Science and Numerical Simulation*, vol. 9, no. 5, October 2004, pp. 499-513.
[32] L. Chen and K. Aihara, "Transient chaotic neural networks and chaotic simulated annealing", in M. Yamguti(ed.), *Towards the Harnessing of Chaos.* Amsterdam, Elsevier Science Publishers B.V. pp. 347-352, 1994.
[33] L. Chen and K. Aihara, " Chaotic simulated annealing by a neural network model with transient chaos", Chaotic simulated annealing by a neural network model with transient chaos", IEEE Transactions on Neural Networks, vol.8, no. 6, 1995, pp.915-930.
[34] L. Wang, and K. Smith, "On chaotic simulated annealing", *IEEE Transactions on Neural Networks*, vol. 9, no.4, July, 1998, pp. 716-718.
[35] L. Wang and F. Tian, "Noisy chaotic neural networks for solving combinatorial optimization problems", *Proc. International Joint Conference on Neural Networks.* (IJCNN 2000, Como, Italy, July 24-27, 2000)
[36] F. Tian, L. Wang, and X. Fu, "Solving channel assignment problems for cellular radio networks using transiently chaotic neural networks", *Proc. International Conference on Automation, Robotics, and Computer Vision.* (ICARCV 2000, Singapore)
[37] L. Wang, "Oscillatory and chaotic dynamics in neural networks under varying operating conditions", *IEEE Transactions on Neural Networks,* vol. 7, no. 6, pp. 1382-1388, 1996.
[38] L. Wang and K. Smith, "Chaos in the discretized analog Hop.eld neural network and potential applications to optimization", *Proc. International Joint Conference on Neural Networks,* vol. 2, 1998, pp. 1679-1684,.

[39] T. Kwok, K. Smith and L. Wang, "Solving combinatorial optimization problems by chaotic neural networks", C. Dagli et al.(eds) *Intelligent Engineering Systems through Arti.cial Neural Networks* vol. 8, 1998, pp. 317-322.
[40] T. Kwok, K. Smith and L. Wang, "Incorporating chaos into the Hopfield neural network for combinatorial optimization", *Proc. 1998 World Multiconference on Systemics, Cybernetics and Informatics*, N. Callaos, O. Omolayole, and L. Wang, (eds.) vol. 1, 1998, pp. 646-651.
[41] I. Tokuda, K. Aihara, and T. Nagashima, "Adaptive annealing for chaotic optimization," *Phys. Rev. E*, vol. 58, 1998, pp. 5157–5160.
[42] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," Science, vol. 220, pp. 671-680, 1983.
[43] F. Busetti, "Simulated annealing overview", (http://www.geocities.com/francorbusetti/saweb.pdf).
[44] V. Cerny, "A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm," Journal of Optimization Theory and Applications, vol.45, no. 1 1985, pp.41-51.
[45] L. Ingber, "Simulated Annealing: Practice Versus Theory." *Math. Comput. Modelling* 18, 29-57, 1993.
[46] N. Metropolis; A. W. Rosenbluth; M. Rosenbluth,; A. H. Teller; and E. Teller, "Equation of State Calculations by Fast Computing Machines." *Journal of Chem. Phys.* vol. 21, no.6, 1953, pp.1087-1092,.
[47] P. J. M. van Laarhoven and E. H.L. Aarts, *Simulated Annealing: Theory and Applications,* D.Reidel Publishing Company, Boston.
[48] V. J. Rayward-Smith, I. H. Osman , C. R. Reeves , G. D. Smith , *Modern heuristic search methods*, Chichester ; New York : Wiley, c1996.
[49] H. Kantz and T. Schreiber, nonlinear time series analysis, *Cambridge University Press*, 2004.
[50] G. L. Baker and J. P. Gollub, Chaotic dynamics: an introduction, *Cambridge University Press*, 1990.
[51] R. M. May, "Simple mathematical models with very complicated dynamics," *Nature*, vol. 26, 1976, pp. 459-467.
[52] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Nat. Acad. Sci. USA*, vol. 81, May 1984, pp. 3088–3092.
[53] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems," *Journal of Biological Cybernetics*, vol. 52, no.3 , July 1985, pp. 141–152,.
[54] E. Ippen, J. Lindner, and W. L. Ditto, "Chaotic Resonance: A Simulation," *Journal of Statistical Physics*, vol. 70, no.1, January 1993, pp. 437-450.
[55] R. Rajamani, D. Levinson, P. Michalopoulos, J. Wang, K. Santhanakrishnan, X. Zou, "Adaptive cruise control system design and its impact on traffic flow", From Web Resource: http://www.cts.umn.edu/pdf/CTS-05-01.pdf.
[56] Y. Jiang, "On Ulam-von Neumann transformations", *Comm. in Mathematical Phys.* vol.172, no. 3, September, 1995, pp. 449-459.
[57] C.Z. Luo and H.H. Shao, "Evolutionary algorithms with chaotic mutations," *Control Decision* , vol. 15, 2000, pp. 557–560.

[58] L. Yang and T. Chen, "Application of chaos in genetic algorithms," *Commun. Theor. Phys.* vol. 38, 2002, pp. 168–172.
[59] M. Dorigo, T. Stützle. "The ant colony optimization metaheuristic: Algorithms, applications and advances," In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.
[60] V. Maniezzo, L. M. Gambardella, Fabio de Luigi, "Ant colony optimization," in G. C. Onwubolu, B. V. Babu, editors, *New optimization techniques in engineering*, Spring, 2004.
[61] A.J. Noest, " Discrete-state phasor neural nets," *Physical Review A*, vol.38, pp. 2196-2199, 1988.
[62] M.S. Kim, and C.C.Guest, "Modification of Backpropagation for complex-valued signal processing in frequency domain," *in Proc. Int. Conf. Neural* Networks, San Diego, CA, USA, June 17- 21,1990, vol. 3, pp. 27-31.
[63] H. Leung, and S. Haykin, " The Complex back propagation algorithm," *IEEE Trans. Signal Process*, vol. 39, 1991, pp.2101-2104.
[64] N. Benvenuto and F. Piazza, "On the complex backpropagation algorithm," *IEEE Trans. Signal Process*, vol. 40, 1992, pp.967-969.
[65] G.M. Georgiou, C. Koutsougeras, "Complex domain backpropagation," *IEEE Transaction on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 5, 1992, pp. 330 – 334.
[66] T.Nitta, "An analysis of the fundamental structure of complex-valued neurons," Neural Processing Letters, vol. 12, 2000, pp.239-246.
[67] T.Nitta, "On the inherent property of the decision of melodies by complex-valued network," *Neurocomputing*, vol. 50, 2003, pp.291-303.
[68] T. Kim and T. Adali, "Fully complex multi-layer perceptron network for nonlinear signal processing," Journal of VLSI Signal Processing Systems for Signal Image and Video Technology, vol. 32, 2002, pp. 29-43.
[69] T. Kim and T. Adali, "Approximation by fully complex multilayer perceptrons," *Neural Computation*, vol. 15, 2003, pp. 1641-1666.
[70] H. Akira, *Complex-Valued Neural Networks*, Springer, 2006.
[71] A. Malekzadeh-A, M. Akbarzadeh-T, "Complex-Valued Adaptive Neuro Fuzzy Inference System-CANFIS," *in Proc. 2004, World Automation Congress*, Seville, Spain vol. 17, June 28– July 1, 2004, pp. 477 – 482.
[72] Y. Li, Y. T. Jang, "Complex adaptive fuzzy inference systems," *Soft Computing in Intelligent Systems and Information Processing, Proceedings of the 1996 Asian*, 11-14 Dec. 1996, pp. 551 – 556.
[73] J. Man, Z. Chen, S. Dick, "Towards inductive learning of complex fuzzy inference systems", in Proc. *NAFIPS '07*, 24-27 June 2007, pp. 415-420.
[74] M. Gruber, *Regression Estimators a Comparative Study*, Academic Press Inc., 1990.
[75] P. R. Gill, W.Murray, and M. H. Wright, *Practical Optimization.* London: Academic Press, pp. 136-137, 1981.
[76] R. Hentschke, M. Johann and R. Reis, Blue Macaw: A Didactic Placement Tool Using Simulated Annealing, *IFIP International Federation for Information Processing*, vol.192, Springer Boston, 2005, pp.37-47.

[77] Z. Michalewicz, "Genetic algorithms, numerical optimization and constraints," in *Proc. 6th Int. Con. Genetic Algorithms*, 1995, pp.151-158. Morgan Kaufmann Publishers, inc..
[78] A. Homaifar, C. X. Qi, S. H. Lai, "Constrained optimization via genetic algorithms," *Simulation*, vol. 62, no.4, 1994, pp. 242-254.
[79] Himmelblau, D. "Applied Nonlinear Programming," McGraw-Hill. 1992.
Jan. 15,2003, pp. 733–745.
[80] E. Wan, "Time series data," www.cse.ogi.edu/~ericwan/data.html.
[81] A.S. Weigend, N.A. Gershenfeld, " Results of the time series prediction competition at the Santa FeInstitute," in Proc. IEEE International Conference on Neural Networks, vol.3,1993, pp.1786-1793.
[82] R. J. Hyndman, "Time series data library,"
(http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/).
[83] T.J. Cholewo, J.M. Zurada, "Sequential network construction for time series prediction," in *Proc. International Conference on Neural Networks*, vol.4, 1997, pp.2034-2038.
[84] M. Li, K. Mehrotra, C. Mohan, S. Ranka, "Sunspot numbers forecasting using neural networks," in *Proc. 5th IEEE International Symposium on Intelligent Control*, vol. 1, 1990, pp. 524-529.
[85] F.H.F. Leung, H.K. Lam, S.H. Ling, P.K.S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Transactions on Neural Networks*, vol. 14, no. 1, Jan. 2003, pp. 79 – 88.
[86] J. Tsai J. Chou , T. Liu, "Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm," *IEEE Transactions on Neural Networks*, vol. 17, no. 1, Jan. 2006, pp. 69 – 80.
[87] Hong X., Harris C.J., "Experimental design and model construction algorithms for radial basis function networks," *International Journal of Systems Science*, vol. 34, no. 1,
[88] R. J. Hyndman, A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, Volume 22, 2006, pp. 679 – 688.
[89] A. Weigend, "The Sante Fe Time Series Conpetition Data," http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html
[90] U. Huebner, N. B. Abraham, and C. O. Weiss, "Dimensions and entropies of chaotic intensity pulsations in a single-mode far-infrared NH3 laser," *Phys. Rev.* A 40, 1989, pp. 6354.
[91] U. Huebner, W. Klische, N. B. Abraham, C. O. Weiss, "On problems encountered with dimension calculations," *Measures of Complexity and Chaos*, Plenum Press, New York 1989, pp. 133.
[92] U. Huebner, W. Klische, N. B. Abraham, C. O. Weiss, "Comparison of Lorenz-like laser behavior with the Lorenz model," *Coherence and Quantum Optics VI*, Plenum Press, New York, 1989, pp. 517.
[93] H. H. Sargent, "A prediction for the next sunspot cycle," *in Proc. 28th IEEE Vehicular Technology Conference*, 1978. vol. 28, 22-24 March 1978, pp. 490 - 496
[94] T. J. Cholewo, J. M. Zurada, "Sequential network construction for time series prediction," *in Proc. International Conference on Neural Networks 1997,* vol. 4, 9-12 June 1997, pp. 2034 – 2038.

[95] J. Wu, M. Liu, "Improving generalization performance of artificial neural networks with genetic algorithms," *in Proc. IEEE International Conference on Granular Computing 2005*, vol. 1, 25-27 July 2005, pp. 288-291.
[96] T. T. Nguyen, C. P. Willis, D. J. Paddon, H. S. Nguyen, "A hybrid system for learning sunspot recognition and classification," *in Proc. International Conference on Hybrid Information Technology 2006*, vol. 2, Nov. 2006, pp. 257 – 264.
[97] T. Nobuhiko, H. van Dijk, "Combined forecasts from linear and nonlinear time series models," *International Journal of Forecasting*, vol. 18, 2002, pp. 421- 438.
[98] L. A. Zadeh, "Evolution of fuzzy logic - from past to future", *in Proc. IEEE International Conference on NAFIPS 2001,* Vancouver, Canada July 25-28, 2001.
[99] C. Schmid, "Dynamics of multidisplicinary and controlled Systems", (http://www.esr.ruhr-uni-bochum.de/rt1/syscontrol/main.html).
[100] N. Soylemezogolu, The Logic of Fuzziness, (http://www.math.harvard.edu/~hmb/issue2.1/FUZZY/fuzzy.html)
[101] J. M. Mendel, "Uncertainty, fuzzy logic, and signal processing," *Signal Proc. J.*, vol. 80, no. 6, pp. 913–933, 2000.
[102] D. Moses, O. Degani, H.-N. Teodorescu, M. Friedman, and A. Kandel, "Linguistic coordinate transformations for complex fuzzy sets," in *Proc. 1999 IEEE Int. Conf. Fuzzy Systems*, Seoul, Korea, Aug. 22–25, 1999, pp. 1340–1345.
[103] A. Kaufman and M. M. Gupta, *Introduction to Fuzzy Arithmetic*. New York: Van Nostrand Reinhold, 1985.
[104] H. T. Nguyen, A. Kandel, and V. Kreinovich, "Complex fuzzy sets: toward new foundations," in *Proc. 2000 IEEE Int. Conf. Fuzzy Systems*, San Antonio, TX, May 7–10, 2000, pp. 1045–1048.