Hybrid Soft Computing Approach to Identification and Control of Nonlinear Systems

by

Yuehui Chen

A Dissertation
Submitted to Graduate School of Science and Technology
in Partial Fulfillment of the Requirement for the Degree
of Doctor of Philosophy

Kumamoto University
February 2001

Acknowledgements

I would like to express my greatest appreciation to Professor Shigeyasu Kawaji for his patient guidance and encouragement during the doctoral degree course. As my supervisor, his insight, observations and suggestions helped me to establish the overall direction of the research and contributed immensely to the success of the work. I would like to appreciate to Professor Hiroshi Kashiwagi, Professor Ryozo Nakamura and Professor Masanao Ebata for their detailed comments and suggestions during the final phases of the preparation of this thesis.

I would like to render thanks to Dr. Shuzhi Sam Ge, Associate Professor of The National University of Singapore and Dr. Yonghong Tan, Associate Professor of The Guilin Institute of Electronic Technology for their discusses and helps about neural networks control theories and applications.

I would like to express my thanks to Professor Bo Yang, Dr. Bingqing Guo, Professor Kuifan Shi, Professor Jiwen Dong, Professor Xiaohong Wang and Dr. Jinping Li of Jinan University for their supports, advice and many helps.

I am also grateful to all the members of Professor Kawaji's research group for giving me a comfortable and active environment for pursuing my research.

Finally, I'd like also to dedicate this thesis to my parents and also my brothers and sisters, thanks for their endless encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Understanding of Soft Computing and Intelligent System

The currently existing complex plants cannot be accurately described by traditional rigorous mathematical models, and there are increasing needs for highly accurate control and autonomous behavior in control, robotics and artificial life communities. The conventional approaches for understanding and predicting the behavior of such systems based on analytical techniques can prove to be inadequate. These difficulties lead to a number of challenging problems, i.e., embed the human intelligence into a machine, because there is a huge gap between the human intelligence and the machine intelligence.

Scientist and researchers have benefit from the researches of the human/natural intelligence. A number of computational models have also been developed by imitating the human/natural intelligence, i.e., evolutionary computation (EC) realizes intelligence through the simulated evolution [1], artificial neural networks (ANNs) realize intelligence through the simulated behavior of neurons in brain [2], fuzzy logic (FL) realizes intelligence through the simulated behavior of human reasoning process [3], and the artificial immune systems realize intelligence through the simulated behavior of the immune-logical mechanisms with the antigens and the antibodies[4].

In order to cope with the difficulties mentioned above, an emerging framework- soft computing has been developed recently, which has the following properties:

- *Soft computing* is oriented towards the analysis and design of intelligent systems. It contains fuzzy logic, artificial neural networks and probabilistic reasoning including evolutionary algorithms, chaos theory and parts of machine learning and has the attributes of approximation and dispositionality;

- *Soft computing* is aimed at a formalization of the remarkable human ability to make rational decision in uncertain and imprecise environment;
- The constituents of soft computing are complementary rather than competitive. The experiments gained over the past decade have indicated that it can be more effective to use them in a combined manner, rather than exclusively;
- *Soft computing* is an open framework. This means that the newly created techniques come from the imitating of the human/natural intelligence always appropriated to be added to the soft computing framework.

Soft computing, according to the ideas of Prof. Zadeh, differs from conventional hard computing in that it is tolerant of imprecision, uncertainty, partial truth, and approximation. The guiding principle of soft computing is to exploit the tolerance for imprecision, uncertainty, partial truth, and approximation in order to achieve tractability, robustness and low cost solutions [3]. Here the flexible information processing capability and effective or efficient computation methodology are two key properties of soft computing.

So far, more and more researches and engineering applications show that soft computing can play a key role in the designing of the intelligent systems, i.e., in knowledge representation and processing, in sensor fusion, in prediction and classification, in the identification and control of nonlinear systems and robotics, in image and signal processing and so on. A recent survey paper has been discussed the hybrid soft computing systems and their industrial and commercial applications [58], in which some combinations of hybrid soft computing systems, such as fuzzy logic controller tuned by neural networks and evolutionary computing, neural network tuned by evolutionary computing or fuzzy logic system, and evolutionary computing tuned by fuzzy logic systems were given, and three applications in diagnostics, control and prediction were summarized. Also a growing number of consumer products, for example, washing machines and air conditioners, are released with the embedded soft computing techniques.

An intelligent system means that it has human-like abilities, e.g., recognition, prediction, imprecision reasoning, high-performance computation, self-learning, self-diagonal, self-repairing, tolerant of error, and higher autonomous behavior and so on. It is valuable to partition an intelligent system into different levels. We assume that an intelligent system is developed with a life-long period via learning and evolution not created suddenly. A beautiful intelligent system may appear once a complete body of the system has been constructed, in which each constituent of the body may not be intelligent but it has specified functions. In fact, some intelligent behaviors only exist in the harmonious body. Therefore, in order to construct an intelligent system, we not only need construct

or develop the separate constituent of the intelligent body, but also need construct or develop the organic relationship between the components.

The current researches in intelligent systems, e.g., intelligent control, intelligent agent, intelligent robotics, shown that only partial intelligence were extended or embedded to the system with the utilization of soft computing approaches, i.e., artificial neural networks are useful to handle the nonlinearities and unknown function approximation problems, based on fuzzy logic systems, expert's knowledge can be utilized to design intelligent systems, evolutionary algorithms are helpful to find global solutions in a complex search space. Up to date, the harmonious body of the intelligent system have not been understood efficiently. This is indeed a challenging problem.

1.2 Overview of Soft Computing in System Identification and Control

A survey of existing techniques of nonlinear system identification prior to 1980s is given by Billings [5], a survey of the structure detection of input-output nonlinear systems can be obtained in [6], and a recent survey of nonlinear black-box modeling in system identification can be found in Sjoberg et al. [7]. Several methods have been developed for the identification and control of nonlinear system, including nonlinear autoregressive moving average with exogenous (NARMAX), Hammerstein, Wiener or Hannerstein-Wiener structures, but these methods suffer the difficulty of representing the behavior of the system over its full range of operation [8].

In what follows, we briefly review the recently developed approaches of the identification and control of nonlinear systems by utilizing the soft computing methodologies.

1.2.1 Neural networks and its variations

ANNs are one of alternative methods for nonlinear system identification and control. The early researches in system identification and control using ANNs can be found in [9]-[14], in which multilayer perceptron (MLP), radial basis function networks, recurrent neural networks and back-propagation learning algorithm (gradient decent method) are usually employed to approximate the input/output map of nonlinear systems. The main problems it suffered are that it is a time-consuming procedure and the learnt network may be not optimal due to there is no a prior knowledge to select the proper network structure.

Identification

Several improvements have been made recently for dealing with these problems. In

[15], a robust training algorithm of ANNs is proposed in which dead zone technique is used to adjust the weights of ANNs in the presence of disturbance. In [16], high-order NN and recurrent NN are used for the identification of nonlinear systems and the global convergence, stability and robustness of learning laws are derived in mathematics. Guoping Liu et al. proposed an online identification method by using Volterra polynomial basis function network (VPBF) with a structure selection procedure and a recursive learning algorithm [17], and in [18] the multiobjective criteria and genetic algorithm are used for the structure selection and parameter optimization of the VPBF networks and Gaussian radial basis function networks, respectively.

Other researchers attempt to change the structure of ANNs to get accurate precision of identification. In [19], a functional link ANN (FL-ANN) is proposed in which the input pattern is enhanced by using the nonlinear functional expansion technique, the results show that FL-ANN performs as good as and in some case even better than MLP networks. In [20] a sum-of-product neural network (SOP-NN) is proposed, it can be viewed as a basis function network with a flexible form of the basis functions. Learning start with a small set of submodules and have new submodels added when it becomes necessary. The research shows that SOP-NN have excellent learning convergence characteristics and requires small memory space. D.T. Pham et al. recently successfully described the use of GA to train the modified Elman and Jordan recurrent neural networks for system identification [21][22]. In [23], a Bayesian-Gaussian neural network (BG-NN) and its learning algorithm is proposed for the system identification, in which the training of BG-NN is a minimization process to optimize the input factors rather than the connection weights plus thresholds of the back-propagation neural network or its variations and therefore could save a large amount of time in training. A comparative study of soft-computing methodologies in identification of robotic manipulators have been proposed in [24], in which four kinds of soft computing models: feedforward neural network architecture (FNN), RBF-NN, Rung-Kutta NN (RK-NN) and adaptive neuro-fuzzy inference systems (ANFIS) have been used for the system identification at same conditions. The results show that for the tracking error performance, ANFIS showed the best performance, and the RBF-NN and FNN are the simplest approaches in the sense of computational complexity.

Wavelets are alternative universal approximators; Pati and Krishnaprasad discussed using wavelets for linear system identification [43]. Wavelet networks have been investigated in [44]-[46] for nonlinear system identification. In [49], based on the orthogonal wavelets a system identification scheme is proposed, in which better accuracy of estimation is obtained by adding more terms (according to the regions of interest) to the wavelet based identifier, and these terms do not alter the coefficients of the existing terms. A

training algorithm for wavelet neural networks is proposed in [51], in which an original initialization procedure is presented that takes the locality of the wavelet functions into account.

Control

In the past several years, active research has been carried out in neural network control. Most of these works use neural networks as nonlinear models for the underlying nonlinearity [177][196][197]. For stable and efficient NN control, off-line training phases were usually required before the NN controller can be put into operation. To overcome such a problem, Lyapunov's stability theory was applied in the controller design and several adaptive NN control schemes were developed [170][199][198][200],[202]-[205]. To elude the instability during the on-line adaptation, radial basis functions are used [199] [201], where the network output is linearly parameterized, and the nonlinear system is for the special case of the affine system. The dynamic neural controller is proposed in [198], where Lyapunov approach is used to assure the input-to-state stability. In [200], the so called θ-adaptive neural networks are utilized to satisfy conditions that guarantee boundedness of the closed-loop systems, allowing a larger stability region compared with others [199]. It is valuable noting that the most of above neural control schemes need some types of matching conditions, i.e., the unknown nonlinearities appear in the same equation as the control input in a state-space representation. Recently, using the idea of adaptive backstepping, an interesting neural based adaptive controller without satisfying matching condition has been proposed [206], which also ensures the semi-global stability of the closed-loop systems. This design technique is further studied in [207]-[210].

Another active field of research is the combination of neural network and sliding mode control. Recently a number of neural network based sliding mode control schemes with guaranteed stability analysis have been proposed, e.g., in [211]-[213]. The advantage of this control scheme is that it is insensitivity to parameter variation and disturbances.

1.2.2 Fuzzy and/or neurofuzzy systems

Fuzzy and/or neurofuzzy systems is another alternative approaches for the identification and control of nonlinear systems due to the universal approximation capability of these systems. The early works in system identification and control by using fuzzy methods can be found in [25][26]. But there remains some problems to be solved, for example, how to automatically partition the input space for each variables, how many fuzzy rules are really needed for properly approximating the unknown nonlinear systems. As is well known, the curse-of-dimensionality is an unsolved problem in the fields of fuzzy systems, neurofuzzy networks, B-spline neural networks, wavelet neural networks and so on. In designing

of fuzzy and/or neurofuzzy systems, the structure discovery and parameter learning are also two coupled problems like the cases of designing of ANNs. The structure of a fuzzy and/or neurofuzzy system can be designed by expert's knowledge, fixed the rule base and the automatically find the rule base. The parameters used in fuzzy and/or neurofuzzy systems such that centers, widths, and slopes, and weights of neurofuzzy networks, can be optimized by gradient decsent method (error backpropagation) [27], reinforcement learning [28], approximate least square estimator [29] and evolutionary algorithms (EA) [30][31].

Recently some considerable developments in the area of system identification using fuzzy and/or neurofuzzy approaches have been achieved [32][33][51]. In [32], a method for determining the number of rules of fuzzy model is introduced by utilizing the concept of terminal attractors, and then a fast online learning rule to adjust free parameters is proposed. S. Barada et al. proposed a method of generating optimal adaptive neuro-fuzzy models [33], in which the structure of Takagi-Sugeno-Kang (TSK) fuzzy model is determined via a combination of modified mountain clustering algorithm, recursive least-squares estimation and a group method of data handling. FuGeNeSys is proposed by M. Russo [31], which is a kind of neurofuzzy system learned by Genetic Algorithm (GA). In [234], the use of genetic programming to identify the input variables, the rule base and the involved membership functions of a fuzzy model is proposed. Based on evolving fuzzy neural networks (EFu-NN), N. Kasabov recently proposed a new neurofuzzy model entitled evolving connectionist systems (ECOS)[34]-[36] in which incremental evolution, hybrid (supervised/unsupervised), on-line learning are used to build on-line, adaptive intelligent systems. Most recently, Y. Shi et al. discussed the limitations of conventional neuro-fuzzy learning algorithms and proposed a new learning algorithm, in which the advantages of new algorithms are that the tuning parameters of the fuzzy rules can be learned without changing the form of fuzzy rule table and the case of weak-firing or non-firing can be avoided [52][53]. Q. Gan and C. J. Harris proposed a modified algorithm for adaptive spline modeling of observation data (MASMOD) for determining the number of necessary B-splines and their knot positions. This research shown that fuzzy local linearization models have several advantages over local basis function expansion based models in nonlinear system modeling [59].

There exist two different types of fuzzy controller: the Mamdani type [235][236] and the Takagi-Sugeno type [25]. They mainly differ in the fuzzy rule consequent: a Mamdani fuzzy controller utilizes fuzzy sets as the consequent whereas a TS fuzzy controller employs linear functions of input variables. Significant effort has been made to analytical study Mamdani fuzzy controllers (e.g., [237]-[246]). In contrast, analytical results of TS fuzzy

controllers are still rather limited [247]-[249].

The combination of the fuzzy control and the PID control has been extended in [250]-[253] recently. In fact, a fuzzy controller is nothing but a nonlinear PID controller with time-dependent variable gains as discussed in [254].

Robust stability problem of fuzzy control systems has been discussed in [255]. An improved fuzzy gain-schedule controller is proposed in [256]. An output tracking problem by using fuzzy neural networks with guaranteed stability has been proposed in [257]. Evolutionary algorithms assistant fuzzy logic controller design can be found in [258]-[260]. Recent developments of the fuzzy sliding mode control schemes have been discussed in [229]-[231][261]-[263].

1.2.3 Evolutionary algorithms

There are also some works in the area of system identification and control by using evolutionary algorithms. K. Kristinn's research first showed that GA can be applied for system identification and control of both continuous- and discrete-time systems. In this research, GA is directly employed to identify the physical parameters or poles and zeros [37]. The genetic adaptive identification and control, genetic adaptive state estimation have been proposed and further researched in [217][218]. In [38], the author demonstrated that both a GA and the method of approximating nonlinearity with piecewise linears can be used to estimate a Hammerstain model. Systems identification by using genetic programming (GP) and PIPE algorithms have been reported in [39]-[41]. P.J. Angeline proposed multiple interacting program evolution algorithm which involved the simultaneous evolution of a set of equations represented as parse tree, and successfully applied it to system identification of nonlinear system [42]. Andrew proposed a system identification method by using genetic programming [219]. Howley used GP to get a sub-optimal control law for simulated specific spacecraft attitude maneuvers [220]. Dracopoulos used GP to derive an attitude control law to de-tumble a spinning satellite and discussed the stability by the Lyapunov method [221]. Dominic et al. used GP to design a discrete-time dynamic controller of chemical process that offers similar performance as PID controller [222]. Chellapilla used tree-structure-based evolutionary programs to derive nonlinear control laws for broom balancing problem and backing up a truck-and-trailer problem [223].

Most of the other works for system identification and controller design are the combination of evolutionary algorithms with other soft computing methodologies [54]-[56]. B.-T. Zhang proposed the evolutionary induction of sparse tree, in which a higher order neural network is coded as a neural tree, and the structure and parameters of higher order neural networks are evolved by GP and GA, respectively. In [139] a learning algorithm is

proposed, in which the parameter optimization and structure adaptation of MLP neural networks are dealt with a modified PIPE and a random search algorithm, respectively, and the results show that the proposed method can be usefully applied to evolve the non-regular ANNs. Yun L. and his group have been interested in the controller design of nonlinear systems, in which the NN controller or fuzzy/neuro-fuzzy controller are trained by a GA [214]-[216]. An evolutionary algorithm for optimizing local control of chaos is proposed in [233]. In this scheme, based on a Lyapunov approach, a linear control law and the state-space region in which this control law is activated are determined by an evolutionary algorithm.

1.3 Some of Unsolved Problems

Some of the unsolved problems related to this thesis are those as follows.

- Is there a unified framework in which various soft computing models can be calculated, evolved and evaluated?
- A bigger ANN may have poor generalization capability, meanwhile a smaller ANN cannot achieve the high level of approximation accuracy. Can we determine an optimal architecture of ANN automatically? Can we design a non-regular ANN automatically?
- In the designing of fuzzy model and neuro-fuzzy model, the problems are that how to automated partition the input space? How many fuzzy rules are really needed for properly approximating a nonlinear function? How to solve the Curse-of-dimensionality problem?
- Can we design a flexible hierarchical T-S fuzzy model automatically with a small rule base and a high level of approximation accuracy?
- In the basis function neural networks, how many of the basis functions are proper for approximating an unknown system, which type of basis function is good (globally or locally)? How to select these basis functions?
- Can we enhance the control performance of the traditional control system by embedding the soft computing approach into it ?
- Neural and fuzzy models have been used for the design of stable adaptive control system in different ways. Is there a common design principle for designing of soft computing based control systems with guaranteed stability?

1.4 Aim of the Thesis

The objective of this thesis is to investigate the ability of soft computing approaches to learn how to identification and control of nonlinear systems. A hybrid soft computing framework is constructed and applied to the identification and control of nonlinear systems.

This study makes following contributions. The main contributions are

- A hybrid learning algorithm for evolving various soft computing models is developed.
- A computational framework based on tree structural representation is developed. With this framework, a number of soft computing models can be constructed flexibly.
- The proposed soft computing models for the identification of nonlinear systems include those as follows
  - Additive model and re-construction of polynomials.
  - Evolving non-regular MLP neural networks.
  - Optimization the basis function neural networks (include the fuzzy basis function, the wavelet basis function, the Volterra polynomial basis function, the Gaussian radial basis function, the B-spine basis function, the recurrent fuzzy basis function and the local radial basis function networks).
  - Evolutionary design of hierarchical T-S fuzzy models.
- Some of soft computing based controller design principles are discussed.
- A probabilistic incremental program evolution (PIPE) based identification and control scheme is developed.
- A hybrid soft computing approach based direct feedback adaptive control scheme is developed.

1.5 Organization of the Thesis

In **chapter 2**, the basic elements of soft computing technique are listed and discussed which include the evolutionary algorithms (genetic algorithm, evolutionary programming, probabilistic incremental program evolution and random search algorithm), neural networks (the main topic focus on its training methods), and a simple T-S fuzzy model, all of these contents will be used in the following chapter.

In **chapter 3**, a unified framework is constructed in which various hybrid soft computing models can be calculated, evolved and evaluated. In this framework, the proposed hybrid soft computing models include: the additive model and reconstruction of polynomials, the non-regular multilayer perceptron, the basis function networks, and the hierarchical T-S fuzzy models. A hybrid learning algorithm is also proposed in which the architecture of the hybrid soft computing models is evolved by a modified probabilistic incremental program evolution (PIPE), and the parameters used in hybrid soft computing models are optimized by hybrid or non-hybrid parameter optimization strategy, respectively.

In **chapter 4**, Firstly, some common soft computing based controller design principle are discussed briefly. Then we proposed a new control scheme for nonlinear systems based on PIPE algorithm. Finally, based on the basis function neural networks a uniformly framework for control of affine and non-affine nonlinear systems is presented with the guaranteed stability analysis. Also in this research, we pay much attention to the online training methods of soft computing controller in ord er to gain the real implementation of the soft computing control schemes.

In **chapter 5**, the soft computing based identification and control schemes developed in Chapter 3 and 4 are applied to the drilling system. In order to control thrust force (cutting torque) in the drilling process, a number of thrust force (cutting torque) identification methods are developed. Based on the soft computing models of the thrust force, a neural control scheme of the thrust force is presented. Real time implementations show that the soft computing based estimation models of thrust force (cutting torque) are efficient and effective.

Finally in **chapter 6**, the results obtained in previous chapters are summarized, and a number of topics for the future research in this direction are given.

Chapter 2

A Review of Soft Computing Approaches

2.1 Evolutionary Algorithms and Random Search Algorithm

2.1.1 Genetic Algorithm

Genetic algorithms (GAs) are globally stochastic search technique that emulates the laws of evolution and genetics to try to find optimal solutions to complex optimization problems. GAs are theoretically and empirically proven to provide to robust search in complex spaces, and they are widely applied in engineering, business and scientific circles.

GAs are different from more normal optimization and search procedures in four ways [60]:

- GAs work with a coding of the parameter set, not the parameter themselves.
- GAs search from a population of points, not a single point.
- GAs use objective function information, not derivatives or other auxiliary knowledge, but with modifications they can exploit analytical gradient information if it is available.
- GAs use probabilistic transition rules, not deterministic rules.

Coding and Decoding

Coding referred to the representation of the parameter used in the optimization problem. The usually used coding methods in GAs are base-2, base-10 and floating-point coding

methods. In a base-2 representation, alleles (values in the position, genes on the chromosome) are 0 and 1. In base-10, the alleles take on integer values between 0 and 9. In floating-point representation, the alleles are real-valued number. In base-2 and base-10 representations, the relationship between the real value of a parameter and its integer representation can be expressed by

$$x = a + \bar{x}\frac{range}{resolution} \tag{2.1}$$

where x is the real value of the parameter, $\bar{x}$ is the integer value corresponding to the x, a is the lowest value assumed by $\bar{x}$, $range$ is the interval of definition of the parameters, and $resolution$ is the number that take in account the number of bits used, i.e., $2^{number\ of\ bits} - 1$.

Genetic Operators

A simple Genetic algorithm that yields good results in many practical problems consists of three genetic operators:

- *Reproduction* is a process in which individual strings are copied according to their objective or fitness function values. Fitness function can be imagined as some measure of profit, utility, or goodness to be optimizes. For example, in curve fitting problem, the fitness function cab be mean square error

$$Fitness = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(y_i, a_i))^2 \tag{2.2}$$

  where the y_i are experimental data, $f(y_i, a_i)$ is the function chosen as model and a_i are the model parameters to be optimized by GA. When GA is used to optimize an adaptive controller, the error and change in error information can be taken account into the designing of a proper fitness function. In general, operator of reproduction guarantee survival of the better individual to the next generation with a higher probability, which is an artificial version of natural selection.

- *Crossover* is a partial exchange of the genetic content between couples of members of the population. This task can be done in several different ways and it also depends on the representation scheme chosen. In integer representation, the simple way to do it, is to choose a random value with a uniform distribution as $[1, length\ of\ chromosome]$. This number represents a marker inside the two strings of bits representing the couple of chromosomes. It cuts both the chromosomes into

two parts. Then, the left or the right parts of the two chromosomes are swapped. This occurs in such a way that both the two new chromosomes will contain a part of the genetic information of both the parents. In floating-point representation, the crossover should be realized by

$$new_1 = a \cdot old_1 + (1 - a) \cdot old_2, \tag{2.3}$$

$$new_2 = (1 - a) \cdot old_1 + a \cdot old_2, \tag{2.4}$$

where new_1 and new_2 are the chromosomes after the crossover, old_1 and old_2 are the chromosomes before the crossover, a is a random number with uniform distribution in [0,1].

- *Mutation* is needed because, even through reproduction and crossover effectively search and recombine extant notions, occasionally they may become overzealous and lose some potentially useful genetic materials. In GA, the mutation operator protects against such an irrecoverable loss. In other words, mutation tries to escape from a local maximum or minimum of the fitness function, and it seeks to explore other areas of the search space in order to find a global maximum or minimum of the fitness function. In integer representation, the mutation of gene in a position of the chromosome is randomly changed form one integer to another. In floating-point representation, mutation will randomly change the value of the chromosome within a range of definition.

The flow chart of simple genetic algorithm can be seen in Fig. 2.1.

Design Concerns

When a GA is used to solve optimization problems, it is valuable to keep the following points in mind.

- Choice of an optimization problem and understand what you can change to achieve the solutions.
- Select a proper representation. Too detailed a representation increases computational complexity, while too coarse a representation decreases the accuracy of the problem.
- Try to use other genetic operators, e.g., elitist, this means that most fit individual will be copied into next generation without being perturbed by crossover or mutation. This usually leads to a fast convergence speed.

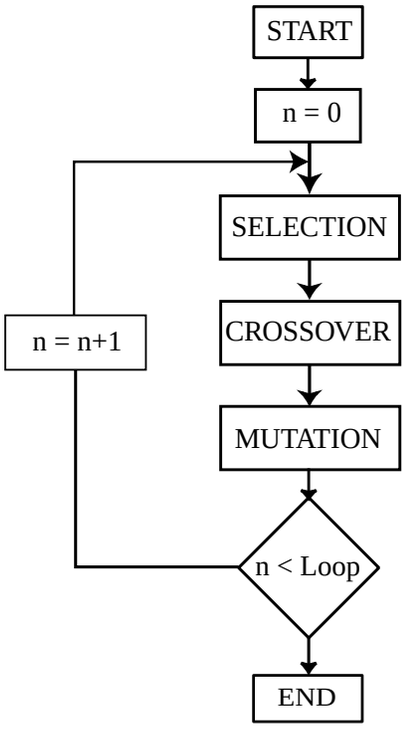


Figure 2.1: The flow chat of the genetic algorithm

2.1.2 Evolutionary Programming

EP is a population based random optimization algorithm, which imitate the principles of natural evolution that maintains a population of individuals for a specific generation [1][26]-[30]. According to our experiments, the convergence speed of EP depends largely on the search steps. Therefore in order to control the convergence speed and the performance of EP search, we introduced a scale factor α, and a dynamic factor $\sigma(k)$ shown in the Eq.(2.6) into the classical evolutionary programming. Our modified EP can be implemented as follows:

i) Generate the initial population of μ individuals, and set $k = 1$. Each individual is taken as a pair of real-valued vectors, (x_i, η_i), $\forall i \in \{1, \ldots, \mu\}$.

ii) Evaluate the fitness score for each individual of the population based on the fitness function.

iii) Each parent (x_i, η_i), $i = 1, \ldots, \mu$, creates a single offspring $(\bar{x}_i, \bar{\eta}_i)$ by

$$\bar{\eta}_i(j) = \eta_i(j) exp(\bar{\tau} N(0,1) + \tau N_j(0,1)) \tag{2.5}$$

$$\sigma(k) = \alpha * (1 - 0.9 * \frac{k}{K}) \tag{2.6}$$

$$\bar{x}_i(j) = x_i(j) + \sigma(k) * \bar{\eta}_i(j) * N_j(0,1), \tag{2.7}$$

for $j = 1, \ldots, n$, where $x_i(j), \bar{x}_i(j), \eta_i(j)$ and $\bar{\eta}_i(j)$ denote the j-th component of the

vectors $x_i, \bar{x_i}, \eta_i$ and $\bar{\eta_i}$, respectively. The scale factor α is used to control the step length of the search, the variable term $(1 - 0.9 * \frac{k}{K})$ showed in Eq.(2.6) is used for further tuning of the search precision, where k is the current generation number varying from 0 to K, K is the maximum generation number of EP search. $N(0,1)$ denotes a normally distributed random number. $N_j(0,1)$ indicates that the random number is generated anew for each value of j. And $\tau = (\sqrt{2\sqrt{n}})^{-1}$, $\bar{\tau} = (\sqrt{2n})^{-1}$.

iv) Calculate the fitness of each offspring.

v) Conduct pairwise comparison over union of parents and offspring. For each individual, Q opponents are chosen randomly from all the parents and offspring with an equal probability. For each comparison, if the individual's fitness is no smaller than the opponent's, it receives a win.

vi) Select the μ individual out of parents and offsprings, that have the most wins to be parents of the next generation.

vii) Stop if the number of EP search steps is reached; otherwise, $k = k + 1$ and go to step iii).

2.1.3 Random Search Algorithm

A random search algorithm scheme, random search with intensification and diversification (RasID) has been proposed for parameter optimization in [61]. RasID does an intensified search where it is easy to find good solutions locally, and a diversified search to escape from local minimum under a pure random search scheme. In doing so, a sophisticated probability density function (PDF) is used for generating search vectors. The PDF has two adjustable parameters, which are used to control the local search range and direction efficiently. Adjusting these parameters according to past success-failure information yields intensified and diversified search.

Given a parameter vector $\theta(k) = [\lambda_1(k), \lambda_2(k), \ldots, \lambda_N(k)]$, where k is random search step. Let $x(k) = [x_1(k), x_2(k), \ldots, x_N(k)]$ denotes the small random disturbance vector which is generated according a probability density function. The random search algorithm can be summarized as follows.

1. Choose an initial value of the parameter vector to be optimized randomly, $\theta(0)$, calculate the objective function, $F(\theta(0))$, and set $k = 0$.

2. Generate random search vector $x(k)$.

   - calculate $F(\theta(k) + x(k))$. If $F(\theta(k) + x(k)) < F(\theta(k))$, the current search is said to be success and then set $y^{(k)} = 1$ and $\theta(k+1) = \theta(k) + x(k)$. else,

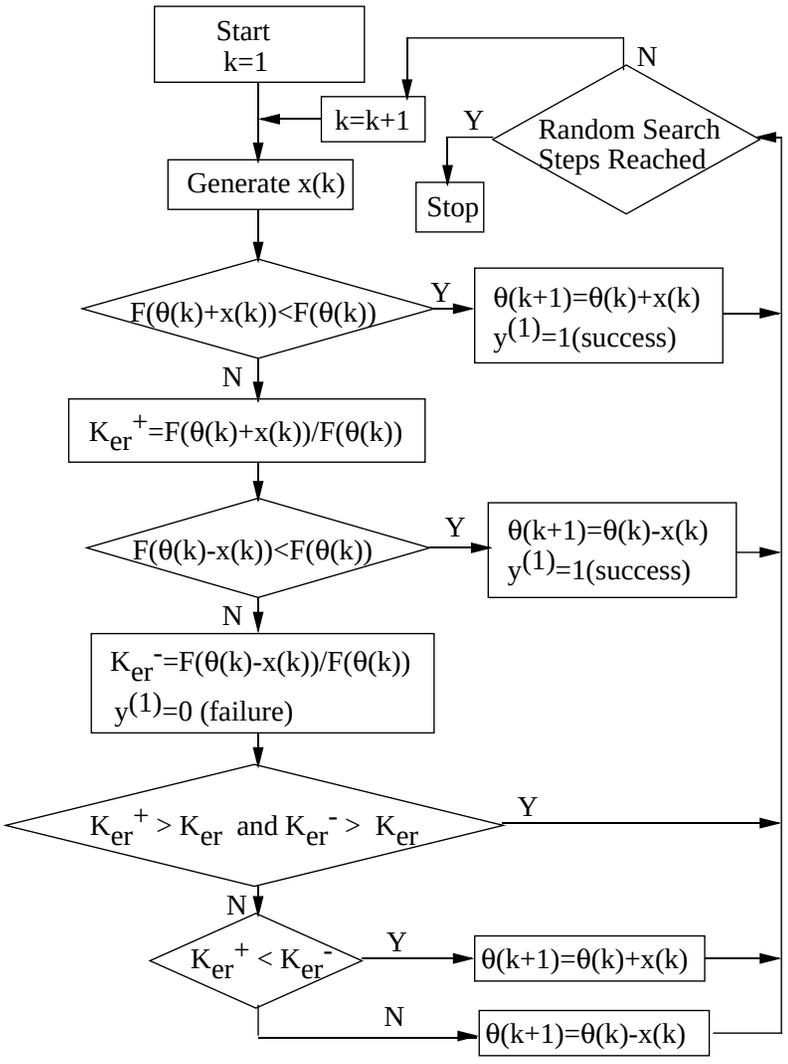


Figure 2.2: The flow chat of the random search algorithm

– calculate $F(\theta(k)-x(k))$. If $F(\theta(k)-x(k)) < F(\theta(k))$, the current search is said to be success too and then set $y^{(k)} = 1$ and $\theta(k+1) = \theta(k) - x(k)$. otherwise,

– the search is said to be failure and then set $y^{(k)} = 0$, and

$$\theta(k+1) = \begin{cases} \theta(k) & If\ K_{er}^{+} > K_{er}\ and\ K_{er}^{-} > K_{er} \\ \theta(k) + x(k) & If\ K_{er}^{+} < K_{er}^{-} \\ \theta(k) - x(k) & If\ K_{er}^{+} \geq K_{er}^{-} \end{cases} \tag{2.8}$$

where $K_{er} \geq 1$ is the maximum error ratio, K_{er}^{+} and K_{er}^{-} are defined by

$$K_{er}^{+} = \frac{F(\theta(k) + x(k))}{F(\theta(k))} \tag{2.9}$$

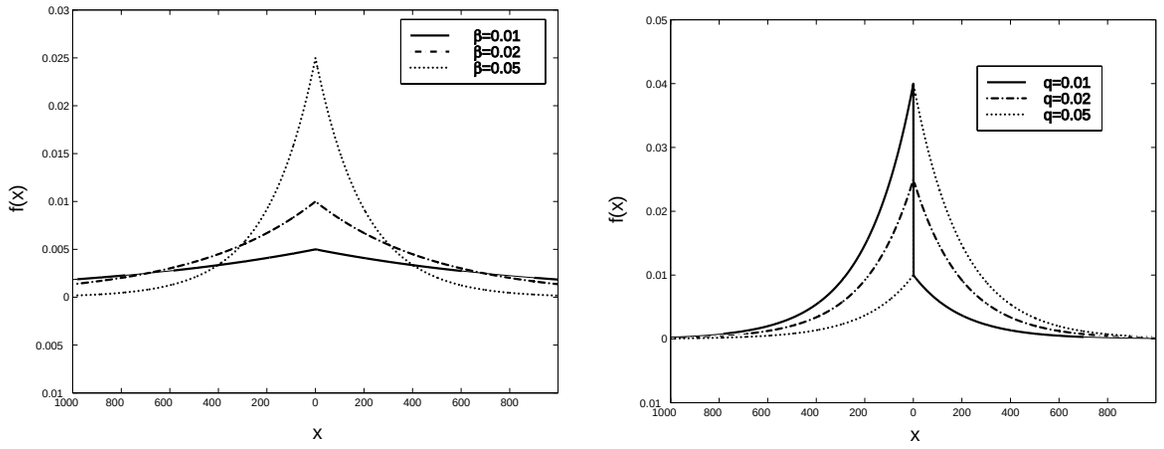


Figure 2.3: A specific probability distribution function in which the shape of the function depends on the parameters β and q. The larger the β is, the smaller the local search range is (left). The larger the q_m is, the higher the search probability in positive direction is (right).

$$K_{er}^{-} = \frac{F(\theta(k) - x(k))}{F(\theta(k))} \tag{2.10}$$

3. If satisfied solution found then stop, else set $k = k + 1$ and go to step 2.

The flow chart of the random search algorithm is also shown in Fig. 2.2.

It can be seen that the effectiveness of the random search depends largely on the random search vector $x(k)$. In usually used random search the Gaussian PDFs are used to generate the random search vector [63][64]. In RasID, the used PDF is

$$f(x_m) = \begin{cases} (1 - q_m)\beta e^{\beta x_m}, if\ x_m \leq 0 \\ q_m \beta e^{-\beta x_m}, if\ x_m > 0 \end{cases} \tag{2.11}$$

where adjustable parameters $q_m \in [0, 1]$ and β are used to control the range and direction of the intensification and the diversification search. Two example graphs of the PDF are shown in Fig. 2.3 (left and right), from which we can see that the larger the β is, the smaller the local search range is; the larger the q_m is, the higher the search probability in positive direction is. $q_m = 0.5$ means that there is same search probability in positive and in negative direction.

By using the above probability distribution function the random search vector $x(k)$ can be obtained as follows.

$$x_i(k) = \begin{cases} \frac{1}{\beta} ln\left(\frac{z_i(k)}{1-q_i(k)}\right), & if\ (0 < z_i(k) \leq 1 - q_i(k)) \\ -\frac{1}{\beta} ln\left(\frac{1-z_i(k)}{q_i(k)}\right), & if\ (1 - q_i(k) \leq z_i(k) < 1) \end{cases} \tag{2.12}$$

where $z_i(k)$ is the random real number uniformly distributed at [0,1], $q_i(k) = 0.5$ and $i = 1, 2, \ldots, N$.

The next problem is that how to adapted tuning the parameters β and $q_i(k)$ in order to quickly and efficiently find the global minimum of the search space. In our experiments, the parameter $q_i(k)$ is fixed as $q_i(k) = 0.5$. The parameter β is adaptive changed according to the following equation

$$\beta = \beta_0 + (\beta_1 - \beta_0)\, e^{-\phi I_{sf}} \tag{2.13}$$

where ϕ is designed to realize an intensified search and the index I_{sf} for diversified search, β_0 and β_1 are the lower and upper bound of β, respectively.

In addition, the adjustment of the parameters ϕ, I_{sf} and $q_i(k)$ are given in Eq. (2.14), (2.15) and (2.16), respectively.

$$\phi = \begin{cases} c_i\phi & P_{sf} > P_{sf0} \\ \phi & P_{sf} = P_{sf0}\ or\ \phi \leq \phi_{min} \\ c_d\phi & P_{sf} < P_{sf0}\ and\ \phi > \phi_{min} \\ \phi_0 & k = pre - specified\ integers \end{cases} \tag{2.14}$$

where $c_i \geq 1.0$, $0 < c_d \leq 1.0$ are two coefficients assigned with appropriate value, ϕ_0 and ϕ_{min} the initial and minimum values of ϕ.

$$I_{sf} = \begin{cases} & \phi > \phi_{min}\ or \\ I_{sf0}, & \left(y^{(k)} = 1\ and\ I_{sf} > I_{sfmax}\right)\ or \\ & k = pre - specified\ integers \\ I_{sf} - \triangle I_{sf1}, & y^{(k)} = 1\ and\ \phi \leq \phi_{min} \\ I_{sf} - \triangle I_{sf2}, & y^{(k)} = 0\ and\ \phi \leq \phi_{min} \end{cases} \tag{2.15}$$

where I_{sf0} is the initial value of I_{sf}, $\triangle I_{sf1}$ and $\triangle I_{sf2}$ are two appropriate positive values with $\triangle I_{sf1} < \triangle I_{sf2}$.

$$q_i(k) = \begin{cases} \alpha q_i(k), & if\ x_i(k) < 0\ or\ \frac{\partial^+ F}{\partial \lambda_i} > 0 \\ q_i, & if\ x_i(k) = 0\ or\ \frac{\partial^+ F}{\partial \lambda_i} = 0 \\ \alpha q_i + (1+\alpha), & if\ x_i(k) > 0\ or\ \frac{\partial^+ F}{\partial \lambda_i} < 0 \end{cases} \tag{2.16}$$

where $\alpha \in [0, 1]$ is an appropriate value and $\frac{\partial^+ F}{\partial \lambda_i(k)}$ is the ordered derivative of F for $\lambda_i(k)$.

2.1.4 Probabilistic Incremental Program Evolution (PIPE)

PIPE is a recent discrete method for automated program synthesis, which contains probability vector coding of program instructions, population based incremental learning and tree-coded programs like those used in variants of GP. The main principle of PIPE algorithm is that it increases the probability of the best program to be found by using the adaptive tuning of the probability distribution for choosing the proper instructions.

Given the instruction set I which contains a function set F=$\{f_1\ , f_2, \cdots, f_k\}$ and a terminal set $T = \{t_1, t_2, \cdots, t_l\}$. Each node $N_{d,w}$ of the tree contains a random constant $R_{d,w}$ or an instruction $I_{d,w}$. $P\left(I_{d,w}\right)$ denotes the probability of choosing instruction $I_{d,w} \in F \cup T$ at node $N_{d,w}$.

The general flow chart of PIPE algorithm is shown in Fig.2.4. The detailed procedure of the PIPE algorithm including initialization, population based learning, elitist learning and termination criterion are discussed in the following.

Initialization

1) Set the elitist program as NULL and its fitness value as a biggest positive real number of the computer at hand.
2) Set the initial value of parameters including population size PS, initial terminal probability P_T, elitist learning probability P_{el}, learning rate lr, fitness constant ε, overall mutation probability P_M, mutation rate mr, prune threshold T_P, and random constant threshold T_R.
3) Read the training data, the numbers of input and output and the number of training data.
4) Allocate the initial Probabilistic Prototype Tree (PPT) node, and set the initial probability of selecting the instructions as follows

$$P_{d,w}\left(I\right) = \frac{P_T}{l}, \forall I \in T \tag{2.17}$$

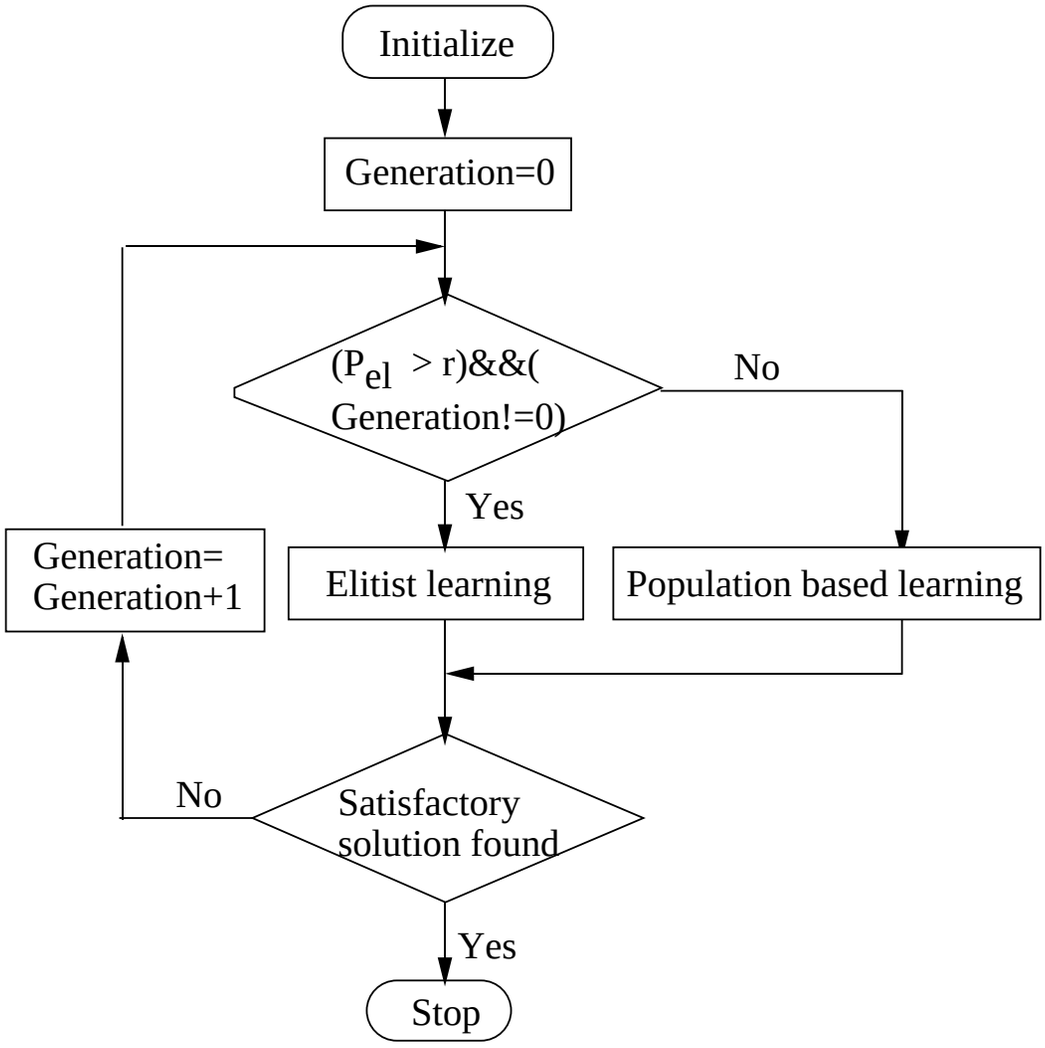


Figure 2.4: The flow chart of PIPE algorithm

$$P_{d,w}(I) = \frac{1 - P_T}{k}, \forall I \in F \tag{2.18}$$

where l and k are the numbers of instruction in the terminal set T and function set F, respectively.

5) Set the initial value of generation is equal to zero.

6) In every generation, create a uniformly random number r at [0,1], and one decision is made according to the values of elitist learning probability P_{el} and r. That is, if P_{el} is bigger than r and the generation is not equal to zero, then do elitist learning, otherwise do population based learning.

Population Based Learning

1) Create the initial population $P_{ROG_j}(0 < j \leq PS)$ by using PPT.

2) Calculate the fitness value $FIT\left(P_{ROG_j}\right)$, which reflects the program's performance on a given task.

3) Let P_{ROG_b} and $P_{ROG_{el}}$ be the best program of the current generation (best program) and the one found so far (elitist program), respectively. Define the probability and the target probability of best program as

$$P\left(P_{ROG_b}\right) = \prod_{\substack{I_{d,w}:used\ to \\ generate\ P_{ROG_b}}} P\left(I_{d,w}\right) \tag{2.19}$$

and

$$P_{TARGET} = P\left(P_{ROG_b}\right) + \left(1 - P\left(P_{ROG_b}\right)\right) \cdot lr \cdot \frac{\varepsilon + FIT\left(P_{ROGel}\right)}{\varepsilon + FIT\left(P_{ROG_b}\right)} \tag{2.20}$$

where $FIT\left(P_{ROG_b}\right)$ and $FIT\left(P_{ROGel}\right)$ denote the fitness of the best and elitist program. In order to increase the probability $P(P_{ROG_b})$, repeat the following process until $P\left(P_{ROG_b}\right) \geq P_{TARGET}$:

$$P\left(I_{d,w}\right) = P\left(I_{d,w}\right) + c^{lr} \cdot lr \cdot \left(1 - P\left(I_{d,w}\right)\right) \tag{2.21}$$

where c^{lr} is a constant influencing the number of iterations. This procedure is called $adapt - PPT - towards\left(P_{ROG_b}\right)$.

4) Define the mutation probability as

$$P_{M_p} = \frac{P_M}{(l+k) \cdot \sqrt{|P_{ROG_b}|}} \tag{2.22}$$

where $|P_{ROG_b}|$ denotes the number of nodes in program. All the probabilities $P\left(I_{d,w}\right)$ are mutated with probability P_{M_P} according to

$$P\left(I_{d,w}\right) = P\left(I_{d,w}\right) + mr \cdot \left(1 - P\left(I_{d,w}\right)\right) \tag{2.23}$$

5) Prune the subtrees attached to nodes that contain at least one probability vector above a predefined prune threshold T_P.

Elitist Learning

In order to search the previously discovered promising parts of the search space and increase the probability of elitist program, the elitist learning is realized by $adapt - PPT - towards\left(P_{ROG_{el}}\right)$. That is, get the elitist program, and calculate the probability

and the target probability of elitist program. Finally increase the probability of elitist program according to Eq.(2.21) until $P\left(P_{ROG_{el}}\right) \geq P_{TARGET}$.

Termination Criterion

The termination criterion of PIPE is that either a fixed number of iteration (time constraint) achieved or until found a solution with fitness better than the satisfactory fitness (quality constraint).

Remark

The instructions used in PIPE algorithm must be properly selected. The usually used instruction set for function approximation is

$$\{+, -, *, \%, sin, con, exp, rlog\}$$

where $+, -, *, \%, sin, con, exp$ and $rlog$ denote addition, subtraction, multiplication, protected division ($\forall x, y \in R, y \neq 0 : x\%y = x/y$ and $x\%0 = 1$), sine, cosine, exponent and protected logarithm ($\forall x \in R, x \neq 0 : rlog\left(x\right) = log\left(abs\left(x\right)\right)$ and $rlog\left(0\right) = 0$), and taking 2, 2, 2, 2, 1, 1, 1 and 1 arguments respectively. But due to different nonlinear system may have different characteristics, the further research about how to select the proper instructions for different nonlinear system is needed.

Example

As is well known, $\sin(x)$ have the standard Taylor Series

$$\sin\left(x\right) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, for\ x \in R \tag{2.24}$$

from which, the instruction set can be selected as

$$I = F \cup T = \{+, -, *, \%\} \cup \{x, R\}$$

where R is a random number $R \in [0, 1]$. Next, design a fitness function (sum of absolute error between the desired output and evolved output in this experiment) to evaluate the fitness of individual for the problem at hand. Finally, the best individual can be found after the end of PIPE run by using above learning algorithm. The control parameters of PIPE run for this experiment are shown in Table 2.1. A comparison of evolved sine function (dotted line) and true sine function (solid line) is shown in Fig.2.5.

Table 2.1: PIPE parameters

Population Size PS	10
Initial Terminal Probability P_T	0.8
Elitist Learning Probability P_{el}	0.01
Learning Rate lr	0.01
Fitness Constant ε	0.000001
Overall Mutation Probability P_M	0.4
Mutation Rate mr	0.4
Prune Threshold T_P	0.999999
Random Constant Threshold T_R	0.3

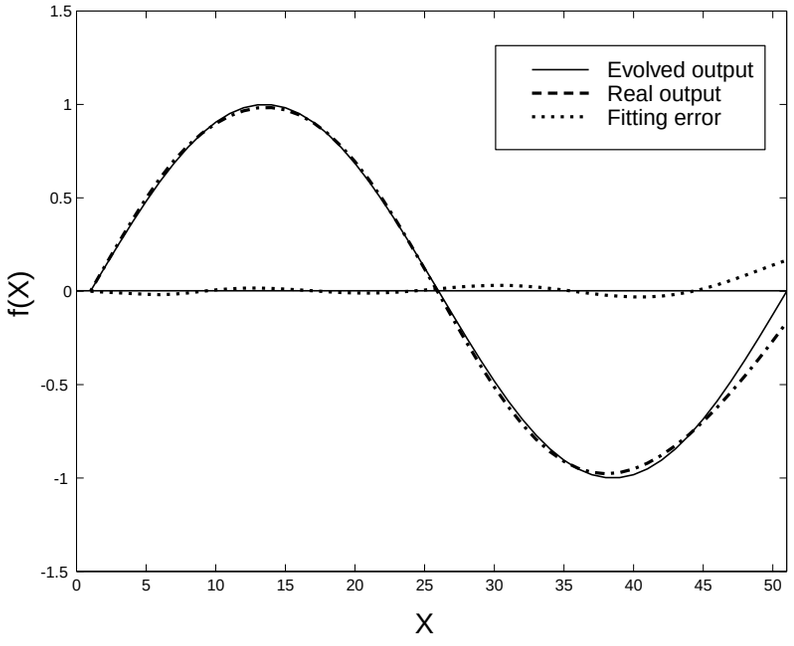


Figure 2.5: Curve fitting of the simple sine function by PIPE

2.2 Artificial Neural Networks

ANNs are computer programs or mathematical representation loosely inspired by the massively connected set of neurons that form the biological neural networks in brain.

The earlier experimental research works about ANNs include:

- McCulloch and Pitts studied the potential and capabilities of the interconnection of several basic components based on the model of a neuron in 1943 [64].
- The name of Perceptron was proposed by Rosenblatt in 1958 [65].
- The Perceptron was analyzed, its properties and limitations were given by Minsky

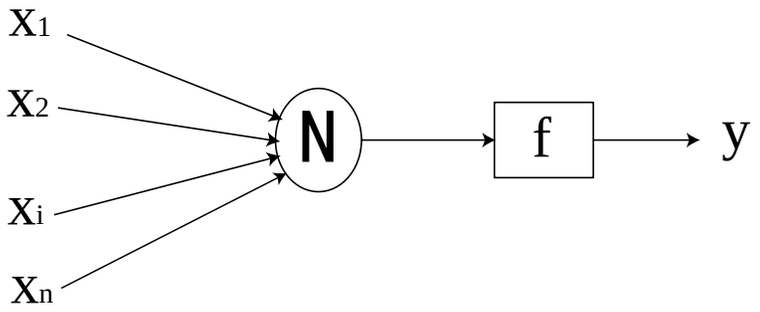


Figure 2.6: The processing unit of an ANN- a neuron

and Papert in 1969 [66].

- A number of neural processing models were proposed include learn matrix [67] in 1960's, associative content addressable memory (ACAM) networks [68], and cooperative-competitive neural network models in 1970's [69] .

- A particular dynamic network structure was proposed by Hopfield in 1982 [70].

ANNs are the alternative computing technologies that have been proven useful in a variety of function approximation, pattern recognition, signal processing, system identification and control.

In this section, a brief discuss of artificial neural networks (ANNs) is given. Only a few network topologies, tuning techniques and properties are referred according to their utility in following chapters.

The properties and functions of ANNs depend on:

- *The properties of single neuron model.* Currently, the usually used single neuron model is shown in Fig. 2.6, in which the output of the neuron is the weighted sum of its input x_i, $u_i = \sum_j \omega_{ij} x_j$, biased by a threshold value θ_i and passed through an activation function f.

$$y_i = f(\sum_j \omega_{ij} x_j - \theta_i) \tag{2.25}$$

  The activation function is selected differently in different applications. Some common choices of selecting activation function in function approximation, system identification and control are shown Table 2.2.

- *The topologies of the neural nets* are referred to the number of layers and the ways of the connections of the neurons. Different topologies or architectures of neural nets

Table 2.2: The Activation functions

Name	Formula
Sigmoid Function	$f(x) = \frac{1}{1+e^{-x}}$
Gaussian Function	$f(x) = exp(-\frac{x^2}{\sigma^2})$
Symmetric Sigmoid Function	$f(x) = \frac{1-e^{-x}}{1+e^{-x}}$
Hyperbolic Tangent Function	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Augmented Ratio of Squares	$f(x) = \frac{x^2}{1+x^2}sgn(x)$
Flexible Unipolar Sigmoid Function	$f(x,a) = \frac{2\|a\|}{1+e^{-2\|a\|x}}$
Flexible Bipolar Sigmoid Function	$f(x,a) = \frac{1-e^{-2xa}}{a(1+e^{-2xa})}$

should have different performance and different computational structure. Some network topologies are shown in Fig. 2.7.

- *Parameter tuning techniques* are referred to update the adjustable parameters including the weights, bias, and the parameters used in flexible activation functions. In general, the performance of a certain algorithm for adaptively tuning the parameters in neural network training stage can be evaluated by its convergence speed, stable properties, robustness and generalization ability. Recently, more and more researches have been focus on the stable training problem of ANNs. To some extend, neural network learning problem can be posed as a control problem, therefore, some of adaptive control strategies can be introduced directly into the training of ANNs.

Up to date, a number of kinds of neural network architectures have been developed. Among of them, the multilayer perceptron (MLP), the recurrent neural network (RNN), the fuzzy neural network (FNN), the radial basis function network (RBF) and the wavelet neural network (WNN) are most used neural networks in function approximation, system identification and controller design. In what follows, the MLP, recurrent neural networks and fuzzy neural networks are discussed briefly and the tuning strategies of MLP are analyzed in detail.

2.2.1 Multilayer Perceptron

MLP is a completely connected feedforward neural network as shown in Fig. 2.8. By properly selecting the number of hidden neurons, and the activation function in the hidden layer (i.e., hyperbolic tangent, f) and in the output layer (i.e., flexible sigmoid F), the

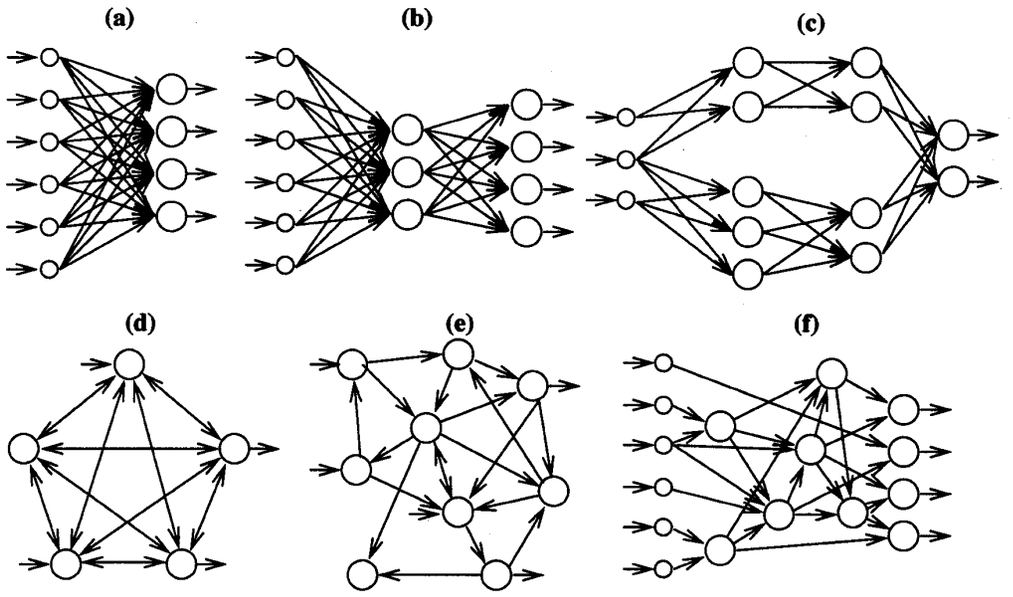


Figure 2.7: Some network topologies. (a) A fully connected single-layer perceptron (SLP). (b) A fully connected multilayer perceptron (MLP). (c) A modular MLP. (d) A fully connected recurrent network. (e) A sparsely connected recurrent network. (f) A feedforward network with sparse connections.

output of the MLP can be calculated as follows

$$y_k(t|\theta) = F_k(\sum_{j=0}^{n_h} w_{k,j} h_j(t)) = F_k(\sum_{j=1}^{n_h} w_{k,j} f_j(\sum_{l=0}^{n_i} w_{j,l} x_l(t)) + w_{k,0}) \tag{2.26}$$

where $y_k(t|\theta), (k = 1, 2, \ldots, n_o)$ is the k-th output of neural network, f_j is the j-th activation function for the unit j in the hidden layer and F_k specifies the activation function for output k. $h_j(t)$ is the j-th output of the hidden layer. $w_{k,j}$ and $w_{j,l}$ are the hidden-to-output and input-to-hidden layer weights, respectively. In addition, the bias are regarded as additional weights, i.e., $h_0(t) = x_0(t) = 1$.

Back-Propagation

Assume that the used activation functions in the hidden and output layers of the neural network are hyperbolic tangent and flexible bipolar sigmoid function (see Table. 2.2).

The derivatives of $F(x, a)$ with respect to the variable x and the parameter a can be obtained as

$$F^{'}(x, a) = 1 - a^2 F^2(x, a) \tag{2.27}$$

$$F^{*}(x, a) = \frac{1}{a}[F^{'}(x, a)x - F(x, a)] \tag{2.28}$$

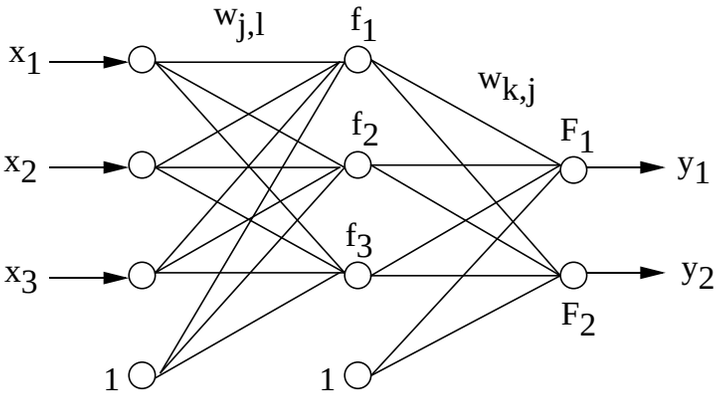


Figure 2.8: A 3-inputs 2-outputs three-layers perceptron

A batch version of momentum backpropagation algorithm can be easily derived as follows. Given a set of data

$$Z^N = \{[u(t), y(t)], t = 1, \ldots, N\} \tag{2.29}$$

Define the objective function as

$$J = \frac{1}{2N}\sum_{t=1}^{N}\sum_{k=1}^{n_o}(y_k(t) - y_k(t|\theta))^2 = \frac{1}{2N}\sum_{t=1}^{N}\sum_{k=1}^{n_o}\varepsilon_k^2(t,\theta) \tag{2.30}$$

The general gradient of the least squares criterion takes form

$$\frac{\partial J}{\partial \theta} = -\frac{1}{N}\sum_{t=1}^{N}\sum_{k=1}^{n_o}\frac{\partial y_k(t|\theta)}{\partial \theta}[y_k(t) - y_k(t|\theta)] \tag{2.31}$$

The partial derivatives of the network output with respect to the weights in hidden-to-output layer

$$\frac{\partial J}{\partial w_{k,j}} = -\frac{1}{N}\sum_{t=1}^{N} h_j(t)F_k^{'}(\sum_{j=0}^{n_h} w_{k,j}h_j(t))(y_k(t) - y_k(t|\theta)) = -\frac{1}{N}\sum_{t=1}^{N} h_j(t)\delta_k(t) \tag{2.32}$$

where

$$\delta_k(t) = F_k^{'}(\sum_{j=0}^{n_h} w_{k,j}h_j(t))(y_k(t) - y_k(t|\theta)) \tag{2.33}$$

Similarly, the partial derivatives of the network output with respect to the weights in input-to-hidden layer cab be obtained as

$$\frac{\partial J}{\partial w_{j,l}} = -\frac{1}{N}\sum_{t=1}^{N} x_l(t)f_j^{'}(\sum_{l=0}^{n_i} w_{j,l}x_l(t))\sum_{k=1}^{n_o} w_{k,j}\delta_k(t) = -\frac{1}{N}\sum_{t=1}^{N} x_l(t)\delta_j(t) \tag{2.34}$$

where

$$\delta_j(t) = f_j^{'}(\sum_{l=0}^{n_i} w_{j,l}x_l(t)) \sum_{k=1}^{n_o} w_{k,j}\delta_k(t) \tag{2.35}$$

The partial derivatives of the network output with respect to the parameter a_k in output layer cab be obtained as

$$\frac{\partial J}{\partial a_k} = -\frac{1}{N}\sum_{t=1}^{N} F_k^*(\sum_{j=0}^{n_h} w_{k,j}h_j(t))(y_k(t) - y_k(t|\theta)) \tag{2.36}$$

Therefore, the weights in hidden-to-output layer and in input-to-hidden layer, and parameters in output layer can be updated by

$$w_{k,j} = w_{k,j} + \eta_1 \frac{\partial J}{\partial w_{k,j}} + \alpha_1 \Delta w_{k,j} \tag{2.37}$$

$$w_{j,l} = w_{j,l} + \eta_2 \frac{\partial J}{\partial w_{j,l}} + \alpha_2 \Delta w_{j,l} \tag{2.38}$$

$$a_k = a_k + \eta_3 \frac{\partial J}{\partial a_k} + \alpha_3 \Delta a_k \tag{2.39}$$

Modification of the Back-Propagation

BP algorithm has some disadvantages, such as, slow convergence speed, sensitivity to initial conditions, trapping in local minima, instability if the settings of the learning rate are not appropriate. In general, the performance of a MLP network can be improved by following ways.

- *Approximation accuracy and generalization.* The higher approximation accuracy can be obtained by increasing the number of hidden layer neurons as the *universal approximation theory* has been said. But, if the number of hidden layer neurons is increased too large, the generalization should be decreased. Therefore, some researchers attempt to find the alternative methods for improving the approximation accuracy and generalization ability. Some of the techniques be and should be
  - Different neurons have different activation functions, especially, the use of flexible activation functions may increases the probability of finding a good MLP net for curve fitting.
  - The techniques of finding a sparse feedforward neural net will decrease the size of MLP net efficiently and improve the generalization ability.
  - In the training of MLP net, embed the structure information, i.e., the number of neurons in MLP net, into the optimization of the objective function should

improve the generalization ability of the MLP net. Simultaneously optimizing the objective function and the norm of the weight vector by a multiobjective optimization algorithm can also enhance the performance of the MLP net.

- *Convergence Speed.* Some of the methodologies have been proposed in order to speed up the convergence speed of the BP algorithm, such as, adaptive learning rate [71], quickpropagation [72], resilient propagation [73], backpropagation with adaptive gains [74], and delta-bar-delta method [75]. Heuristically, the search step should be increased when search process is far away from the global minima and decreased if the search is near to the minima. There is no common recognition now about what can be based for setting an adaptive learning rate. Some methods based on the network architecture and training data, some methods based on the error function. Most recently, the learning rate adaptation methods have been summarized by G.D. Magoulas with three algorithms [76]. These are

  - *Algorithm-1: BP with Adaptive Learning Rate*

    * 1. Set $\eta = \eta_0, m = 1$, and go to the next step.
    * 2. If $E(\omega^k - \eta g(\omega^k)) \leq -\frac{1}{2}\eta \| \bigtriangledown E(\omega^k)\|^2$, go to step 4; otherwise, set $m = m + 1$ and go to the next step.
    * 3. Set $\eta = \frac{\eta_0}{q^{m-1}}$, and return to step 2.
    * 4. Set $\omega^{k+1} = \omega^k - \eta g(\omega^k)$.
    * 5. If the convergence criterion $E(\omega^k - \eta g(\omega^k)) \leq \varepsilon$ is met, then terminate; otherwise go to next step.
    * 6. If $k < MIT$, increase k and begin recursion; otherwise terminate.

    where ω^k is the weight vector at search step k, q is reduction factor, MIT is the maximum number of allowed iterations. E is the batch error measure defined as the sum-of-squared-differences error function over the entire training set. $g(\omega)$ defines the gradient $\bigtriangledown E(\omega)$ of the sum-of-squared-differences error function E at ω.

  - *Algorithm-2: BP with Adaptation of a Self-Determined Learning Rate*

    * 1. Set $m = 1$, and go to the next step.
    * 2. Set $\eta_0 = \frac{E(\omega^k)}{\|\bigtriangledown E(\omega^k)\|^2}$; also set $\eta = \eta_0$.
    * 3. If $E(\omega^k - \eta g(\omega^k)) - E(\omega^k) \leq -\frac{1}{2}\eta \| \bigtriangledown E(\omega^k)\|^2$, go to step 5; otherwise, set $m = m + 1$ and go to the next step.
    * 4. Set $\eta = \frac{\eta_0}{q^{m-1}}$, and return to step 3.

* 5. Set $\omega^{k+1} = \omega^k - \eta g(\omega^k)$.
* 6. If the convergence criterion $E(\omega^k - \eta g(\omega^k)) \leq \varepsilon$ is met, then terminate; otherwise go to next step.
* 7. If $k < MIT$, increase k and begin recursion; otherwise terminate.

– *Algorithm-3: BP with Adaptive Learning Rate for Each Weight*

* 1. Set $\gamma = \gamma_0, m = 1$, and go to the next step.
* 2. If $k \geq 1$ set $\Lambda_i^k = \frac{|\partial_i E(\omega^k) - \partial_i E(\omega^{k-1})|}{|\omega_i^k - \omega_i^{k-1}|}$, $i = 1, 2, \ldots, n$; otherwise set $\Lambda^k = \eta_0^{-1} I$.
* 3. If $E(\omega^k - \gamma diag\{\frac{1}{\Lambda_1^k}, \ldots, \frac{1}{\Lambda_n^k}\} \bigtriangledown E(\omega^k)) - E(\omega^k) \leq -\frac{1}{2}\gamma \| diag\{\frac{1}{\Lambda_1^k}, \ldots, \frac{1}{\Lambda_n^k}\} \bigtriangledown E(\omega^k)\|^2$, go to step 5; otherwise, set $m = m + 1$ and go to the next step.
* 4. Set $\gamma = \frac{\gamma_0}{q^{m-1}}$, and return to step 3.
* 5. Set $\omega^{k+1} = \omega^k - \gamma diag\{\frac{1}{\Lambda_1^k}, \ldots, \frac{1}{\Lambda_n^k}\} \bigtriangledown E(\omega^k)$.
* 6. If the convergence criterion $E(\omega^k - \gamma diag\{\frac{1}{\Lambda_1^k}, \ldots, \frac{1}{\Lambda_n^k}\} \bigtriangledown E(\omega^k)) \leq \varepsilon$ is met, then terminate; otherwise go to next step.
* 7. If $k < MIT$, increase k and begin recursion; otherwise terminate.

where γ is the relaxation coefficient.

- *Stable Training.* As it is well known that if the setting of the learning rate is not appropriate the unstable problem may occur in the training of MLP by using backpropagation. Some techniques have been proposed in order to cope with this problem recently. St. Maruster discussed the stability problem of gradient-like method in the mathematics sense [77]. A training strategy for computationally intelligent systems based on variable structure systems with stability analysis in Lyapunou sense has been proposed in [78][79].

- *Robustness.* MLP is usually referred to as a universal approximator. Nevertheless, if the used training data are corrupted by noise, such as outliers, standard back-propagation learning scheme may not always come up with acceptable performance. A number of robust training algorithms have been proposed for dealing with the specific problems [15][80].

Second Order Training Methods

Second order gradient methods make use of second derivations of the error function with respect to the weights. These derivations consist of the Hessian matrix. The Hessian matrix contains information about how the gradient changes in different directions in

weight space. The mainly used second order methods include *Newton, Gaussian Newton, Levenberg-Marquardt, Quickprop and Conjugate Gradient Descent* methods. The Levenberg-Marquardt algorithm is briefly described as follows.

It is known that an algorithm with second-order convergence, such as Newton's method can greatly improve the convergence of the optimization. The parameter update rule of Newton's algorithm is

$$\omega_{i+1} = \omega_i - \left[\frac{\partial^2 J}{\partial \omega_i^2}\right]^{-1} \frac{\partial J}{\partial \omega_i} \tag{2.40}$$

where J is the objective or cost function, $\frac{\partial^2 J}{\partial \omega_i^2} = [\frac{\partial f}{\partial \omega_i}][\frac{\partial f}{\partial \omega_i}]^T + \delta(\omega_i)$ is the Hessian matrix that contains the second-order derivative terms. If the higher-order derivative term $\delta(\omega_i)$ is assumed to be omitted, then the algorithm becomes the so-called Gauss-Newton method, i.e.,

$$\omega_{i+1} = \omega_i - \left[\left(\frac{\partial f}{\partial \omega_i}\right)\left(\frac{\partial f}{\partial \omega_i}\right)^T\right]^{-1} \frac{\partial J}{\partial \omega_i} \tag{2.41}$$

The main problem in using Newton's and Gauss-Newton's methods is that these methods may have ill-conditioning if Hessian matrix is close to or is singular. The Levenberg-Marquardt modification to Gauss-Newton algorithm is to introduce a factor $\lambda \geq 0$ into the algorithm, i.e.,

$$\omega_{i+1} = \omega_i - \left[\left(\frac{\partial f}{\partial \omega_i}\right)\left(\frac{\partial f}{\partial \omega_i}\right)^T + \lambda I\right]^{-1} \frac{\partial J}{\partial \omega_i} \tag{2.42}$$

The factor λ can be changed from zero to a very large value. If λ is very small, the algorithm is reduced to being the Gauss-Newton method. While λ is sufficiently large, the algorithm approaches a steepest descent search. Moreover, λ can enhance the numerical stabilization if the algorithm is converging towards a saddle point where $\partial f/\partial \omega_i$ may approach zero. In this singular case, $\lambda \neq 0$ will considerably improve the numerical stability. In detail, the Levenberg-Marquardt algorithm for training MLP may be implemented as follows.

- 1. Select an initial parameter vector $\theta^{(0)}$, and an initial value $\lambda^{(0)}$.
- 2. Determine the search direction from $[R(\theta^{(i)}) + \lambda^{(i)I}] = -G(\theta^{(i)})$.
- If $r^{(i)} > 0.75$, then $\lambda^{(i)} = \frac{\lambda^{(i)}}{2}$; if $r^{(i)} < 0.25$, then $\lambda^{(i)} = 2\lambda^{(i)}$.

- If $J(\theta^{(i)} + f^{(i)}) < J(\theta^{(i)})$ then accept $\theta^{(i+1)} = \theta^{(i)} + f^{(i)}$ as a new iterate and let $\lambda^{(i+1)} = \lambda^{(i)}$.

- If the stopping criterion is not satisfied go to step 2.

where $f^{(i)}$ is search direction at search step i, and $G(\theta^{(i)})$ and $R(\theta^{(i)})$ are defined as follows. Let $L^{(i)}(\theta) = \frac{1}{2N}\sum_{t=1}^{N}\varepsilon^2(t,\theta)$ and $\psi(t,\theta) = \frac{\partial y(t|\theta)}{d\theta}$, then

$$G(\theta^{(i)}) = \frac{dL^{(i)}(\theta)}{d\theta}\mid_{\theta=\theta^{(i)}} = \frac{1}{N}\sum_{t=1}^{N}\psi(t,\theta^{(i)})[y(t) - y(t|\theta^{(i)})] \tag{2.43}$$

$$R(\theta^{(i)}) = \frac{d^2L^{(i)}(\theta)}{d\theta^2}|_{\theta=\theta^{(i)}} = \frac{1}{N}\sum_{t=1}^{N}\psi(t,\theta^{(i)})\psi^T(t,\theta^{(i)}). \tag{2.44}$$

Recursive Least Square (RLS)

MLP net can also be trained by recursive least square algorithm both in offline and online ways as some researchers have been pointed [81][82]. Where a simple three layer multi-input single-output MLP net is online trained by RLS in order to show the art of the technique.

Suppose the output of the MLP net is

$$y(t) = \sum_{j=0}^{n_h} w_{1,j}h_j(t) = \sum_{j=1}^{n_h} w_{1,j}f_j(\sum_{l=0}^{n_i} w_{j,l}x_l(t)) + w_{1,0} \tag{2.45}$$

where $w_{1,j}$ denotes the hidden-to-output layer weights, 1 denotes that there is only one output of the MLP net. $h_j(t)$ denotes the outputs of the hidden layer.

Define an objective function as

$$E(t) = \frac{1}{2}e^2(t) = \frac{1}{2}(y(t) - y_r(t))^2 \tag{2.46}$$

where $y_r(t)$ and $y(t)$ are the desired and model output at time t.

Firstly, the regression vectors of input-to-hidden and hidden-to-output layer can be formed as

$$\phi_h^T(t) = [x_1(t), x_2(t), \ldots, x_{n_i}(t), 1] \tag{2.47}$$

$$\phi_o^T(t) = [h_1(t), h_2(t), \ldots, h_{n_h}(t), 1] \tag{2.48}$$

where n_i and n_h are the number of neurons in the input and hidden layer, respectively.

The weights in the input-to-hidden and the hidden-to-output can be formed the following vector form

$$W_{h,l}^T(t) = [w_{h1l(t)}, w_{h2l}(t), \ldots, w_{hn_il}(t), w_{0,l}(t)] \tag{2.49}$$

$$W_{o,1}^T(t) = [w_{o11(t)}, w_{o21}(t), \ldots, w_{on_h1}(t), w_{o,0}(t)] \tag{2.50}$$

where h and o denote the hidden and output layer, respectively.

Then, the sum of the l-th hidden unit and the output unit are

$$S_{hl}(t) = \phi_h^T(t) W_{hl}(t) \tag{2.51}$$

$$S_{o1}(t) = \phi_o^T(t) W_{o1}(t) \tag{2.52}$$

Secondly, define the errors for the l-th hidden units and the output unit as

$$e_{hl}(t) = S_{hl}^*(t) - S_{hl}(t) \tag{2.53}$$

$$e_o1(t) = S_{o1}^*(t) - S_{o1}(t) \tag{2.54}$$

where $S_{hl}^*(t)$ and $S_{o1}^*(t)$ are the desired values and unknown. So the above errors cannot be calculated. But based on the plant Jacobian and the objective function, the errors can be approximated by

$$e_{o1}(t) \cong -\frac{\partial E(t)}{\partial S_{o1}(t)} \cong e(t) J(t) \tag{2.55}$$

where $J(t)$ is the plant Jacobian, it can be calculated in real implementation as follows

$$J(t) = \frac{\partial F(t)}{\partial u(t-1)} \cong sign(F(t) - F(t-1)) \cdot sign(u(t-1) - u(t-2)) \tag{2.56}$$

The error for the hidden layer can be obtained as

$$e_{hl}(t) \cong -\frac{\partial E(t)}{\partial S_{hl}(t)} \cong e_{o1}(t) w_{ol1}(t) f_{hl}^{'}(t) \tag{2.57}$$

where $f_{hl}^{'}$ is the derivative of the hidden layer activation functions.

Finally the weights of MLP net can be update according to

$$W_{hl}(t) = W_{hl}(t-1) + P_h(t) \phi_h(t) (e_{hl}(t)) \tag{2.58}$$

$$W_{o1}(t) = W_{o1}(t-1) + P_o(t) \phi_o(t) (e_{o1}(t)) \tag{2.59}$$

where P_h and P_o are the P-matrices of the hidden and output layers respectively. The P-matrice is update according to

$$P(t) = \frac{1}{\lambda}P(t-1)\{I_m - \frac{\phi(t)\phi^T(t)P(t-1)}{\lambda + \phi^T(t)P(t-1)\phi(t)}\} \tag{2.60}$$

Evolutionary Algorithm based Training

Some global optimization techniques like evolutionary programming, simulated annealing [83] and genetic algorithms have also been used for the training of ANNs. A recent survey of such research can be found in X. Yao [84]. It has been shown that the binary coding scheme used in GA is neither necessary nor beneficial [85][86]. In addition, Fogel and Ghozeil [87] showed that under some fairly general assumptions, there are fundamental equivalences between various representations. Several successful studies using real values instead of binary coding scheme include Montana and Davis [88] and Sexton et al., [89] for GA and porto et al., [90] and Saravanan et al., [91] for other evolutionary algorithms.

Most recently, R.S. Sexton et al., compared the use of BP and the GA for training of MLP net [92]. Their empirical results shown that the GA is superior to BP in effectiveness, ease-of-use and efficiency for training MLP net for the five chaotic time series prediction problems. Where *effectiveness* refers to the accuracy of each algorithm to estimate the true functional form. *Ease-of-use* deals with the effort needed for optimal algorithm settings for the problems at hand. *Efficiency* of an algorithm is computed by comparing the CPU time needed for converging upon the best found solutions.

Further researches are needed from a wide range of learning algorithms in order to evaluate the performance of various tuning strategies for MLP net training, and evaluate the applicability, generalization ability of each training algorithm for various scientific and engineering problems.

Control Strategy Based Training

Viewing the network learning process as the control problem results in the use of some control strategies to optimize and control the learning process of NN training. The basic considerations behind this technique are those as follows:

- The objective of the MLP training is that find a global minima in the weight space for a certain objective criterion through an iteration procedure. This forms the aim of the control problem of the MLP training.

- In order to reduce the error function, an iteration process is usually needed in the MLP training, but, the iteration process may become unstable under some

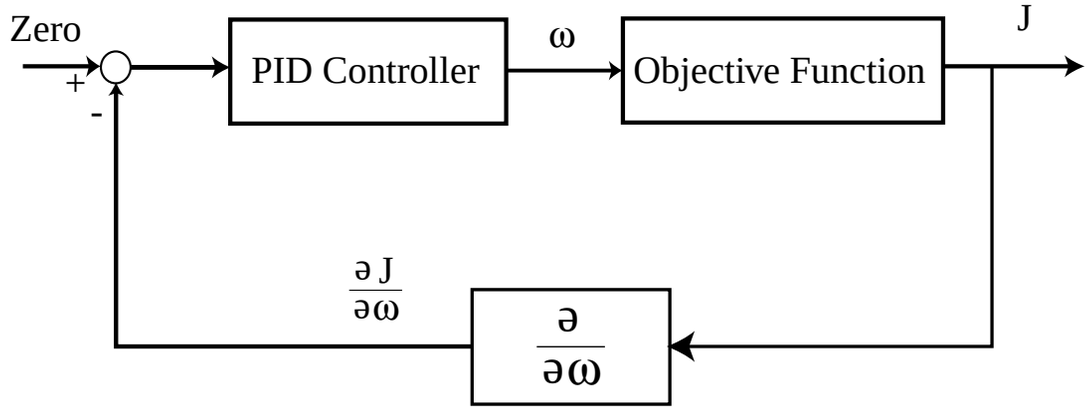


Figure 2.9: Architecture of PID gradient optimization

conditions, i.e., the large learning rate in the backpropagation training. In these cases, the aim of the control may not be achieved or it is time consuming process.

- A natural problem arises like this: Can we control the process of MLP training by using some control strategies in order to achieve the control objective fastest and stably ?

Some characteristics of the control problem of the MLP training, such as observerability, controllability, robustness and stability have not been perfectly understood now. But some encouraging research works have been made recently. Typical examples of this technique are

- *Online PID Gradient Optimization* [93]. In this approach, a PID controller is embedded into the learning process of the MLP training as shown in Fig. 2.9. The output of the PID controller is the weight ω, the controlled process is objective function, i.e., $J(t) = \frac{1}{2}(y(t) - \hat{y}(t))^2$ in instantaneous gradient descent algorithm and $J = \frac{1}{2}\sum_{t=1}^{K}\sum_{i=1}^{P}(y_t(i) - \hat{y}_t(i))^2$ for batch version of gradient method. The feedback signal is the gradient of the objective function with respect to weight ω, i.e., $\partial J(t)/\partial\omega$. The set point of the control system is zero which is the target of the control system since it will force the feedback signal $\partial J(t)/\partial\omega$ to track this target.

  It is known that a stable and fast control response can be obtained by choosing proper gains of the PID controller. Motivated by PID control, the gradient descent algorithm with incremental PID tuning can be formulated as follows:

$$\begin{aligned}\omega(t) = \omega(t-1) - K_i\frac{\partial J(t)}{\partial\omega} - K_p\left[\frac{\partial J(t)}{\partial\omega} - \frac{\partial J(t-1)}{\partial\omega}\right] \\ -K_d\left[\frac{\partial J(t)}{\partial\omega} - 2\frac{\partial J(t-1)}{\partial\omega} + \frac{\partial J(t-2)}{\partial\omega}\right]\end{aligned} \tag{2.61}$$

where K_p, K_i and K_d are proportional, integral and derivative gains of the PID controller. Obviously, if K_p and K_d are zero, the algorithm is the standard instantaneous gradient descent algorithm. Therefore, this algorithm provides more freedom to improve the convergence of MLP training by choosing proper gains.

- *Neural Networks Learning with Sliding Mode Control* [94]. In this scheme, sliding mode theory, which is a simple but robust control technique is used to optimize the direction of weight updates in the standard backpropagation algorithm. Two key points of this technique are:
  - The absolute value of the error is used instead of the actual error of the standard backpropagation.
  - Since the absolute value is considered above, the sign of $\Delta\omega$ is now given by the sign of the sliding surface at the current state. Since the sign of the surface defines the control action in sliding mode control, the learning problem has now been reduced to a standard sliding mode control problem. Network learning is now governed by the control actions of the sliding mode algorithm. Convergence in the learning process is therefore guaranteed because general sliding mode control theory results can now be applied to the controlled neural network learning process.

2.2.2 Recurrent Neural Network

RNN and MLP are different in that the information can flow in the both directions of feedforward and feedback in RNN and information can be propagated in only one direction in MLP. There are apparently many architectures of RNN by using different combinations of feeding back the states to the neurons in each layer. The typical recurrent neural networks include Jordan, Elman and their modifications.

Fig. 2.10 shows the original Elamn and Jordan recurrent neural network. One of the modifications of Elman and Jordan networks are such as Pham and Liu, where self-connections are introduced in the context units.

Let the external inputs to the Elamn net are represented by $u(k)$ and the network outputs by $y(k)$. The activation of the hidden units are $x(k)$. The outputs of the context units are represented by $x'(k)$. Now the output of the Elman net can be calculated in a matrix form as follows.

$$y(k) = \omega_3 x(k) \tag{2.62}$$

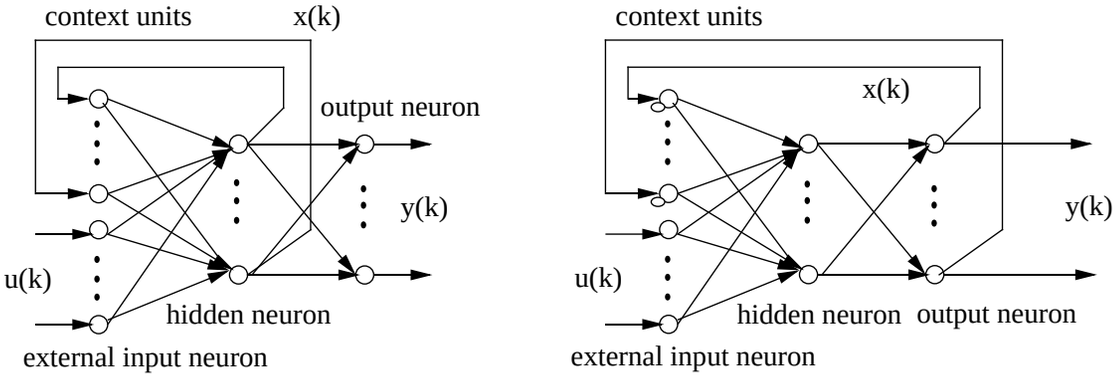


Figure 2.10: Original Elman net (left) and Jordan net (right).

$$x(k) = \omega_2 x'(k) + \omega_1 u(k) \tag{2.63}$$

$$x'(k) = x(k-1) \tag{2.64}$$

where ω_1 is the weight vector of external input to hidden units, ω_2 is the recurrent weight vector, and ω_3 is the hidden-to-output weight vector.

The recurrent neural networks can be easily trained by backpropagation through time or global learning algorithm, such as genetic algorithm.

2.2.3 Fuzzy Neural Network

A flowchart of the fuzzy neural network is shown in Fig. 2.11, which is organized into n input variables, m-term nodes for each input variable, 1 output node, and $m \times n$ rule nodes.

Let u_i^k denotes the ith input of a node in the kth layer, O_i^k denotes the ith node output in layer k. The signal propagation process can be described as follows.

- *Layer 1*: The nodes in this layer only transmit input values to next layer directly.

$$O_i^1 = u_i^1 = x_i. \tag{2.65}$$

- *Layer 2*: Assume that the used membership function is Gaussian function. Then we have

$$O_{ij}^2 = exp(-\frac{(O_i^1 - m_{ij})^2}{(\sigma_{ij})^2}) \tag{2.66}$$

where m_{ij} and σ_{ij} are the center and the width of the Gaussian membership function.

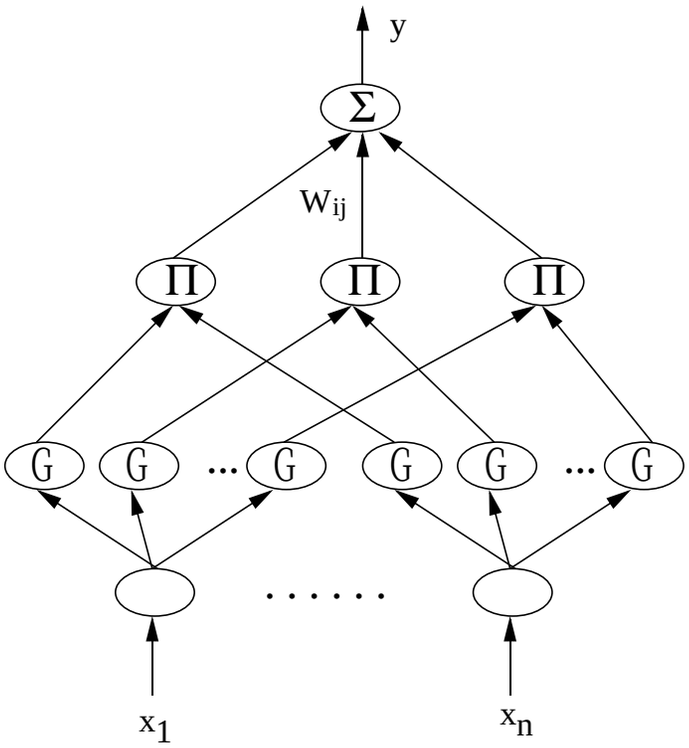


Figure 2.11: A fuzzy neurak network

- *Layer 3*: The calculation of the "firing strength" of the corresponding rule.

$$O_i^3 = \prod_i u_i^3 \tag{2.67}$$

- *Layer 4*: *Output Layer*

$$y_j = O_j^4 = \sum_i^m u_{ij}^4 \omega_i^4 \tag{2.68}$$

Finally, the overall representation of input x and the mth output y is

$$y_m(k) = \sum_{j=1}^{m} \omega_{mj} \prod_{i=1}^{n} exp(-\frac{(x_i(k) - m_{ij})^2}{(\sigma_{ij})^2}). \tag{2.69}$$

A gradient descent method can be used for tuning the parameters in FNN as follows. Let the objective function is given by

$$E(k) = \frac{1}{2}(y(k) - \hat{y}(k))^2 \tag{2.70}$$

The learning rules can be easily derived as follows.

$$\omega_{ij}(k) = \omega_{ij}(k-1) - \eta^{\omega}\frac{\partial E(k)}{\partial \omega_{ij}} \tag{2.71}$$

$$m_{ij}(k) = m_{ij}(k-1) - \eta^{m}\frac{\partial E(k)}{\partial m_{ij}} \tag{2.72}$$

$$\sigma_{ij}(k) = \sigma_{ij}(k-1) - \eta^{\sigma}\frac{\partial E(k)}{\partial \sigma_{ij}} \tag{2.73}$$

where

$$\frac{\partial E(k)}{\partial \omega_{ij}} = -e(k)\cdot O_i^3 \tag{2.74}$$

$$\frac{\partial E(k)}{\partial m_{ij}} = -\sum_k e(k)\omega_{ik}\cdot O_k^3 \cdot \frac{2(x_i - m_{ij})}{(\sigma_{ij})^2} \tag{2.75}$$

$$\frac{\partial E(k)}{\partial \sigma_{ij}} = -\sum_k e(k)\omega_{ik}\cdot O_k^3 \cdot \frac{2(x_i - m_{ij})^2}{(\sigma_{ij})^3} \tag{2.76}$$

2.3 Fuzzy Logic System

Fuzzy sets were introduced in 1965 by Lotfi Zadeth with a view to reconcile mathematical modeling and human knowledge in the engineering sciences. Since then, a considerable body of literature has blossomed around the concept of fuzzy sets in an incredibly wide range of areas, from mathematics and logics to traditional and advanced engineering methodologies.

2.3.1 The Definition of Fuzzy Sets

Fuzzy sets were introduced by Lotfi Zadeth in 1965. To introduce them consider the $X = \{x_1, x_2, x_3, x_4, x_5\}$ crisp set that will be called universe, or universal set and let $Y \subset x = \{x_1, x_2, x_3\}$ is its crisp subset.

By using the characteristic function defined as

$$\mu_Y(x) = \begin{cases} 1, & if\ x \in Y \\ 0, & otherwise \end{cases} \tag{2.77}$$

The subset Y can be uniquely represented by ordered pairs

$$Y = \{(x_1, 1), (x_2, 1), (x_3, 0), (x_4, 0), (x_5, 1)\} \tag{2.78}$$

In his original paper Zadeth proposed that the second member of an ordered pair (which is called the membership grade of the appropriate element) can take its value not only from the set $\{0, 1\}$ but from the closed interval $[0, 1]$ as well. By using this idea fuzzy sets are defined as follows.

Definition. Let X a universal crisp set. The set of ordered pairs

$$Y = \{(x, \mu_Y(x))|x \in X, \mu_Y : X \to [0, 1]\} \tag{2.79}$$

is said to be the fuzzy subset of X. The $\mu_Y : X \to [0, 1]$ function is called as membership function and its vlaue is said to be the membership grade of x.

2.3.2 Takagi-Sugeno Fuzzy Model

A fuzzy model proposed by Takagi and Sugeno [25] is described by fuzzy if-then rules whose consequent parts are represented by linear equations. This fuzzy model is of the following form:

$$R_i : If\ x_1\ is\ A_{i1}\ \ldots, x_n\ is\ A_{in}\ then\ y_i = c_{i0} + c_{i1}x_1 + \ldots + c_{in}x_n \tag{2.80}$$

where $i = 1, 2, \ldots, N$, N is the number of if-then rules, $c_{ik}(k = 0, 1, \ldots, n)$ are the consequent parameters, y_i is the output from the ith if-then rule, and A_{ik} is a fuzzy set.

Given an input $(x_1, x_2, \ldots, x_n)$, the final output of the fuzzy model is referred as follows:

$$y = \frac{\sum_{i=1}^{N} \omega_i y_i}{\sum_{i=1}^{N} \omega_i} = \frac{\sum_{i=1}^{N} \omega_i (c_{i0} + c_{i1}x_1 + \ldots + c_{in}x_n)}{\sum_{i=1}^{N} \omega_i} = \frac{\sum_{k=0}^{n} \sum_{i=1}^{N} \omega_i c_{ik} x_k}{\sum_{i=1}^{N} \omega_i} \tag{2.81}$$

where $x_0 = 1$, ω_i is the weight of the ith IF-THEN rule for the input and is calculated as

$$\omega_i = \prod_{k=1}^{n} A_{ik}(x_k), \tag{2.82}$$

where $A_{ik}(x_k)$ is the grad of membership of x_k in A_{ik}.

2.3.3 Universal Approximation Property

To Takagi-Sugeno approach, the universal approximation property was proved in [95][96]. In addition, a natural further generalization of this approach was proposed in [97][98], in which in the conclusion of each rule, the desired output y is given not by an explicit formula, but by a (crisp) dynamical systems, i.e., by a system of differential equations

that determine the time derivative of the output variable (i.e., its change in time) as a function of the inputs and of the previous values of output. This generalization also has universal approximation property.

A simplified Takagi-Sugeno fuzzy model proposed by Ying Hao [99] has the following rule base.

$$R_i : If\ x_1\ is\ A_{i1}\ \ldots, x_n\ is\ A_{in}\ then\ y_i = k_i(c_0 + c_1x_1 + \ldots + c_nx_n) \tag{2.83}$$

where $i = 1, 2, \ldots, N$, N is the number of if-then rules. From this it can be seen that the free parameters in the consequent part of the IF-THEN rules are reduced significantly. The universal approximation property of this simplified T-S fuzzy model has also been proved, and successfully applied to the identification and control of nonlinear systems [100].

2.3.4 Design Problems

Fuzzy logic systems [25][101] have been successfully applied to a vast number of scientific and engineering problems in recent years. The advantage of solving the complex nonlinear problems by utilizing fuzzy logic methodologies is that the experience or expert's knowledge described as a fuzzy rule base can be directly embedded into the systems for dealing with the problems. A number of improvements have been made in the aspects of enhancing the systematic design method of fuzzy logic systems [102]-[107]. In these researches, the needs for effectively tuning the parameters and structure of fuzzy logic systems are increased. Many researches focus on the automatically finding the proper structure and parameters of fuzzy logic systems by using genetic algorithms [103][106][107], evolutionary programming [105], tabu search [108], and so on. But there still remains some problems to be solved, for example, how to automatically partition the input space for each input-output variables, how many fuzzy rules are really needed for properly approximating the unknown nonlinear systems, and how to determine it automatically. As is well known, the curse-of-dimensionality is an unsolved problem in the fields of fuzzy and/or neurofuzzy systems.

Chapter 3

Hybrid Soft Computing for Identification

The general task of system identification problem is to approximate automatically the behavior of an unknown plant using an appropriate model. The identification techniques of nonlinear systems can be divided into two categories: parametric and nonparametric. The former assume that the functional form of the system model is known (usually based on physical modeling principles), the aim of identification is the estimation of the unknown parameters of the model. In the latter case, both of the functional form and the parameters of the system model is unknown. Therefore, the designer must specify the structure and parameters of the system to be identified by an appropriate method.

The nonparametric system identification has been studied under a variety of titles including neural networks, fuzzy systems and evolutionary computation approaches as discussed in Section 1.2. The basic idea of these methods is that the nonlinear system identification problem can be posed as a nonlinear function approximation problem. Therefore, the performance of the identification depends largely on the characteristics of the approximators. The usually used approximators include the multilayer perceptron networks, RBF networks, CMAC networks, T-S fuzzy systems, B-spline networks, neurofuzzy networks, wavelet networks and so on.

The researches in the areas of neural and fuzzy for function approximation shown that a number of nonlinear systems can be represented by a weighted sum of a number of Basis Functions (or influence functions) [113] [114]. This is also the modification and generalization of the Gabor-Kolmogrov approximation theory, which can be represented as[115]

$$f(x) = f_0 + \sum_{i=1}^{n} f_i(x_i) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} f_{ij}(x_i, x_j) + \ldots + f_{1,2,\ldots,n}(x_1, x_2, \ldots, x_n) \qquad (3.1)$$

where $f_j(.)$, etc. represent additive univariate, bivariate, ... components of $f(.)$. In neurofuzzy approaches, each component is represented by a separate neurofuzzy system with reduced dimension [116]. In the case of basis function networks, each component can be represented/approximated by a basis functions.

But there are still some problems to be solved, for example, how many basis functions are proper for approximating an unknown nonlinear system, which type of basis functions is good (globally or locally), how to select these basis functions including its structure and parameters, how to automatically partition the input space into the minimal set of local models which offer maximal approximation capability. As it is well known that the curse-of-dimensionality is an un-solved problem in the fields of B-spline networks, wavelet neural networks, fuzzy systems and neurofuzzy networks.

In this chapter, a unified framework for automatically evolving the hybrid soft computing models is proposed firstly. The basic ideas behind this technique are that the architecture of the various hybrid soft computing models can be created automatically by using a modified PIPE algorithm with the tree structural representation, and the parameters used in hybrid soft computing models can be simultaneously optimized by parameter optimization techniques. This technique has been implemented to optimize four kinds of hybrid soft computing models. They are:

- Additive model and Reconstruction of polynomials.
- Evolving non-regular MLP nets.
- Evolving the basis function networks
- Evolutionary design of hierarchical T-S fuzzy models

3.1 A Unified Framework of Hybrid Soft Computing Models

PIPE as discussed in Section 2.1.4 can be used for the automated program synthesis. This means that given a data set a symbolic express can be found by PIPE for accurately fitting a data set. But, same as genetic programming, the symbolic express evolved by PIPE is usually redundant, long, and no meaningful sense to some extend. This is because the used function operator has some limitations. Motivated by this disadvantage, a new method is proposed in this chapter, in which a valuable symbolic express, such as, an approximation polynomial, a neural network, a fuzzy T-S model, a basis function neural

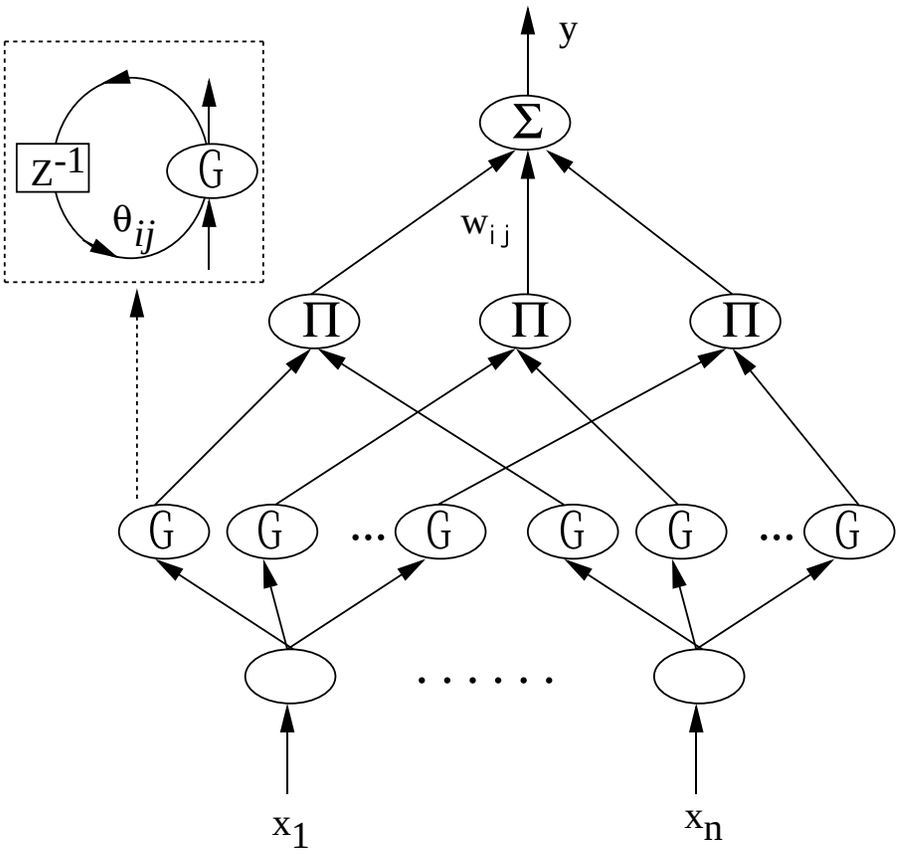


Figure 3.1: A recurrent fuzzy neural network (RFNN)

network and a hierarchical T-S fuzzy model, can be formulated by using the modified PIPE algorithm with specified instruction sets..

3.1.1 Representation and Calculation with Tree: An Example

If a soft computing model, i.e., recurrent fuzzy neural network (RFNN), can be represented as a tree and can be calculated same as the usually used way, some advantages may be appeared. First the different architecture of the RFNN can be created via creating the different tree. And then some tree-based evolutionary algorithms can be used to evolve the architectures of the RFNN, e.g., a modified PIPE algorithm.

RFNN and Tree

The architecture of a RFNN is shown in Fig.3.1 [178], which is the extension of the usually used fuzzy neural network (FNN). In RFNN, the feedback connections are added in the second layer of the FNN. This results in more flexibly adjustable parameters for approximating a nonlinear map by using RFNN than FNN.

The input/output representation of RFNN can be described as follows

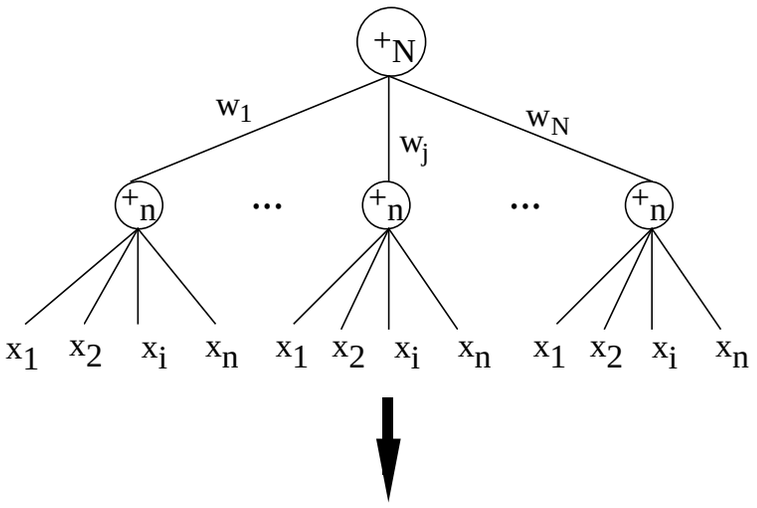
$$y(k) = \sum_{j=1}^{N} \omega_j \prod_{i=1}^{n} exp(-\frac{(x_i(k) + O_{ij}^2(k-1) \cdot \theta_{ij} - m_{ij})^2}{(\sigma_{ij})^2}) \tag{3.2}$$

$$O_{ij}^2(k-1) = exp(-\frac{(x(k-1) + O_{ij}^2(k-2) \cdot \theta_{ij} - m_{ij})^2}{(\sigma_{ij})^2}) \tag{3.3}$$

where n is the number of input variables, N denotes the rule number. The used membership function is the Gaussian function,

$$\mu_{ij} = exp\left(-\frac{(x_{ij} - m_{ij})^2}{(\sigma_{ij})^2}\right) \tag{3.4}$$

where m_{ij} and σ_{ij} are the center and the width of Gaussian membership function. The subscript ij indicates the j-th term of the i-th input x_i.



(1) Create the center, width and recurrent weight for each input vraiable and attach it to the each neuron in middle layer of the tree.

(2) Create the parameters in the conclusion part of the fuzzy rules, and attach it to the root node.

(3) Each node in the middle of the above tree represents a fuzzy rule.

Figure 3.2: Tree structural representation of a RFNN

The RFNN can be represented as a tree, which is shown in Fig. 3.2 with some necessary data structure. The used instruction sets for generating the tree are $I_0 = \{+_2, \ldots, +_N\}$, $I_1 = \{+_n\}$ and $I_2 = \{x_1, x_2, \ldots, x_n\}$. N is used to control the number of hidden neurons which is also the number of rules in the RFNN, and n is the number of inputs in the RFNN.

As mentioned above, the tree (RFNN model) can be generated in a recursive way (deepest first) as follows:

(1) Select a root node from the instruction set I_0 according to the probability of selecting instructions. If the number of arguments of the selected instruction is larger than one, e.g., $+_i$, then create i weights and attach it to the root node.

(2) Create the left sub-node of the root node from the instruction set I_1. If the number of arguments of the selected instruction is larger than one, then create a number of parameters (include the center, width and recurrent weight) and attach them to the node, and then create its left sub-node as same way from the instruction set I_2. Otherwise, if the instruction of the node is an input variable, return to upper layer of the tree in order to generate the right part of the tree.

(3) Repeat this process until a full tree is generated.

Remark: There are two key points in the generation of the tree. The one is that the instruction is selected according to the initial probability of selecting instructions, and the probability will be changed with generation by a modified probabilistic incremental algorithm. The other is that if the selected node is a non-terminal node, then generate the corresponding data structure (parameters used in RFNN model) and attach it to the node.

Calculate the output of RFNN by tree

In order to calculate the output of the RFNN presented as a tree, we need first calculate the output of each node in the middle layer of the tree as follows

$$y_j(k) = \prod_{i=1}^{n} exp(-\frac{(x_i(k) + O_{ij}^2(k-1) \cdot \theta_{ij} - m_{ij})^2}{(\sigma_{ij})^2}) \tag{3.5}$$

It is can be implemented by a recursive way. And then calculate the output of the tree at root node as follows

$$y(k) = \sum_{j=1}^{N} \omega_j y_j(k). \tag{3.6}$$

3.1.2 Structure Optimization by MPIPE

PIPE and MPIPE

In this research, in order to create and optimize the specific type constrained sparse trees which represent some of hybrid soft computing models, the standard PIPE are modified as follows:

- The instruction set is modified from one instruction set used in PIPE to two or three instruction sets in MPIPE. In addition, in order to formulate a meaningful hybrid soft computing model some instructions are removed from the usually used instruction set of PIPE, i.e., sin, cos, exp, log, and new instructions, i.e., $+_n$ are added to the instruction sets of the MPIPE.

- The initial probability of selecting instructions in each of the instruction sets is modified as follows

$$P\left(I_{d,w}\right) = \frac{1}{l_i}, \forall I_{d,w} \in I_i, i = 0, 1, 2 \tag{3.7}$$

  where $I_{d,w}$ denotes the instruction of the node with depth d and width w, $l_i (i = 0, 1, 2)$ is the number of instructions in the instruction set I_i.

- To our knowledge, PIPE cannot be used for the parameter optimization in general. And finding an optimal hybrid oft computing model needs searches in the architecture and parameter space simultaneously. To cope with the specified problem in designing of the hybrid soft computing model, the specified data structure (parameters) are added to the node of the tree in MPIPE.

- The mutation probability of selecting instructions in PIPE is modified as shown in Eq.(3.11).

Although some modifications are made in MPIPE in order to form a hybrid soft computing model, the key learning principle of PIPE are maintained. MPIPE with the population based probabilistic incremental learning and elitist learning are used for selecting the proper instructions of the tree in order for evolving an optimal architecture of the hybrid soft computing model finally.

The learning procedure of the MPIPE

The main learning procedure of MPIPE can be summarized as follows:

1) Initially a population of the tree (include the parameters attached to the nodes) is randomly generated according to the predefined the probability of selecting instructions.

2) Let P_{ROG_b} and $P_{ROG_{el}}$ be the best program of the current generation (best program) and the one found so far (elitist program), respectively. Define the probability and the target probability of best program as

$$P\left(P_{ROG_b}\right) = \prod_{\substack{I_{d,w}:used\ to \\ generate\ P_{ROG_b}}} P\left(I_{d,w}\right) \tag{3.8}$$

and

$$P_{TARGET} = P(P_{ROG_b}) + (1 - P(P_{ROG_b}))\frac{\varepsilon + FIT(P_{ROGel})}{\varepsilon + FIT(P_{ROG_b})} \tag{3.9}$$

where $FIT\left(P_{ROG_b}\right)$ and $FIT\left(P_{ROGel}\right)$ denote the fitness of the best and elitist program. In order to increase the probability $P(P_{ROG_b})$, repeat the following process until $P\left(P_{ROG_b}\right) \geq P_{TARGET}$:

$$P\left(I_{d,w}\right) = P\left(I_{d,w}\right) + c^{lr} \cdot lr \cdot \left(1 - P\left(I_{d,w}\right)\right) \tag{3.10}$$

where c^{lr} is a constant influencing the number of iterations and ε is the fitness constant.

3) Define the mutation probability as

$$P_{M_p} = \frac{P_M}{(l_0 + l_1 + l_2) \cdot \sqrt{|P_{ROG_b}|}} \tag{3.11}$$

where $|P_{ROG_b}|$ denotes the number of nodes in program. All the probabilities $P\left(I_{d,w}\right)$ are mutated with probability P_{M_P} according to

$$P\left(I_{d,w}\right) = P\left(I_{d,w}\right) + mr \cdot \left(1 - P\left(I_{d,w}\right)\right) \tag{3.12}$$

4) Repeat this process according to the new probability of selecting instructions in the instruction sets until the best structure found or the maximum number of MPIPE algorithm is reached.

3.1.3 Parameter optimization

The parameters used in hybrid soft computing model such as weights, centers and widths formulate the parameter search space. Weight is linear-in-parameter, and center and width are the nonlinear-in-parameter.

Parameter Tuning Strategy:

Some of parameter optimization strategies are those described as follows.

- *Non-hybrid parameter optimization strategy.* In this approach, the parameters can be optimized by unified learning algorithm, e.g., gradient descent method, evolutionary programming, random search algorithm and so on.

- *Hybrid parameter optimization strategy.* In this strategy, the suggestion is that the weight (linear-in-parameter) can be optimized by a fast parameter learning algorithm, e.g., least-squares, Lyapunov-based algorithms, robust learning algorithms, and the center and width (nonlinear-in-parameter) are optimized by a global optimization technique, e. g., GA, EP and random search algorithms.

Some Candidate Parameter Tuning Methods:

- *Gradient algorithm* is one of the most straightforward and widely used approach for parameter estimation [113]. The main idea behind the gradient method is to update at each time t the parameter estimate $\hat{\theta}(t)$ in the direction where the cost function $J(\hat{\theta})$ decreases the most. Assume that the approximation function is linearly parameterized as follows

$$y(t) = \theta(t)^T \zeta(t) + \sigma(t) \tag{3.13}$$

Then, the deterministic, continuous-time version of the gradient learning algorithm can be described as

$$\dot{\hat{\theta}}(t) = -\nabla J(\hat{\theta}(t)). \tag{3.14}$$

If we minimize the cost function associate with the instantaneous error,

$$J(\hat{\theta}) = \frac{1}{2}(y(t) - \hat{\theta}^T(t)\zeta(t))^T(y(t) - \hat{\theta}^T(t)\zeta(t)) \tag{3.15}$$

then the following gradient estimation algorithm can be obtained

$$\dot{\hat{\theta}}(t) = \Gamma\zeta(t)(y(t) - \hat{\theta}^T(t)\zeta(t)) \tag{3.16}$$

where γ is a positive-definite symmetric matrix representing the learning rate matrix and the initial condition is given by $\hat{\theta}(0) = \hat{\theta}_0$. In the special case where the same learning rate γ is used for each parameter estimate, then $\Gamma = \gamma I$, where I is the identity matrix.

The normalized gradient algorithm, which is sometimes used to improve the stability and convergence properties of the algorithm is described by

$$\dot{\hat{\theta}}(t) = \frac{\Gamma\zeta(t)(y(t) - \hat{\theta}^T(t)\zeta(t))}{1 + \beta\zeta^T(t)\zeta(t)} \tag{3.17}$$

where $\beta > 0$ is a design constant.

- *Least-squares algorithm* is to fit a mathematical model to a sequence of observed data by minimizing the sum of the squares of the difference between the observed data and computed data. Assume that the cost function is given by

$$J(\hat{\theta}) = \int_0^t (y(\tau) - \hat{\theta}^T(t)\zeta(\tau))^T (y(\tau) - \hat{\theta}^T(t)\zeta(\tau)) \tag{3.18}$$

where $y(\tau)$ is the measured data at time τ, and $\zeta(\tau)$ is the regressor vector at time τ. The above cost function penalizes all the past errors $y(\tau) - \zeta^T(\tau)\hat{\theta}(t)$ for $\tau = 0$ up to $\tau = t$, relative to the current parameter estimate $\hat{\theta}(t)$. By setting to zero the gradient of the cost function ($\nabla J(\hat{\theta}) = 0$), we obtain the least-squares estimate for $\hat{\theta}(t)$:

$$\hat{\theta}(t) = \left[\int_0^t \zeta(\tau)\zeta^T(\tau)d\tau\right]^{-1} \int_0^t \zeta(\tau)y^T(\tau)d\tau. \tag{3.19}$$

This is the batch version of least-squares algorithms. The recursive version of the least-squares algorithm is given by

$$\dot{\hat{\theta}}(t) = P(t)\zeta(t)\left(y^T(t) - \zeta^T(t)\hat{\theta}(t)\right), \hat{\theta}(0) = \hat{\theta}_0 \tag{3.20}$$

$$\dot{P(t)} = -P(t)\zeta(t)\zeta^T(t)P(t), P(0) = P_0, \tag{3.21}$$

where $P(t)$ is a square matrix of the same dimension as the parameter estimate $\hat{\theta}$. The initial condition of P_0 of the P is chosen to be positive-definite.

- *Lyapunov-based algorithm.* In this scheme, the problem of designing an adaptive law is formulated as a stability problem in which the differential equation of the adaptive law is chosen so that certain stability properties can be established using Lyapunov theory.

For the parameter estimation problem of Eq.(3.13), the following adaptive law can

be generated by using Lyapunov method:

$$\dot{\hat{\theta}}(t) = \Gamma\zeta(t)(y(t) - \hat{\theta}^T(t)\zeta(t)) \tag{3.22}$$

which is essentially of the same form as the gradient algorithm and the stability and convergence properties of the two algorithms are also similar.

- *Robust learning algorithm.* Some modifications to the standard parameter learning algorithm in order to provide stability and improve performance in the presence of modeling errors have been proposed. These modifications lead to what is known as *robust learning algorithms*. The term robust is used to indicate that the learning algorithm is such that in the presence of modeling errors it retains its stability properties.

  To illustrate the various options for enhancing the robustness of the adaptive laws, we consider a generic adaptive law

  $$\dot{\hat{\theta}}(t) = \Gamma\xi(t)\varepsilon(t), \tag{3.23}$$

  where Γ is the learning rate matrix, $\xi(t)$ is the regression vector, $\varepsilon(t)$ is the estimation error.

  Some modifications in order to prevent the parameter drift can be summarized as follows.

  - **Projection modification:** In this approach, an effective way to prevent parameter drift is to restrain the parameter estimates within a predefined bounded and convex region P. The projection modification implements this idea as follows: If the parameter estimate $\hat{\theta}$ is inside the desired region P, or is on the boundary and directed inside the region P, then the standard adaptive law (3.23) is implemented. In the case that $\hat{\theta}$ is on the boundary of P and its derivative is directly outside the region, then it is projected onto the tangent hyperplane.
  - **σ-modification:** In this approach, the adaptive law (3.23) is modified to

    $$\dot{\hat{\theta}}(t) = \Gamma\xi(t)\varepsilon(t) - \Gamma\sigma\hat{\theta}(t) \tag{3.24}$$

    where σ is a small positive constant. The additional term $\Gamma\sigma\hat{\theta}$ acts as a stabilizing component for adaptive law. If the parameter estimate $\hat{\theta}(t)$ starts drifting

to large positive values, then $\Gamma\sigma\hat{\theta}$ becomes large and negative, thus forcing the parameter estimate to decrease.

– **ϵ-modification:** The ϵ-modification was motivated as an attempt to eliminate some of the drawbacks associated with the *sigma*-modification. It is given by

$$\dot{\hat{\theta}}(t) = \Gamma\xi(t)\varepsilon(t) - \Gamma|\epsilon|\nu\hat{\theta}(t) \tag{3.25}$$

where $\nu > 0$ is a design constant. The idea behind this approach is to retain the convergence properties of the adaptive scheme by forcing the additional term $\Gamma|\epsilon|\nu\hat{\theta}$ to be zero in the case that $\epsilon(t) = 0$. In the case that the parameter estimate vector $\hat{\theta}(t)$ starts drifting to large values, then the ϵ-modification again acts as a stabilizing force.

– **Dead-zone modification:** In the presence of approximation errors, the adaptive law (3.23) tries to minimize the estimation error ϵ, sometimes at the expense of increasing the magnitude of the parameter estimates. The idea behind the dead-zone modification is to enhance robustness by tuning off adaptation when the estimation error becomes relatively small compared to the approximation error. The dead-zone modification is given by

$$\dot{\hat{\theta}}(t) = \begin{cases} \Gamma\xi(t)\epsilon(t), & if\ |\epsilon| \geq \delta_0 \\ 0, & if\ |\epsilon| < \delta_0 \end{cases} \tag{3.26}$$

where δ_0 is a positive design constant that depends on the approximation error. One of the drawbacks of the dead-zone modification is that the designer needs an upper bound on the approximation error, which is usually not available.

3.1.4 The Proposed Algorithm for Designing of Hybrid Soft Computing Models

Combining the self-organizing and learning characteristics in the aspects of structure optimization of MPIPE and the some parameter optimization strategies, a hybrid learning algorithm (see Fig. 3.3) for designing of hybrid soft computing models is proposed as follows.

1) Set the initial values of parameters used in the MPIPE and parameter learning algorithms. Set the elitist program as NULL and its fitness value as a biggest positive real number of the computer at hand. Create the initial population (tree) and corresponding parameters used in hybrid soft computing model.

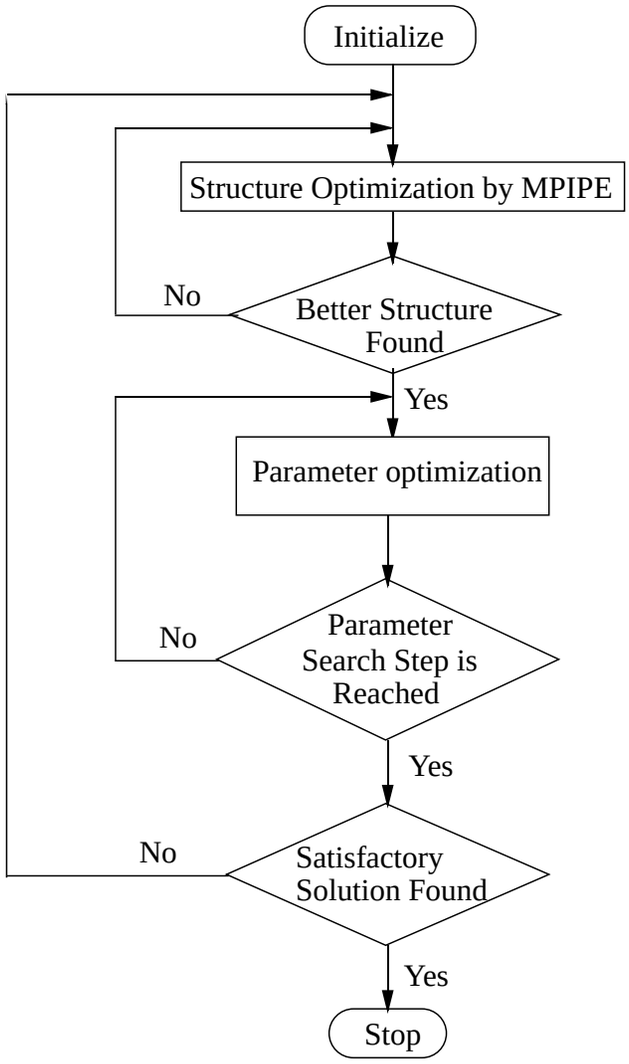


Figure 3.3: The flow chart of the proposed algorithm for designing of hybrid soft computing models

2) Recall the MPIPE sub-program for structure optimization, in which the fitness function can be calculated by the mean square error (MSE) or the sum of absolute error (SAE)

$$Fit(i) = \frac{1}{P}\sum_{j=1}^{P}(y_1^j - y_2^j)^2 \tag{3.27}$$

$$Fit(i) = \sum_{j=1}^{P}|y_1^j - y_2^j|^2 \tag{3.28}$$

where P is the total number of samples, y_1^i and y_2^i are the actual and model output of i-th sample $Fit(i)$ denotes the fitness value of i-th individual.

3) If the better structure found, then go to step 4), otherwise go to step 2). The criterion concerning with better structure found is distinguished as follows: if the fitness value of the best program is smaller than the fitness value of the elitist program, or the fitness values of two programs are equal but the nodes of the former is lower than the later, then we say that the better structure is found. Where the best and elitist programs are the best program at current generation and the one found so far, respectively.

4) Recall the parameter optimization strategy sub-program for parameter optimization. In this step, the tree structure or architecture of the hybrid model is fixed, and it is the best tree taken from the end of run of MPIPE search. All of the parameters used in the best tree or the corresponding hybrid soft computing model formulate a vector which to be optimized by parameter optimization strategy in order to decrease the fitness value of best program.

5) If the maximum number of parameter search is reached, then go to step 6); otherwise go to step 4).

6) If satisfied solution is found, then stop; otherwise go to step 2).

3.2 Additive Model and Reconstruction of Polynomials

3.2.1 Additive Model and Tree

In function approximation, an arbitrary nonlinear function can be approximated by many techniques such as polynomial regression, artificial neural networks, fuzzy systems, wavelet analysis and so on. One common feature of these methods is that the nonlinear function to be approximated can be usually represented as a linear combination of nonlinear terms (base functions). The approximation accuracy depends largely on the selection of the base functions and the weights of the linear combination terms. In fact, the problem of base function selection corresponding to the model structure selection problem, and the determination of the weight same as the parameter identification problem.

Based on this idea, the candidate solution (symbolic expression) can be represented as a partially weighted sparse tree (Fig.3.4), in which the root node returns the weighted sum of a number of linear/nonlinear terms (base function or linear/nonlinear components).

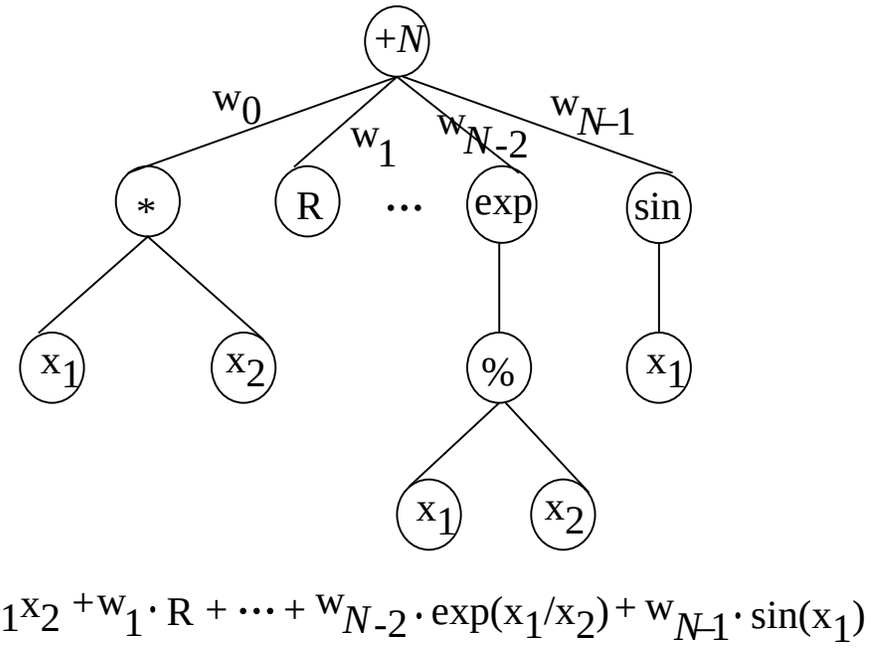


Figure 3.4: Interpretation of a sparse tree and its symbolic expression

In order to create above specific tree, two instruction sets I_0 and I_1 are used in the modified PIPE. The instruction set I_0 is used for creating the instruction of the root node and the instructions of the remained node are selected from the instruction set I_1 by MPIPE.

$$I_0 = \{+_2, +_3, \ldots, +_N\} \tag{3.29}$$

$$I_1 = F \cup T = \{*, \%, sin, cos, exp, rlog, x, R\} \tag{3.30}$$

where $F = \{*, \%, sin, cos, exp, rlog\}$ and $T = \{x, R\}$ are function set and terminal set. $+_N$, $*$, $\%$, sin, con, exp, $rlog$, x, and R denote addition, multiplication, protected division ($\forall x, y \in R, y \neq 0 : x\%y = x/y$ and $x\%0 = 1$), sine, cosine, exponent, protected logarithm ($\forall x \in R, x \neq 0 : rlog\,(x) = log\,(abs\,(x))$ and $rlog\,(0) = 0$), system inputs, and random constant number, and taking N, 2, 2, 1, 1, 1, 1, 0 and -1 arguments respectively.

It is interesting to mention that if the instruction set I_1 only contains the random real number, input variables, and product operator, i.e., $I_1 = \{R, x_1, x_2, \ldots, x_n, *, *_3\}$, then we can evolve the polynomials for realizing the polynomial regression.

3.2.2 Implementation and Experiments

Some simulations have been made in order to verify the effectiveness of the proposed method for the identification of linear/nonlinear system. In this experiments, the architecture and parameters of the additive model are evolved by MPIPE and least squares, respectively. It can be seen that for the polynomial regression problem, an almost true model of the system can be obtained by using the proposed method.

Linear system identification

A benchmark ARMAX system often used to test various identification methods is given by

$$y(k) = b_0u(k-5) + b_1u(k-6) + a_0y(k-1) + a_1y(k-2)$$
$$+c_0e(k) + c_1e(k-1) + c_2e(k-2) \tag{3.31}$$

where $a_0 = 1.5$, $a_1 = -0.7$, $b_0 = 1.0$, $b_1 = 0.5$, $c_0 = 1.0$, $c_1 = -1.0$, $c_2 = 0.2$. The objective here is to optimally identify the structure and parameters of the system in the presence of the noise. The input $u(k)$ is randomly generated at [-5, 5]. The $e(k)$ is Gaussain-distributed random variable with mean 0 and deviation 1. 400 samples are generated by using the above input $u(k)$ and Eq.(3.17), in which 200 data used for training and the other 200 data used for validation.

The used instruction set $I_1 = \{+_2, +_3, +_4, +_5, +_6, +_7, +_8, +_9, +_{10}\}$ and $I_2 = \{*, u(k-5), u(k-6), y(k), y(k-1), y(k-2), e(k), e(k-1), e(k-2)\}$.

The used fitness function is the sum of absolute error between the actual and evolved output of the plant. The control parameters of the proposed method are shown in Table 3.1.

The following model are obtained at generation 123 with the fitness 0.669568:

$$y(k) = 1.000307u(k-5) + 0.505093u(k-6) + 1.499481y(k-1)$$
$$-0.699698y(k-2) + 0.998582e(k) - 1.001312e(k-1) + 0.198232e(k-2) \tag{3.32}$$

Fig. 3.5 presents the outputs of actual system and evolved model for the validation set, and the identification error is shown in Fig. 3.6.

Table 3.1: Parameters of the proposed Algorithm

Population Size PS	10
Initial Terminal Probability P_T	0.8
Elitist Learning Probability P_{el}	0.01
Learning Rate lr	0.01
Fitness Constant ε	0.000001
Overall Mutation Probability P_M	0.4
Mutation Rate mr	0.4
Prune Threshold T_P	0.999999
Random Constant Threshold T_R	0.3
Initial Weights	[-1, 1]

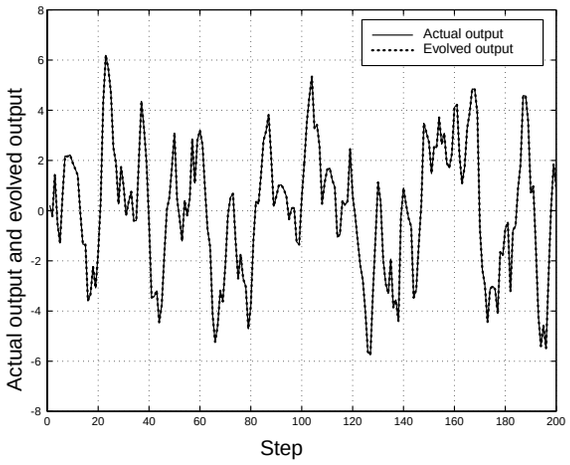


Figure 3.5: Actual and evolved output

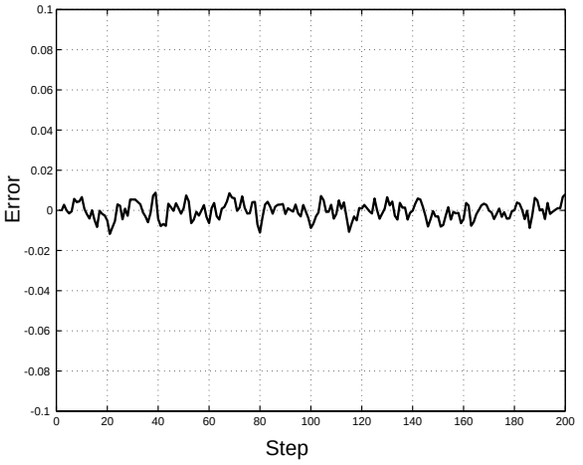


Figure 3.6: Error of the identification

Chaotic time series prediction

Given Henon map as follows

$$x(k+1) = \alpha - x(k)^2 + \beta x(k-1) \tag{3.33}$$

where $x(k) \in [-2.0, 2.0]$, $\alpha = 1.4$ and $\beta = 0.3$.

100 training data are generated at initial conditions of $x(0) = 1.2$ and $x(1) = 0.8$ by using Eq.(3.19). The structure and parameters of the system are identified by using proposed method with the instruction set $I_1 = \{+_2, +_3, +_4, +_5, +_6, +_7, +_8\}$ and $I_2 = \{*, x, R\}$. The parameters of PIPE and random search are same as those described in above.

The evolved Henon map as the best solution is obtained at generation 127 with fitness

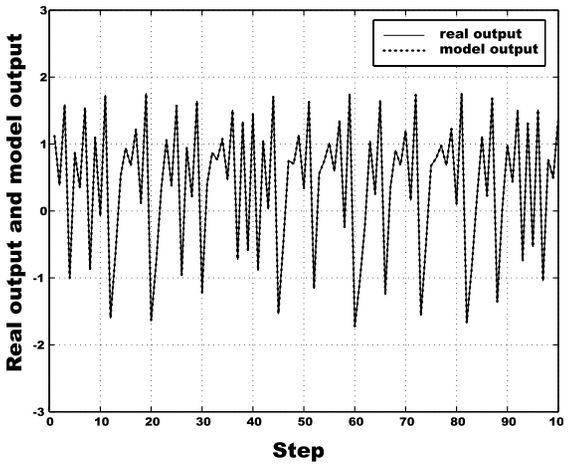


Figure 3.7: Actual and evolved output

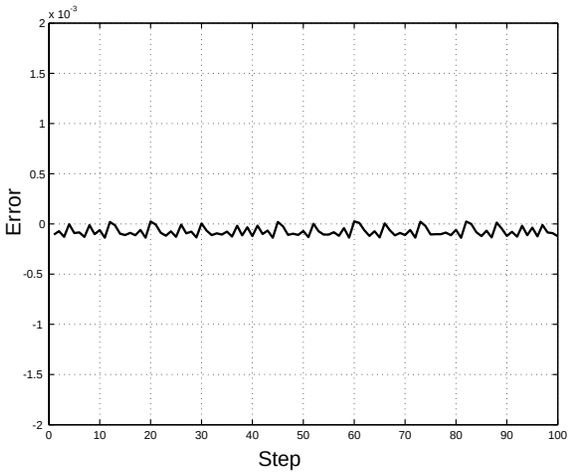


Figure 3.8: Error of the identification

0.007911:

$$x(k+1) = 1.400015 - 1.000050x(k)^2 + 0.300007x(k-1) \tag{3.34}$$

Fig.3.7 presents the outputs of actual system and evolved model, and the identification error is shown in Fig. 3.8. It is obviously that the generalization ability of evolved model is very well due to the evolved model is almost same with the original system model.

In addition, in order to learn about how to select the number of instructions in the instruction set I_0, we vary the instruction set I_0 as follows

case 1: $I_0 = \{+_2, +_3\}$

case 2: $I_0 = \{+_2, +_3, \cdots, +_5\}$

case 3: $I_0 = \{+_2, +_3, \cdots, +_{10}\}$

case 4: $I_0 = \{+_2, +_3, \cdots, +_{15}\}$

Four independent experiments were done. Simulation results show that a nonlinear system can be identified by proposed method with a proper selected instruction set I_0, in which the number of instructions in the instruction set I_0 (the number of nonlinear terms of a nonlinear system to be approximated) will affect the convergence speed of the hybrid method. The smaller the number of instructions is, the fast the convergence speed is. But it is valuable to note that the nonlinear system may not be identified while the number of instructions is too small. The bigger the number of instructions is, the slow the convergence speed is. In our experiments, with the increase of number of instructions, it need 132 generations to get a solution with fitness value 0.028245 at case 3, and 2217 generations with fitness value 0.016810 at case 4. The detailed procedure of structure and

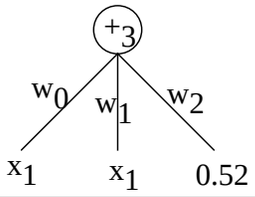
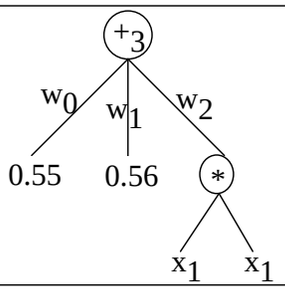
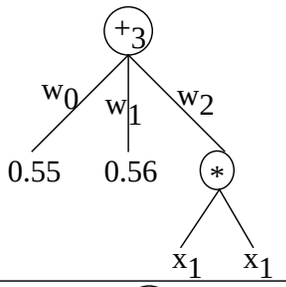
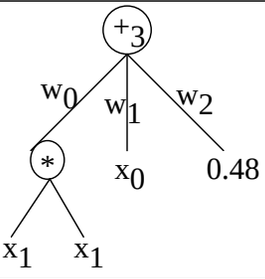
Generation	Evolution of Structure	Evolution of Parameters	Fitness Value
0		w_0=-0.336780 w_1=-0.009015 w_2=1.436127	75.447236
8		w_0=2.040441 w_1=1.009927 w_2=-1.075994	22.355437
41		w_0=-1.008165 w_1=-1.940611 w_2=-0.141510	17.591037
70		w_0=-1.000080 w_1=0.300171 w_2=2.917001	0.016910

Figure 3.9: The procedure of structure and parameter evolution at instruction set $I_0 = \{+_2, +_3\}$

parameters evolution of the proposed method at case 1 and case 2 are shown in Fig.3.9 and Fig.3.10, respectively.

Complex nonlinear system identification

The plant to be identified is given by

$$y = 0.2 + 0.3 * x_1 + 0.4 * x_2 + 0.5 * x_3 + 0.6 * x_1^2 + 0.7 * x_2^2 + 0.8 * x_3^2$$
$$+0.9 * x_1 * x_2 + 0.1 * x_1 * x_3 + 0.2 * x_2 * x_3; \quad (3.35)$$

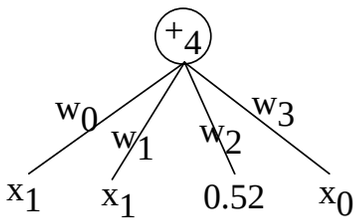
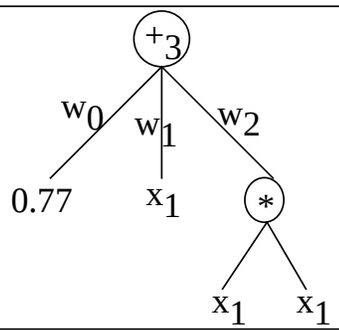
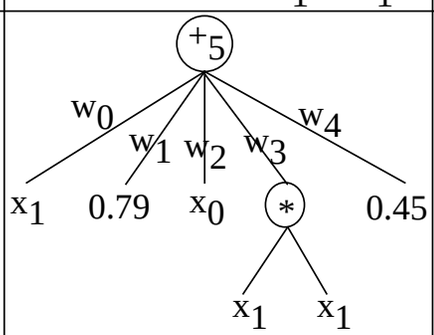
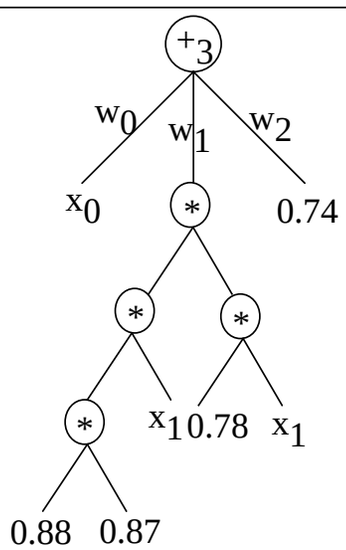
Genera -tion	Evolution of Structure	Evolution of Parameters	Fitness Value
0		w_0=-0.480395 w_1=0.088255 w_2=1.448029 w_3=0.176401	74.247236
4		w_0=2.218472 w_1=-0.140756 w_2=-1.008854	17.574925
11		w_0=-0.000560 w_1=0.960001 w_2=0.300700 w_3=-1.000202 w_4=1.420876	0.094891
70		w_0=0.299944 w_1=-1.664293 w_2=1.894790	0.013576

Figure 3.10: The procedure of structure and parameter evolution at instruction set $I_0 = \{+_2, +_3, +_4, +_5\}$

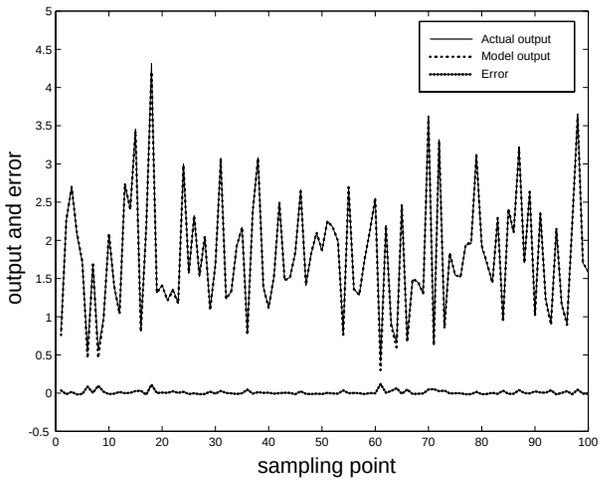


Figure 3.11: Actual and evolved outputs for training data

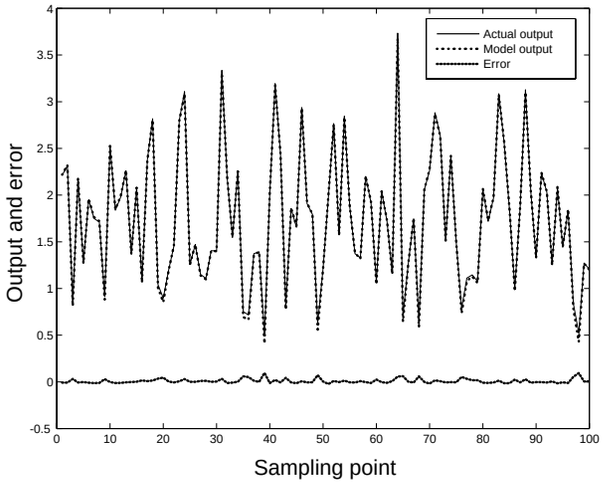


Figure 3.12: Actual and evolved outputs for test data

The objective here is to optimally identify the structure and parameters of the nonlinear system. 400 samples are randomly generated at interval [0,1]. The first 200 data points are used for the training and the remind data used for validation.

The used instruction set $I_1 = \{+_{10}, +_{11}, +_{12}\}$ and $I_2 = \{*, x_1, x_2, x_3\}$.

The used fitness function is the sum of absolute error between the actual and evolved output of the plant. The control parameters of the proposed method are shown in Table 3.1.

The identification results are shown in Fig. 3.11 and Fig. 3.12 for training data set and test data set, respectively.

From above simulation results, it can be seen that the proposed method works very well for generating of the system model, especially for the reconstructing the polynomials.

3.3 Evolving the Non-Regular MLP Neural Networks

3.3.1 MLP Net and Tree

The usually used representation/encoding methods for evolutionary training of ANNs are direct encoding scheme and indirect encoding scheme. The former uses a fixed structure (connection matrix or bitstrings) to specify the architecture of the corresponding ANN [109][110]. The latter uses rewrite rules (cellular encoding [111] and graph generation grammars [112]) to specify a set of construction rules that are recursively applied to yield the ANN.

In order to evolve the architecture and parameters of MLP network simultaneously, the candidate solution is represented as a type constrained sparse tree (Fig.3.13), and the corresponding MLP network is shown in Fig.3.14. The layer 0 and layer 1 of the tree denote the output and hidden layer of the corresponding MLP network. The other layers of the tree denote the input layer of the corresponding MLP network. It can be seen that the node $*$ in the tree is used for creating the higher order terms in the corresponding ANN. In addition, every node of layer 0 and layer 1 have a bias and a activation function which is randomly selected from a predefined activation function set.

The output of each node in the layer 0 and layer 1 is calculated same as the output of neurons in the hidden layer and output layer of MLP. For example, the output of node $+_2$ in the layer 1 of the tree is $f(w_6 * x_1 * x_2 + w_7 * x_3 - \theta)$, where f and θ are the activation function and bias of node $+_2$.

Three instruction sets I_0, I_1 and I_2 are used in this approach. The instructions of the node in the layer 0, and layer 1 of the tree are selected from instruction sets I_0 and I_1, respectively. The instructions of the node in other layer is randomly selected from the instruction set I_2.

$$I_0 = \{+_2, +_3, \ldots, +_N\} \tag{3.36}$$

$$I_1 = \{+_3, \ldots, +_M\} \tag{3.37}$$

$$I_2 = \{*, x_1, x_2, \ldots, x_n\} \tag{3.38}$$

where $+_N$, $*$ and x_n denote addition, multiplication and nth system inputs, and taking N, 2, 1 arguments respectively. It can be seen that the instruction $*$ determine the higher order input in the tree or corresponding MLP networks.

N is an integer number, and is used to determine the maximum number of hidden neurons in the MLP network. The instructions in the instruction set I_1 are used to select the numbers of inputs of the hidden neurons. Each node $N_{d,w}$ of the subtree contains an

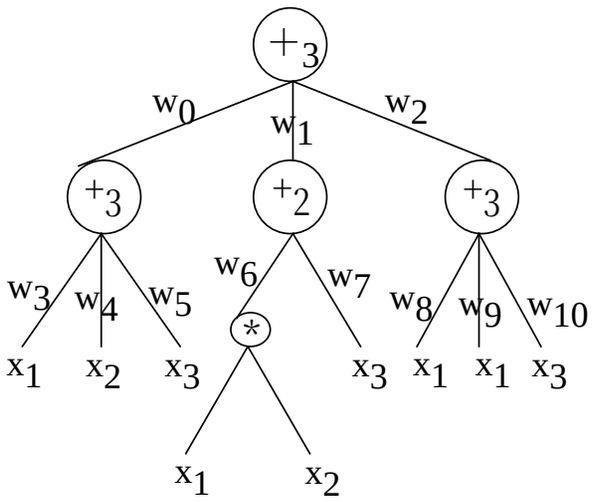


Figure 3.13: A type constrained sparse tree (genotype)

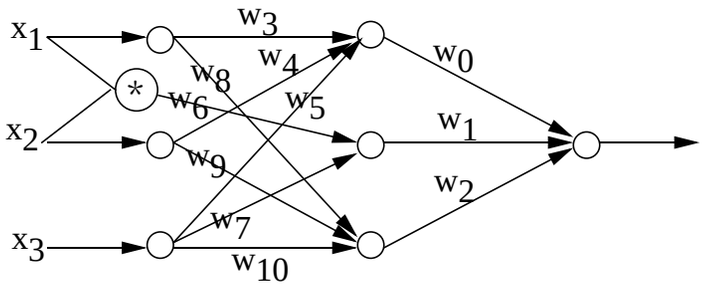


Figure 3.14: Corresponding MLP network of above type constrained sparse tree (phenotype)

instruction $I_{d,w}$, where d and w denote the node's depth and horizontal position of the tree. $P\left(I_{d,w}\right)$ denotes the probability of choosing instruction $I_{d,w}$ at node $N_{d,w}$.

In addition, the activation functions used in the layer 0 and layer 1 of the tree is randomly selected from the predefined activation function set $F = \{f_0(x), f_1(x), f_2(x), f_3(x)\}$. In our experiments, the used activation functions are

$$f_0(x) = \frac{1}{1 + exp(-x)} \tag{3.39}$$

$$f_1(x) = \frac{1 - exp(-x)}{1 + exp(-x)} \tag{3.40}$$

$$f_2(x) = \frac{1 - exp(-2.0 * x)}{1 + exp(-2.0 * x)} \tag{3.41}$$

$$f_3(x) = \frac{exp(x) - exp(-x)}{exp(x) + exp(-x)}. \tag{3.42}$$

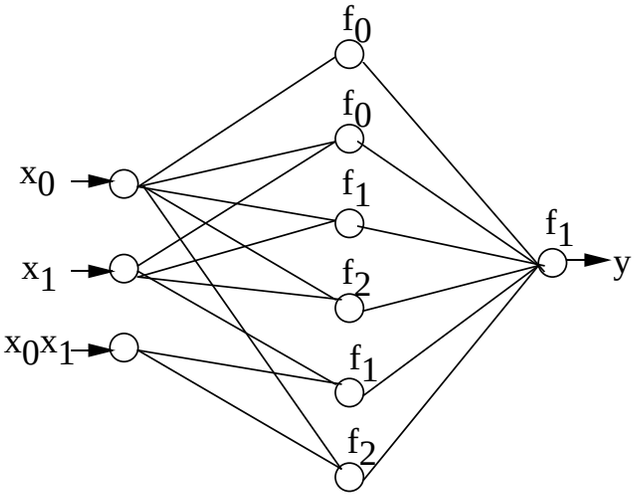


Figure 3.15: The evolved MLP neural network (Experiment 1)

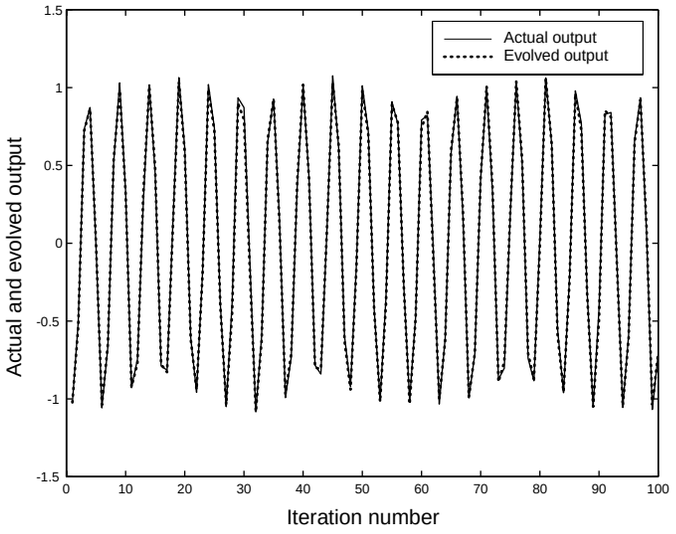


Figure 3.16: The actual and evolved output (Experiment 1)

3.3.2 Implementation and Experiments

In the simulation, the architecture of the non-regular MLP is evolved by MPIPE and the parameters used in the MLP are optimized by random search algorithm. The initial values of parameters used in random search are shown in Table 3.2.

Experiment 1

The first plant to be identified is a benchmark nonlinear autoregressive time series [9]

$$y(k) = (0.8 - 0.5exp(-y^2(k-1)))y(k-1) - (0.3 + 0.9exp(-y^2(k-1)))y(k-2)$$

Table 3.2: Parameters used in random search algorithm

β_0	β_1	α	ϕ_0	ϕ_{min}	I_{sf0}	P_{sf0}
0.1	1000	0.995	0.1	0.001	10	0.3
$\triangle I_{sf1}$	$\triangle I_{sf2}$	I_{smax}	K_{er}	c_i	c_d	
0.02	0.1	100	1.001	1.01	0.995	

$$+0.1sin(3.1415926y(k-1)) + e(k) \tag{3.43}$$

where the noise $e(k)$ was a Gaussian white sequence with mean zero and variance 0.02. 600 data points were generated and the first 500 points were used as training data set. The remaining data were used as a validation data set. The input vector is set as $x = [y(k-1), y(k-2)]$.

The used instruction sets are $I_0 = \{+_2, +_3, \ldots, +_{10}\}$, $I_1 = \{+_2\}$ and $I_2 = \{*, x_0, x_1\}$.

The MSE for training and test data sets are 0.000907 and 0.000821, respectively. Fig.3.16 shows the evolved MLP network. From Fig.3.15, it can be seen that the evolved MLP has incomplete interconnections, higher order inputs and different activation functions. The comparison of actual and evolved output for validation data set is shown in Fig.3.16.

Experiment 2

The plant to be identified is given by the following equation [22]

$$y(k+1) = \frac{y(k)y(k-1)y(k-2)u(k-1)(y(k-2)-1) + u(k)}{1 + y^2(k-1) + y^2(k-2)} \tag{3.44}$$

The input and output of system are $x(k) = [u(k), u(k-1), y(k), y(k-1), y(k-2)]$ and $y(k+1)$, respectively.

The data sets for training and test are generated using input signals shown in Eq. (3.45) and Eq. (3.46), respectively.

$$u(k) = \begin{cases} sin\left(\frac{2\pi k}{250}\right), & if\ (k < 500) \\ 0.8sin\left(\frac{2\pi k}{250}\right) + 0.2sin\left(\frac{2\pi k}{25}\right), & if\ (k >= 500) \end{cases} \tag{3.45}$$

$$u(k) = 0.3sin(k\pi/25) + 0.1sin(k\pi/32) + 0.1sin(k\pi/10) \tag{3.46}$$

The used instruction sets are $I_0 = \{+_2, +_3, \ldots, +_{15}\}$, $I_1 = \{+_2, +_3, +_4, +_5\}$ and $I_2 = \{*, x_0, x_1, x_2, x_3, x_4\}$.

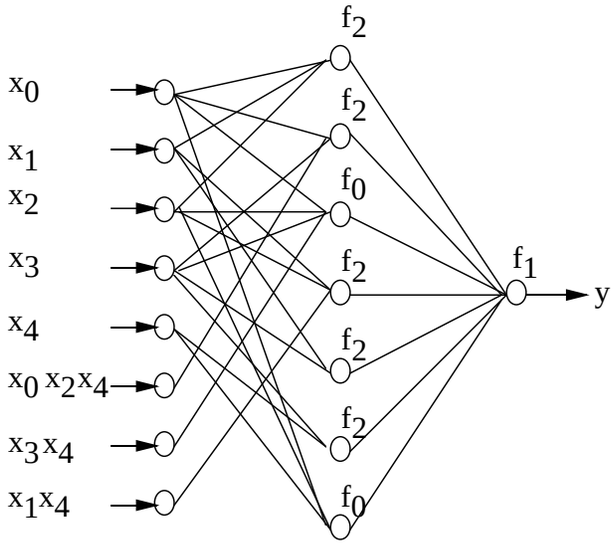


Figure 3.17: The evolved MLP neural network (Experiment 2)

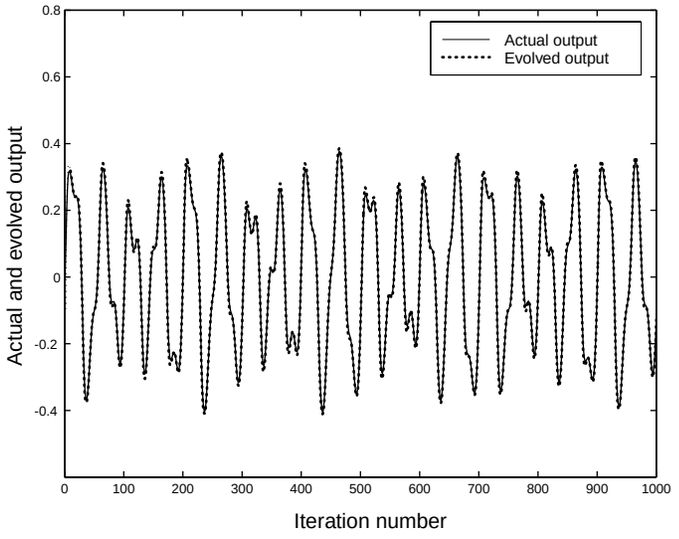


Figure 3.18: The actual and evolved output (Experiment 2)

The MSE for training and test data sets are 0.000091 and 0.000104, respectively. Fig.3.17 shows the evolved MLP network. From this figure, it can be seen that the size of evolved sparse MLP neural network is smaller (it only has 7 hidden neurons and 35 parameters including weights and biases). The comparison of actual and evolved output of is shown in Fig. 3.18.

Experiment 3

In this experiment, the unified framework is used to evolve the flexible non-regular MLP neural networks. The used activation function is the flexible bipolar sigmoid function as

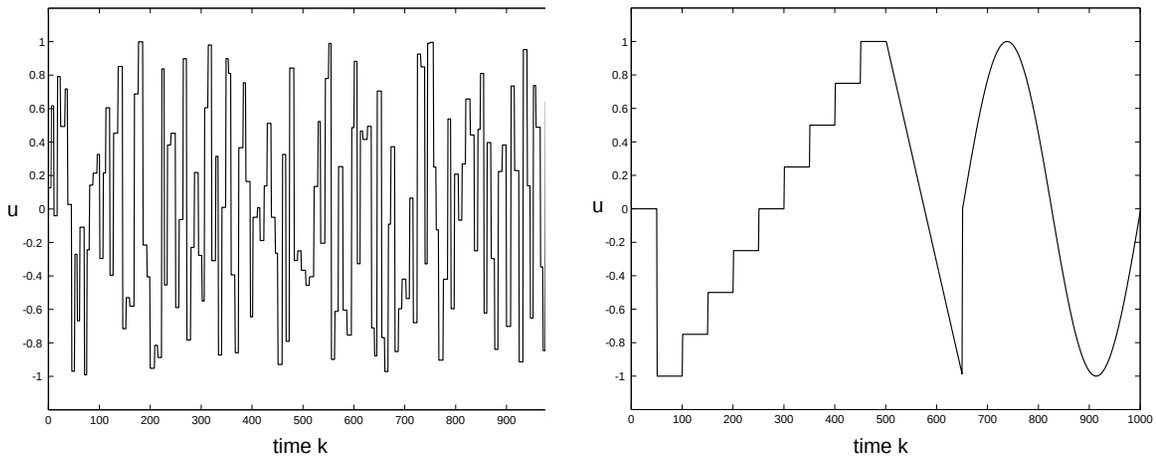


Figure 3.19: Input signals for generating the excitation and test data sets of the dynamic system. The left-hand side shows the input signal for creating the training data set and the right-hand side the test data set

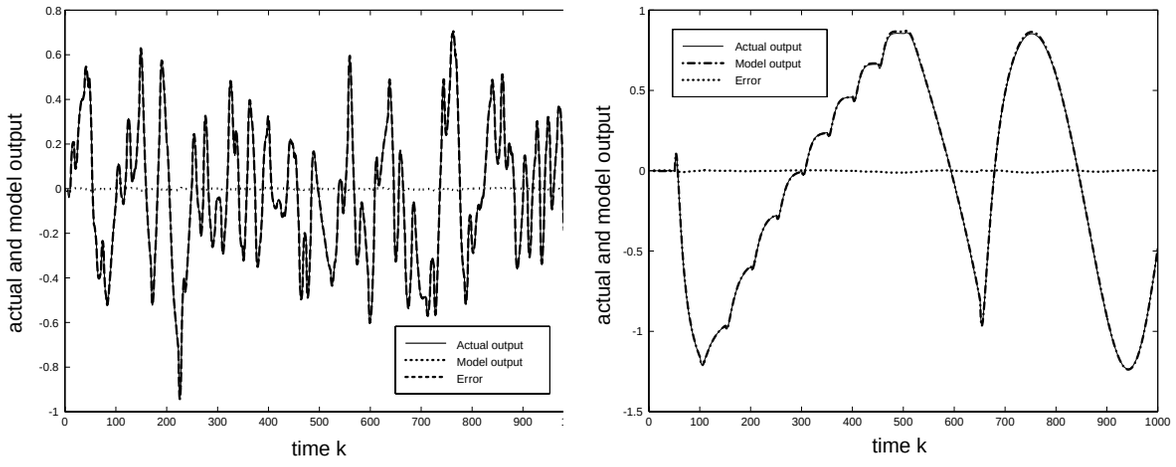


Figure 3.20: Comparison between process and simulated model output on training data set (left) and test data set (right)

shown in Table 2.2. The flexible parameter a is also optimized together with the other parameters by using random search algorithm.

A second-order non-minimum phase system with gain 1, time constants $4s$ and $10s$, a zero at $1/4s$, and output feedback with a parabolic nonlinearity is chosen to be identified. With sampling time $T_0 = 1s$ this system follows the nonlinear difference equation

$$y(k) = -0.07289[u(k-1) - 0.2y^2(k-1)] + 0.09394[u(k-2) - 0.2y^2(k-2)] \\ +1.68364y(k-1) - 0.70469y(k-2). \tag{3.47}$$

where the input lie in the interval [-1,1].

The training data and test data are generated using input signals shown in Fig. 3.19. The used instruction sets are $I_0 = \{+_3, \ldots, +_8\}$, $I_1 = \{*, u(k-1), u(k-2), y(k-1), y(k-2)\}$, and $I_2 = \{u(k-1), u(k-2), y(k-1), y(k-2)\}$. The evolved flexible MLP network has 3 hidden neurons. The used cost function is the sum of the absolute error (SAE). The SAE for training data and test data are 1.873652 and 2.349801, respectively. A comparison between process and simulated model output on training data set and test data set is shown in Fig. 3.20.

From above simulation results, it can be seen that the proposed approach works very well for generating the incomplete MLP networks and it can effectively used for the identification of nonlinear systems. In this sense, the proposed method can be seen as a power tool for evolving the non-regular MLP networks.

3.4 Optimization of the Basis Function Networks

3.4.1 Volterra Polynomial Basis Function Net

Volterra Polynomial Basis Function Network and Tree

A network whose basis functions consist of the Volterra polynomials is named as the Volterra polynomial basis function network (VPBF-NN) [17][18]. For function approximation, the usually used Volterra polynomial basis functions is

$$\phi = \{\phi_0(x), \phi_1(x), \ldots, \phi_N(x)\} = \{1, x_0, x_1, \ldots, x_n, x_0x_1, x_0x_2, \ldots, x_0x_n, x_1x_2, x_1x_3, \ldots, x_1x_n, \ldots, x_{n-1}x_n, {x_0}^2, {x_1}^2, \ldots, {x_n}^2\} \tag{3.48}$$

where $N = (n+1)(n+2)/2$ is the number of Volterra polynomial basis functions. n is the number of inputs. Therefore, the nonlinear function can be represented/approximated as

$$f(x) = \sum_{i=0}^{N} \omega_i * \phi_i(x) + O(x^3) \tag{3.49}$$

where $O(x^3)$ represents the model mismatch. For complex approximation problems, the higher order Volterra polynomial basis functions may be needed.

One of the type constrained sparse tree that represented as a VPBF-NN is shown in Fig.3.21. The used instruction sets for generating the tree are $I_0 = \{+_2, \ldots, +_N\}$, $I_1 = \{R, x_0, x_1, \ldots, x_n, *2, *3\}$ and $I_2 = \{x_0, x_1, \ldots, x_n\}$, where instruction $+N$ is used to control the number of Volterra polynomial basis functions for approximating the nonlinear functions at hand, the instruction $*2$ and $*3$ are used to select the higher components of

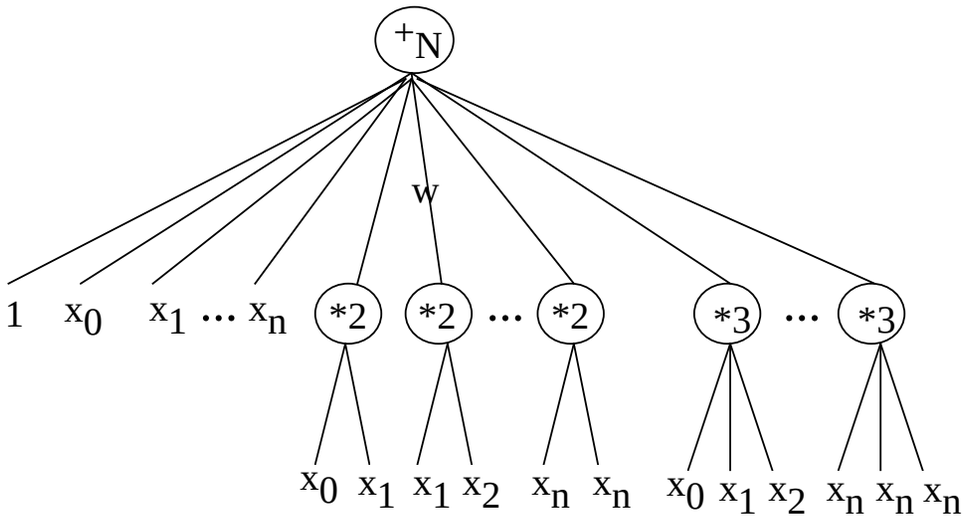


Figure 3.21: A tree structural representation of the VPBF network

the Volterra polynomial basis function set.

The main problem of optimizing the VPBF-NN is determination of network architecture which is also a model selection problem, that is, which Volterra basis functions are proper for the unknown nonlinear system, and estimating the corresponding parameters of each Volterra polynomial function.

3.4.2 GRBF, B-spline, Wavelet, Fuzzy Basis, Recurrent Fuzzy Neural, and Local linear GRBF Net and Trees

One of the type constrained sparse tree and its corresponding GRBF (B-spline, wavelet and fuzzy basis function) network are shown in Fig.3.22 and Fig.3.23, respectively. The used instruction sets for generating the tree are $I_0 = \{+_2, \ldots, +_N\}$, $I_1 = \{+_n\}$ and $I_2 = \{x_1, x_2, \ldots, x_n\}$. N is used to control the numbers of hidden neurons which is also the number of basis functions in the hidden layer of GRBF (B-spline, wavelet and fuzzy basis function) networks, and n is the number of inputs of GRBF (B-spline, wavelet and fuzzy basis function) networks.

- *GRBF Network*

  The used Gaussian radial basis function is given by

$$\phi_i(x) = exp(-\frac{\parallel x - c_i \parallel^2}{{d_i}^2}) \tag{3.50}$$

  where $\phi_i(x)$ is i-th Gaussian radial basis function, x is input vector, $x \in R^n$, c_i and d_i are the center and width of i-th basis function. Thus, the output of whole tree

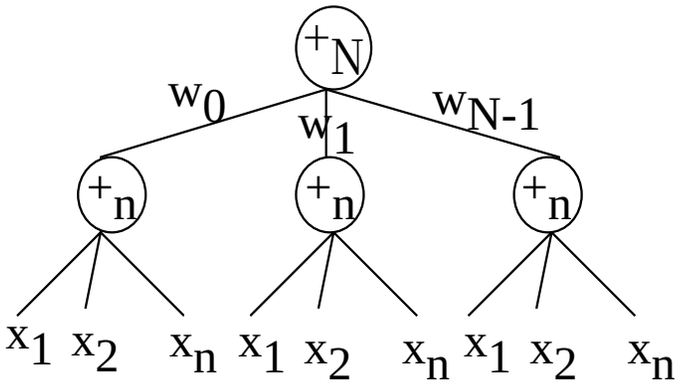


Figure 3.22: A type constrained sparse tree (genotype)

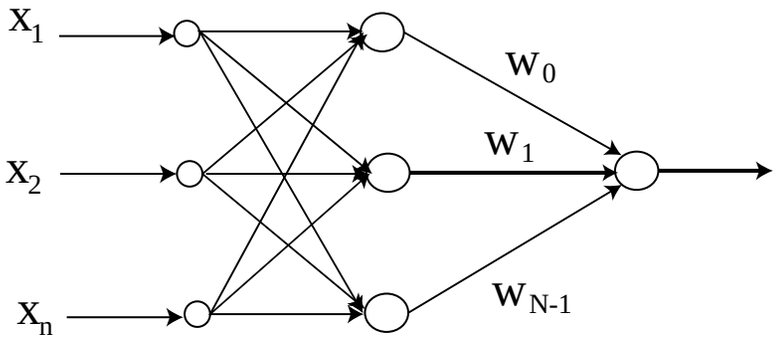


Figure 3.23: Corresponding GRBF(B-spline and fuzzy basis function) network of the type constrained sparse tree (phenotype)

in the Fig.3.22 is calculated as

$$y = \sum_{i=0}^{N-1} \omega_i * exp(-\frac{\parallel x - c_i \parallel^2}{{d_i}^2}) \tag{3.51}$$

The objective of optimization of GRBF networks is determination of a number of basis functions, the center and width of each Gaussian radial basis functions by appropriate methods.

- *B-spline Network*

  The B-spline functions can be defined in a recursive way,

$$B_0(t) = \begin{cases} 1, & t \in [-\frac{1}{2}, \frac{1}{2}] \\ 0, & otherwise \end{cases} \tag{3.52}$$

$$B_m(t) = (B_{m-1} * B_0)(t) \tag{3.53}$$

The translation and dilation of the order 2 B-spline function is given by

$$f_{a,b}(t) = B_2(\frac{t-b}{a}) = \begin{cases} \frac{9}{8} + \frac{3}{2}(\frac{t-b}{a}) + \frac{1}{2}(\frac{t-b}{a})^2, & t \in [-\frac{3}{2}a + b, -\frac{1}{2}a + b) \\ \frac{3}{4} - (\frac{t-b}{a})^2, & t \in [-\frac{1}{2}a + b, \frac{1}{2}a + b) \\ \frac{9}{8} - \frac{3}{2}(\frac{t-b}{a}) + \frac{1}{2}(\frac{t-b}{a})^2, & t \in [\frac{1}{2}a + b, \frac{3}{2}a + b] \\ 0, & otherwise \end{cases} \tag{3.54}$$

The single input B-spline basis function is given by Eq.(3.54). For the n-dimensional input space $x = [x_0, x_1, \ldots, x_{n-1}]$, the multivariate B-spline functions is selected by the product of single B-spline functions

$$N_i(x) = \prod_{j=0}^{n-1} B_2(\frac{x_j - b_j}{a_j}) \tag{3.55}$$

Thus, the output of whole tree in the Fig.3.22 is calculated as

$$y = \sum_{i=0}^{N-1} \omega_i * N_i(x) = \sum_{i=0}^{N-1} \omega_i * \prod_{j=0}^{n-1} B_2(\frac{x_j - b_j}{a_j}) \tag{3.56}$$

The objective of optimization of B-spline networks is determination of a number of basis functions, the center and width of each B-spline basis functions by appropriate methods. It can be seen that in contrast to the usually methods the pre-dividing of input space is not needed in our approach.

- *Wavelet Neural Network*

  Given the mother wavelet, a family of equally shaped functions by shifts in time (translation) and scaling (dilation) can be obtained as

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\psi(\frac{t-b}{a}), a \neq 0, a, b \in R \tag{3.57}$$

  This is a single variable wavelet basis function. For the n-dimensional input space $x = [x_1, x_2, \ldots, x_n]$, the multivariate wavelet basis function is calculated by the product of n single wavelet basis functions

$$s_i(x) = \prod_{j=1}^{n} \frac{1}{\sqrt{|a_j|}}\psi(\frac{x_j - b_j}{a_j}) \tag{3.58}$$

Thus, the overall output of the wavelet basis function network is calculated as

$$y = \sum_{i=0}^{N-1} \omega_i * s_i(x) = \sum_{i=0}^{N-1} \omega_i * \prod_{j=0}^{n-1} \frac{1}{\sqrt{|a_j|}} \psi(\frac{x_j - b_j}{a_j}) \tag{3.59}$$

The problem in designing the wavelet basis function network is to determine an optimal network size (the numbers of the wavelet basis functions), the parameter a_j and b_j $(j = 0, 1, \ldots, (N-1) * (n-1))$ for each single wavelet basis function. It can be seen that in contrast to the usually methods the pre-determining of the parameter a_j and b_j is not needed in our approach.

- *Fuzzy basis function Networks*

  In adaptive fuzzy systems, the fuzzy rule is represented as (T-S Model)

  if x_0 is A_{i0} and x_1 is A_{i1} ... and x_{n-1} is $A_{i(n-1)}$
  then $y_i = b_{i0}x_0 + b_{i1}x_1 + ... + b_{i(n-1)}x_{n-1} + b_{in}$
  (i=0, 1, ... N-1)

  When algebraic operator are used to implement fuzzy logic functions, the real valued inputs are represented via fuzzy membership function, and a center of defuzzification method is used, then the output of fuzzy basis function network is given by

$$y = \sum_{i=0}^{N-1} \phi_i(x) * \omega_i \tag{3.60}$$

  where $\phi_i(x)$ is the multivariate fuzzy basis function. It is calculated as

$$\phi_i(x) = \frac{\prod_{j=0}^{n-1} \mu_{A_{ij}}(x_j)}{\sum_{i=0}^{N-1} \prod_{j=0}^{n-1} \mu_{A_{ij}}(x_j)} \tag{3.61}$$

  Thus, the output of the fuzzy basis function networks or corresponding tree is calculated as

$$y = \sum_{i=0}^{N-1} \omega_i * \frac{\prod_{j=0}^{n-1} \mu_{A_{ij}}(x_j)}{\sum_{i=0}^{N-1} \prod_{j=0}^{n-1} \mu_{A_{ij}}(x_j)} \tag{3.62}$$

  The objective are that directly learning and evolving a number of fuzzy rules, determining the parameters of each fuzzy membership functions, and optimizing the corresponding weight of each fuzzy basis functions for fitting a given data set. In our experiments of this research the selected fuzzy membership function is Cauchy

function. It is also valuable to mention that the initial fuzzy partition of input space is not needed in our approaches.

- *Recurrent Fuzzy Neural Network*

  The difference between the FNN and RFNN can be clearly distinguished as follows:

  – The input/output representation of FNN

$$y(k) = \sum_{j=1}^{m} \omega_{mj} \prod_{i=1}^{n} exp(-\frac{(x_i(k) - m_{ij})^2}{(\sigma_{ij})^2}) \tag{3.63}$$

  – The input/output representation of RFNN

$$y(k) == \sum_{j=1}^{m} \omega_{mj} \prod_{i=1}^{n} exp(-\frac{(x_i(k) + O_{ij}^2(k-1) \cdot \theta_{ij} - m_{ij})^2}{(\sigma_{ij})^2}) \tag{3.64}$$

$$O_{ij}^2(k-1) = exp(-\frac{(x(k-1) + O_{ij}^2(k-2) \cdot \theta_{ij} - m_{ij})^2}{(\sigma_{ij})^2}) \tag{3.65}$$

  where n is the number of input variables, m is the number of term nodes for each input variable. The used membership function is the Gaussian function,

$$\mu_{ij} = exp(-\frac{(x_{ij} - m_{ij})^2}{(\sigma_{ij})^2}) \tag{3.66}$$

  where m_{ij} and σ_{ij} are the center and the width of Gaussian membership function. The subscript ij indicates the j-th term of the i-th input x_i.

- *Local Linear Gaussian Basis Function Network*

  Local linear Gaussian basis function network is an extended radial function network that is obtained by replacing the output layer weights with a linear function of the network inputs. Each neuron represents a local linear model with its corresponding validity function. Further more, the radial basis function network is normalized, that is the sum of all validity functions for a specific input combination sums up to one. The Gaussian validity functions determine the regions of the input space where each neuron is active. The input space of the net is divided into N hyper-rectangles each represented by a linear function.

  The output of a local linear Gaussian basis function network with n inputs x_1,x_2,. . .,

x_n is calculated by summing up the contributions of all N local linear models

$$y = \sum_{i=1}^{N} (\omega_{i0} + \omega_{i1}x_1 + \ldots + \omega_{in}x_n)\phi_i(x) \tag{3.67}$$

where ω_{ij} are the parameters of the ith linear regression model and x_i is the model input. The validity functions ϕ_i are typically chosen as normalized Gaussian weighting function:

$$\phi_i(x) = \frac{\mu_i}{\sum_{j=1}^{N} \mu_j} \tag{3.68}$$

with

$$\mu_i = exp(-\frac{1}{2}\frac{(x_1 - c_{i1})^2}{\sigma_{i1}^2} - \ldots - \frac{1}{2}\frac{(x_n - c_{in})^2}{\sigma_{in}^2}) \tag{3.69}$$

3.4.3 Implementation and Experiments

In this experiments, the structure of the basis function networks is evolved by MPIPE, and the parameters used in basis function networks are optimized by a hybrid learning strategy, that is, the weight (linear-in-parameter) is optimized by least square algorithm, and the center and width (nonlinear-in-parameter) are optimized by random search algorithm. The used parameters in PIPE are shown in Table 3.1. The initial values of parameters in random search are shown in Table 3.2.

Example 1

The first plant to be identified is a benchmark nonlinear autoregressive time series [26]

$$y(k) = (0.8 - 0.5exp(-y^2(k-1)))y(k-1) - (0.3 + 0.9exp(-y^2(k-1)))y(k-2)$$
$$+0.1sin(3.1415926y(k-1)) + e(k) \tag{3.70}$$

where the noise $e(k)$ was a Gaussian white sequence with mean zero and variance 0.02. 600 data points were generated and the first 500 points were used as an estimation data set. The remaining data were used as a validation data set. The input vector is set as $x = [y(k-1), y(k-2)]$.

Experiment 1: VPBF Network

The used instruction sets are $I_0 = \{+_5, +_6, \ldots, +_{15}\}$, $I_1 = \{R, x_0, x_1, *_2, *_3\}$ and $I_2 = \{x_0, x_1\}$.

The optimal model (shown in Eq. (3.71)) is obtained with the mean square errors (MSE) for training data set and test data set are 0.000587 and 0.000596, respectively.

$$f(x_0, x_1) = -0.0024 - 1.23x_0 + 0.60x_1 - 0.001{x_0}^2$$

$$+0.003x_1x_2 + 0.65x_0{x_1}^2 + 0.056{x_0}^3 \tag{3.71}$$

Experiment 2: GBRF Network

The used instruction sets are $I_0 = \{+_5, +_6, \ldots, +_{15}\}$, $I_1 = \{+_2\}$ and $I_2 = \{x_0, x_1\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000602 and 0.000653, respectively. The evolved model has 18 Gaussian basis functions.

Experiment 3: B-Spline Network

The used instruction sets are $I_0 = \{+_5, +_6, \ldots, +_{15}\}$, $I_1 = \{+_2\}$ and $I_2 = \{x_0, x_1\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000523 and 0.000618, respectively. The evolved model has 16 B-spline basis functions.

Experiment 4: Wavelet basis Network

The used instruction sets are $I_0 = \{+_3, +_4, \ldots, +_{12}\}$, $I_1 = \{+_2\}$ and $I_2 = \{x_0, x_1\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000471 and 0.000537, respectively. The evolved model has 15 wavelet basis functions.

Experiment 5: Fuzzy basis Network

The used instruction sets are $I_0 = \{+_5, +_6, \ldots, +_{15}\}$, $I_1 = \{+_2\}$ and $I_2 = \{x_0, x_1\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000662 and 0.000570, respectively. The evolved model has 16 fuzzy basis functions.

Experiment 6: Recurrent fuzzy neural network

The used instruction sets are $I_0 = \{+_5, +_6, \ldots, +_{15}\}$, $I_1 = \{+_2\}$ and $I_2 = \{x_0, x_1\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000362 and 0.000371, respectively. The evolved model has 14 fuzzy basis functions.

Experiment 7: Local RBF neural network

The used instruction sets are $I_0 = \{+_5, +_6, \ldots, +_{15}\}$, $I_1 = \{+_2\}$ and $I_2 = \{x_0, x_1\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000427 and 0.000450, respectively. The evolved model has 12 fuzzy basis functions.

Example 2

The second plant to be identified is given by the following equation

$$y(k+1) = \frac{y(k)y(k-1)y(k-2)u(k-1)(y(k-2)-1)+u(k)}{1+y^2(k-1)+y^2(k-2)} \tag{3.72}$$

The input and output of system are $x(k) = [u(k), u(k-1), y(k), y(k-1), y(k-2)]$ and $y(k+1)$, respectively.

The training samples and the test data set are generated using following input signals, respectively.

$$u(k) = \begin{cases} sin\left(\frac{2\pi k}{250}\right), if\ (k < 500) \\ 0.8sin\left(\frac{2\pi k}{250}\right) + 0.2sin\left(\frac{2\pi k}{25}\right), \quad if\ (k >= 500) \end{cases} \tag{3.73}$$

$$u(k) = 0.3sin(k\pi/25) + 0.1sin(k\pi/32) + 0.1sin(k\pi/10) \tag{3.74}$$

Experiment 1: VPBF Network

The used instruction sets are $I_0 = \{+_5, +_6, \ldots, +_{15}\}$, $I_1 = \{R, x_0, x_1, x_2, x_3, x_4, *_2\}$ and $I_2 = \{x_0, x_1, x_2, x_3, x_4\}$.

The optimal model (shown in Eq. (3.75)) is obtained with the mean square errors (MSE) for training data set and test data set are 0.000041 and 0.000071, respectively.

$$\begin{aligned} f(x_0, x_1, x_2, x_3, x_4) &= 0.0003 - 0.63x_0 + 0.64x_1 - 0.28x_2 + 0.46x_3 + 0.8x_4 - 0.49{x_3}^2 \\ &+0.28x_0x_2 + 0.19x_3x_4 - 0.14x_1x_4 - 0.12x_1x_3 - 0.05x_1x_2 + 0.34x_2x_4 \end{aligned} \tag{3.75}$$

Experiment 2: GBRF Network

The used instruction sets are $I_0 = \{+_8, +_6, \ldots, +_{18}\}$, $I_1 = \{+_5\}$ and $I_2 = \{x_0, x_1, x_2, x_3, x_4\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000151 and 0.000126, respectively. The evolved model has 11 Gaussian basis functions.

Experiment 3: B-Spline Network

The used instruction sets are $I_0 = \{+_8, +_6, \ldots, +_{18}\}$, $I_1 = \{+_5\}$ and $I_2 = \{x_0, x_1, x_2, x_3, x_4\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000133 and 0.000097, respectively. The evolved model has 19 B-spline basis functions.

Experiment 4: Wavelet basis Network

The used instruction sets are $I_0 = \{+_5, +_6, \ldots, +_{15}\}$, $I_1 = \{+_2\}$ and $I_2 = \{x_0, x_1, x_2, x_3, x_4\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000179 and 0.000121, respectively. The evolved model has 19 wavelet basis functions.

Experiment 5: Fuzzy basis Network

The used instruction sets are $I_0 = \{+_5, +_6, \ldots, +_{18}\}$, $I_1 = \{+_5\}$ and $I_2 = \{x_0, x_1, x_2, x_3, x_4\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000662 and 0.000570, respectively. The evolved model has 16 fuzzy basis functions.

Experiment 6: Recurrent fuzzy neural network

The used instruction sets are $I_0 = \{+_5, +_6, \ldots, +_{15}\}$, $I_1 = \{+_5\}$ and $I_2 = \{x_0, x_1, x_2, x_3, x_4, x_5\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000113 and 0.000055, respectively. The evolved model has 10 fuzzy basis functions.

Experiment 7: Local RBF neural network

The used instruction sets are $I_0 = \{+_5, +_6, \ldots, +_{15}\}$, $I_1 = \{+_5\}$ and $I_2 = \{x_0, x_1, x_2, x_3, x_4, x_5\}$.

The optimal model is obtained with the mean square errors (MSE) for training data set and test data set are 0.000124 and 0.000087, respectively. The evolved model has 12 fuzzy basis functions.

From the above simulation results it can be seen that the proposed methods works very well for generating the different basis function networks. For the comparison (see Table 3.3), the VPBF network in general has smaller number of parameters, good approximation ability and fast convergence speed. In addition, the evolved GBRF,B-spline, fuzzy, recurrent fuzzy neural, wavelet and local RBF basis function networks have smaller size and parameters than the conventional methods.

Table 3.3: The number of parameters, the training and test error of the evolved basis function networks

Experiment	No. of parameters	MSE for training	MSE for validation
Ex. 1 VPBF-NN	7	0.000587	0.000596
Ex. 1 GRBF-NN	72	0.000602	0.000653
Ex. 1 B-spline-NN	80	0.000523	0.000618
Ex. 1 Wavelet-NN	75	0.000471	0.000537
Ex. 1 FNN	80	0.000662	0.000570
Ex. 1 RFNN	96	0.000362	0.000371
Ex. 1 Local-RBF-NN	96	0.000427	0.000450
Ex. 2 VPBF-NN	13	0.000041	0.000071
Ex. 2 GRBF-NN	77	0.000151	0.000126
Ex. 2 B-spline-NN	209	0.000133	0.000097
Ex. 2 Wavelet-NN	209	0.000179	0.000121
Ex. 2 FNN	176	0.000141	0.000065
Ex. 2 RFNN	156	0.000113	0.000055
Ex. 2 Local-RBF-NN	149	0.000124	0.000087

3.5 Evolutionary Design of Hierarchical T-S Fuzzy Models by Modified PIPE and Evolutionary Programming

3.5.1 Introduction

Fuzzy logic systems [117][118] have been successfully applied to a vast number of scientific and engineering problems in recent years. The advantage of solving the complex nonlinear problems by utilizing fuzzy logic methodologies is that the experience or expert's knowledge described as the fuzzy rule base can be directly embedded into the systems for dealing with the problems. A number of improvements have been made in the aspects of enhancing the systematic design method of fuzzy logic systems [119]-[124]. In these researches, the needs for effectively tuning the parameters and structure of fuzzy logic systems are increased. Many researches focus on the automatically finding the proper structure and parameters of fuzzy logic systems by using genetic algorithms [120][123][124], evolutionary programming [122], tabu search [125], and so on. But there still remains some problems to be solved, for example, how to automatically partition the input space for each input-output variables, how many fuzzy rules are really needed for properly approximating the unknown nonlinear systems, and how to determine it automatically. As is well known, the

curse-of-dimensionality is an unsolved problem in the fields of fuzzy and/or neurofuzzy systems.

The problems mentioned above can be partially solved by the recent developments of hierarchical fuzzy systems [125]-[133]. As a way to overcome the curse-of dimensionality, it was suggested in [133] to arrange several low-dimensional rule base in a hierarchical structure, i.e. a tree, causing the number of possible rules to grow in a linear way with a number of inputs. But no method was given on how the rules for these hierarchical fuzzy systems could be determined automatically. In [131] the author describes a new algorithm, which derives the rules for hierarchical fuzzy associative memories that are structured as a binary tree. In Ref. [128][127][134], a specific hierarchical fuzzy system is proposed and its universal approximation property was proved. But the main problems in fact lies that this is a specific hierarchical fuzzy systems which lack of the flexibility in the structure adaptation, and it is difficult to arrange the input variables due to there is no general method to determine which inputs to the hierarchical fuzzy system are more influential to the output. In [135], a genetic algorithm-based evolutionary learning approach to the search for the best hierarchical structure of the five input-variable fuzzy controller and the parameters of the combined controller is proposed for the low-speed control.

In intuition, the hierarchical fuzzy logic systems not only provide a more complex and flexible structure for modeling the nonlinear systems, but can also reduce the size of rule base in some extend. But there is no systematic method for designing of the hierarchical T-S fuzzy systems now. The problems in designing of hierarchical fuzzy logic system lie that:

- Selecting a proper hierarchical structure;
- Selecting the inputs for each partial fuzzy model;
- Determining the rule base of each fuzzy logic T-S model;
- Optimizing the parameters used in the fuzzy membership functions and the then-part of T-S fuzzy model.

In this sense, finding a proper hierarchical T-S fuzzy model can be posed as a search problem in the structure and parameter space. Fig. 3.24 shows some possible hierarchical T-S fuzzy models for the number of input variables 4 and the number of hierarchical layers 3. It can be seen that

- It is important to select a proper hierarchical T-S fuzzy model structure for approximating a unknown nonlinear system;

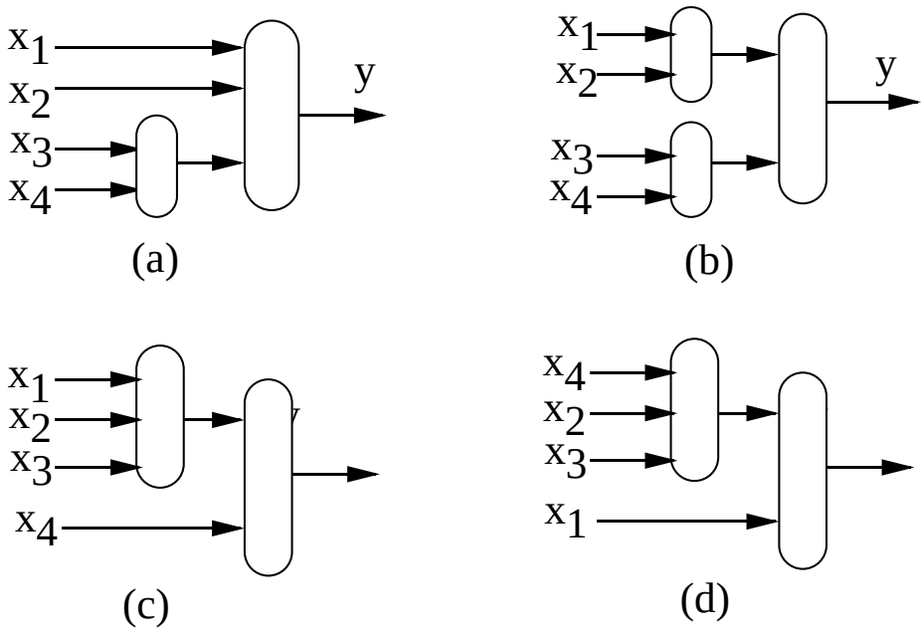


Figure 3.24: Example of possible hierarchical fuzzy logic models, number of inputs: 4, number of layers: 3.

- It is important to select a proper hierarchical T-S fuzzy model structure for approximating a unknown nonlinear system;
- Due to there is no general conclusion about which inputs are more influential to the system output, thus it is important to select the appropriate input in hierarchical T-S fuzzy model. In fact, different inputs selection can form the different hierarchical models (see Fig. 3.24 (c) and (d));
- If each variable is divided into 2 fuzzy sets, then the number of fuzzy rules in each of hierarchical fuzzy systems (Fig.3.24 (a), (b), (c) and (d)) is 12 which is in general smaller than the number of rules in non-hierarchical fuzzy systems with complete rule set.

In this section, a method for automatically designing of hierarchical T-S fuzzy systems is proposed. The structure discovery and parameter adjustment of the hierarchical T-S fuzzy model is addressed by a hybrid technique of type constrained sparse tree, MPIPE and EP algorithms. Our researches demonstrate that the model structure selection and parameters optimization of the hierarchical T-S fuzzy model can be flexibly coded as a type constrained sparse tree. Therefore, the optimal nonparametric model of nonlinear systems can be obtained by the evolutionary induction of the type constrained sparse tree. Furthermore, the architecture of the hierarchical T-S fuzzy model is evolved by MPIPE

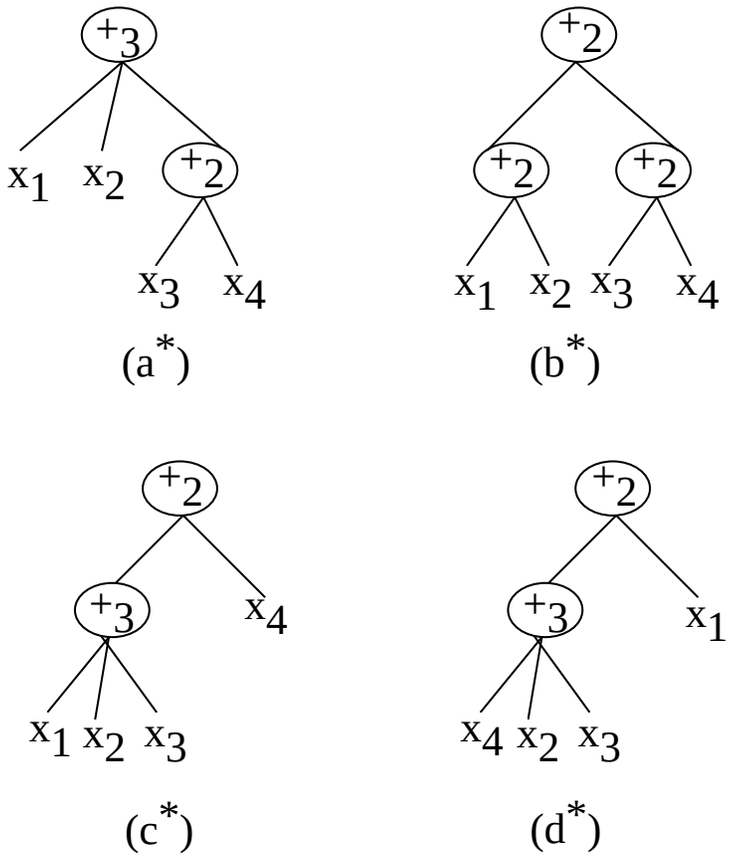


Figure 3.25: Tree structural representation of hierarchical T-S fuzzy models shown in Fig.3.24 (a), (b), (c) and (d).

algorithm [126]-[130], and the corresponding parameters are optimized by modified EP algorithm, respectively.

3.5.2 Representation and Calculation of Hierarchical T-S fuzzy model

In this section, we focus on the tree structural representation of the hierarchical T-S fuzzy model, in which each of the hierarchical T-S fuzzy model can be coded as a type constrained sparse tree. There is no need to encode and decode between the tree and the hierarchical T-S fuzzy model in the calculation of the hierarchical T-S fuzzy model due to we put a flexible data structure into the nodes of the tree which can be directly calculated in a recursive way. Therefore, the optimization of the hierarchical T-S fuzzy model can be directly replaced by the evolutionary induction of the type constrained sparse tree.

Creation of the Tree and Its Data Structure

Suppose that the hierarchical T-S fuzzy models have three hierarchical layers. In this case, three instruction sets I_0, I_1 and I_2 can be used for generating the tree. In general, the used instruction sets are $I_0 = \{+_2, +_3, \ldots, +_{n_1}\}$, $I_1 = \{\ x_1,\ x_2,\ \ldots, x_n,\ +2, +3, \ldots, +_{n_2}\}$ and $I_2 = \{x_1, x_2, \ldots, x_n\}$, where the instruction $+_{n_i}$ $(i = 2, 3, \ldots, n_1)$ in the instruction set I_0 means that there are n_i input variables in the output layer of the T-S fuzzy model, which is also the number of branches of the root node. The instructions in the instruction set I_1 are used to generate the hidden layer of the T-S fuzzy models. It can be seen that if all the instructions used in the hidden layer are input variables the T-S hierarchical model becomes an usually used non-hierarchical fuzzy model. In contrast, if there is one instruction which is not any one of the input variables $x_1, x_2, \ldots, x_n$, a hierarchical T-S fuzzy model is created. This means that the non-input variable instruction becomes a sub-T-S fuzzy model, which has its input variables come from the instruction set I_2 and it's output is the input of the next layer of the T-S fuzzy model. The instructions used in the layer 0, 1 and 2 of the tree are randomly selected from instruction set I_0, I_1 and I_2, respectively. The initial probability of selecting instructions is given by

$$P\left(I_{d,w}\right) = \frac{1}{l_i}, \forall I_{d,w} \in I_i, i = 0, 1, 2 \tag{3.76}$$

where $I_{d,w}$ denotes the instruction of the node with depth d and width w, $l_i (i = 0, 1, 2)$ is the number of instructions in the instruction set I_i.

As mentioned above, the tree (T-S hierarchical fuzzy model) can be generated in a recursive way (deepest first) as follows:

(1) Select a root node from the instruction set I_0 according to the probability of selecting instructions. If the number of arguments of the selected instruction is larger than one, for example, in the Fig. 3.25 (a^*) the root node $+_3$ has three arguments, which means that this is a three inputs T-S fuzzy model, then create a number of parameters (including the fuzzy membership function parameters for each input variable, the THEN-part parameters in each fuzzy rule) attached to the node.

(2) Create the left sub-node of root from the instruction set I_1. If the number of arguments of the selected instruction is larger than one, then create a number of parameters attached to the node, and then create its left sub-node as same way from the instruction set I_2. Otherwise, if the instruction of the node is an input variable, return to upper layer of the tree in order to generate the right part of the tree.

(3) Repeat this process until a full tree is generated.

Remark

There are two key points in the generation of the tree. The one is that the instruction is selected according to the initial probability of selecting instructions, and the probability will be changed with generation by a probabilistic incremental algorithm (see Section 3.1). The other is that if the selected node is a non-terminal node, then generate the corresponding data structure (parameters used in hierarchical T-S fuzzy model) attached to the node.

Fig. 3.25 shows tree structural representation of the hierarchical T-S fuzzy models shown in Fig. 3.24.

Calculation of the Tree

The value of the tree is calculated in a recursive way. In the following the calculation process of the tree (Fig. 3.25 (a^*)) is given for displaying the method.

Suppose that each input variable is divided into two fuzzy sets and the used fuzzy membership function is

$$\mu(a, b; x) = \frac{1.0}{1.0 + (\frac{x-a}{b})^2} \tag{3.77}$$

Firstly the corresponding fuzzy rules of node +2 can be formed as follows [31]:

$$\begin{aligned}
&\text{R1}: \text{ if } x_3 \text{ is } A_{11} \text{ and } x_4 \text{ is } A_{21} \text{ then } y1 = k_1 * (a_1 x_3 + a_2 * x_4) \\
&\text{R2}: \text{ if } x_3 \text{ is } A_{11} \text{ and } x_4 \text{ is } A_{22} \text{ then } y2 = k_2 * (a_1 x_3 + a_2 * x_4) \\
&\text{R3}: \text{ if } x_3 \text{ is } A_{12} \text{ and } x_4 \text{ is } A_{21} \text{ then } y3 = k_3 * (a_1 x_3 + a_2 * x_4) \\
&\text{R4}: \text{ if } x_3 \text{ is } A_{12} \text{ and } x_4 \text{ is } A_{22} \text{ then } y4 = k_4 * (a_1 x_3 + a_2 * x_4)
\end{aligned}$$

The output of node $+_2$ can be calculated based on the T-S fuzzy model:

$$y = \frac{\mu_{A_{11}}(x_3)\mu_{A_{21}}(x_4)y1 + \mu_{A_{11}}(x_3)\mu_{A_{22}}(x_4)y2 + \mu_{A_{12}}(x_3)\mu_{A_{21}}(x_4)y3 + \mu_{A_{12}}(x_3)\mu_{A_{22}}(x_4)y4}{\mu_{A_{11}}(x_3)\mu_{A_{21}}(x_4) + \mu_{A_{11}}(x_3)\mu_{A_{22}}(x_4) + \mu_{A_{12}}(x_3)\mu_{A_{21}}(x_4) + \mu_{A_{12}}(x_3)\mu_{A_{22}}(x_4)} \tag{3.78}$$

Now the overall output of tree in Fig.3.24 (a^*) can be calculated based on the following fuzzy rules:

R1 : if x_1 is B_{11} and x_2 is B_{21} and y is B_{31} then $z_1 = c_1 * (d_1x_1 + d_2x_2 + d_3y)$
R2 : if x_1 is B_{11} and x_2 is B_{21} and y is B_{32} then $z_2 = c_2 * (d_1x_1 + d_2x_2 + d_3y)$
R3 : if x_1 is B_{11} and x_2 is B_{22} and y is B_{31} then $z_3 = c_3 * (d_1x_1 + d_2x_2 + d_3y)$
R4 : if x_1 is B_{11} and x_2 is B_{22} and y is B_{32} then $z_4 = c_4 * (d_1x_1 + d_2x_2 + d_3y)$
R5 : if x_1 is B_{12} and x_2 is B_{21} and y is B_{31} then $z_5 = c_5 * (d_1x_1 + d_2x_2 + d_3y)$
R6 : if x_1 is B_{12} and x_2 is B_{21} and y is B_{32} then $z_6 = c_6 * (d_1x_1 + d_2x_2 + d_3y)$
R7 : if x_1 is B_{12} and x_2 is B_{22} and y is B_{31} then $z_7 = c_7 * (d_1x_1 + d_2x_2 + d_3y)$
R8 : if x_1 is B_{12} and x_2 is B_{22} and y is B_{32} then $z_8 = c_8 * (d_1x_1 + d_2x_2 + d_3y)$

The overal output of the tree is

$$z = \frac{\sum_{j=1}^{8} \mu_j z_j}{\sum_{j=1}^{8} \mu_j} \tag{3.79}$$

where

$$\begin{aligned}
\mu_1 &= \mu_{B_{11}}(x1)\mu_{B_{21}}(x2)\mu_{B_{31}}(y) \\
\mu_2 &= \mu_{B_{11}}(x1)\mu_{B_{21}}(x2)\mu_{B_{32}}(y) \\
\mu_3 &= \mu_{B_{11}}(x1)\mu_{B_{22}}(x2)\mu_{B_{31}}(y) \\
\mu_4 &= \mu_{B_{11}}(x1)\mu_{B_{22}}(x2)\mu_{B_{32}}(y) \\
\mu_5 &= \mu_{B_{12}}(x1)\mu_{B_{21}}(x2)\mu_{B_{31}}(y) \\
\mu_6 &= \mu_{B_{12}}(x1)\mu_{B_{21}}(x2)\mu_{B_{32}}(y) \\
\mu_7 &= \mu_{B_{12}}(x1)\mu_{B_{22}}(x2)\mu_{B_{31}}(y) \\
\mu_8 &= \mu_{B_{12}}(x1)\mu_{B_{22}}(x2)\mu_{B_{32}}(y)
\end{aligned}$$

where all the parameters k_i (i=1,2,3,4), a_1, a_2, and the shape parameters used in fuzzy sets A_{11}, A_{12}, A_{21} and A_{22} are attached to the node $+_2$. The parameters c_j (j=1,2...8), d_1, d_2, d_3 and the shape parameters used in fuzzy sets B_{11}, B_{12}, B_{21}, B_{22}, B_{31} and B_{32} are attached to the node $+_3$ of the tree. All these parameters are randomly generated alone with the creation of the tree at initial step, and will be optimized by EP algorithm.

3.5.3 Implementation and Experiments

In this section, we give some simulation results to verify the effectiveness of the proposed method for the identification of nonlinear system. The architecture of the hierarchical T-S fuzzy model is evolved by MPIPE and the parameters such as the membership function parameters and the parameters used in conclusion part of the fuzzy rule, are optimized by a global search algorithm- modified evolutionary programming.

The used parameters in MPIPE are shown in Table 3.4. The parameters used in EP: population size is 20, opponent number $Q = 12$, $\alpha = 0.3$. In each generation, the

Table 3.4: Parameters used in MPIPE Algorithm

Population Size PS	10
Elitist Learning Probability P_{el}	0.01
Learning Rate lr	0.01
Fitness Constant ε	0.000001
Overall Mutation Probability P_M	0.4
Mutation Rate mr	0.4

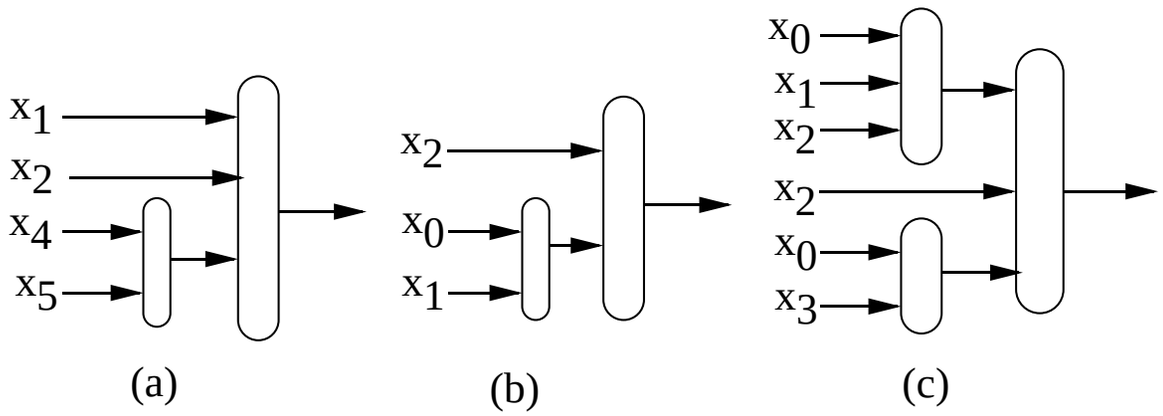


Figure 3.26: The evolved hierarchical T-S fuzzy models, (a) plant 1, (b) plant 2, and (c) plant 3

maximum number of EP search is

$$step = \beta * (1 + generation) \tag{3.80}$$

where β is the basic steps of EP search.

Example 1

Let's the plant be given by [25],

$$y(k+1) = 2.627771y(k) - 2.333261y(k-1) + 0.697676y(k-2) + 0.017203u(k)$$
$$-0.030862u(k-1) + 0.014086u(k-2) \tag{3.81}$$

400 data points were generated with the randomly selected input signal $u(k)$ between -1.0 and 1.0. The first 200 points were used as an estimation data set and the remaining data were used as a validation data set. The input vector is set as $x = [y(k), y(k-1), y(k-2), u(k), u(k-1), u(k-2)]$. The used instruction sets are $I_0 = \{+_2, +_3, \ldots, +_6\}$, I_1

$= \{x_0, x_1, x_2, x_3, x_4, x_5, +_2, +_3\}$ and $I_2 = \{x_0, x_1, x_2, x_3, x_4, x_5\}$. The evolved hierarchical T-S fuzzy model is shown in Fig.3.26 (a). The output of the evolved model and the actual output for validation data set are shown in Fig.3.28, and the identification error is shown in Fig. 3.27.

Example 2

Let's the plant be given by

$$y(k+1) = \frac{y(k)}{1.5 + y^2(k)} - 0.3y(k-1) + 0.5u(k) \tag{3.82}$$

The input and output of system are $x(k) = [u(k), u(k-1), y(k), y(k-1)]$ and $y(k+1)$, respectively.

The training samples and the test data set are generated using the same sequence of random input signals as mentioned previously. The used instruction sets are $I_0 = \{+_2, +_3, \ldots, +_5\}$, $I_1 = \{x_0, x_1, x_2, x_3, +_2, x_3\}$ and $I_2 = \{x_0, x_1, x_2, x_3\}$.

The evolved hierarchical T-S fuzzy model is shown in Fig. 3.26 (b). The output of the evolved model and the actual output for validation data set are shown in Fig.3.30, and the identification error is shown in Fig. 3.29.

Example 3

Let's the plant be given by

$$y(k+1) = 1.752821y(k) - 0.818731y(k-1) + 0.011698u(k) + 0.010942\frac{u(k-1)}{1 + y^2(k-1)} \tag{3.83}$$

The input and output of system are $x(k) = [u(k), y(k), y(k-1)]$ and $y(k+1)$, respectively. The training samples and the test data set are generated using the same sequence of random input signals as mentioned previously. The used instruction sets are $I_0 = \{+_2, +_3, +_4\}$, $I_1 = \{x_0, x_1, x_2, +_2, +_3\}$ and $I_2 = \{x_0, x_1, x_2\}$. The evolved hierarchical T-S fuzzy model is shown in Fig. 3.26 (c). The output of the evolved model and the actual output for validation data set are shown in Fig. 3.32, and the identification error is shown in Fig. 3.31.

For the comparison, the result obtained by Elman nets, Jordan nets [21], the wavelet neural networks [23] and the proposed hierarchical T-S fuzzy model is shown in Table.

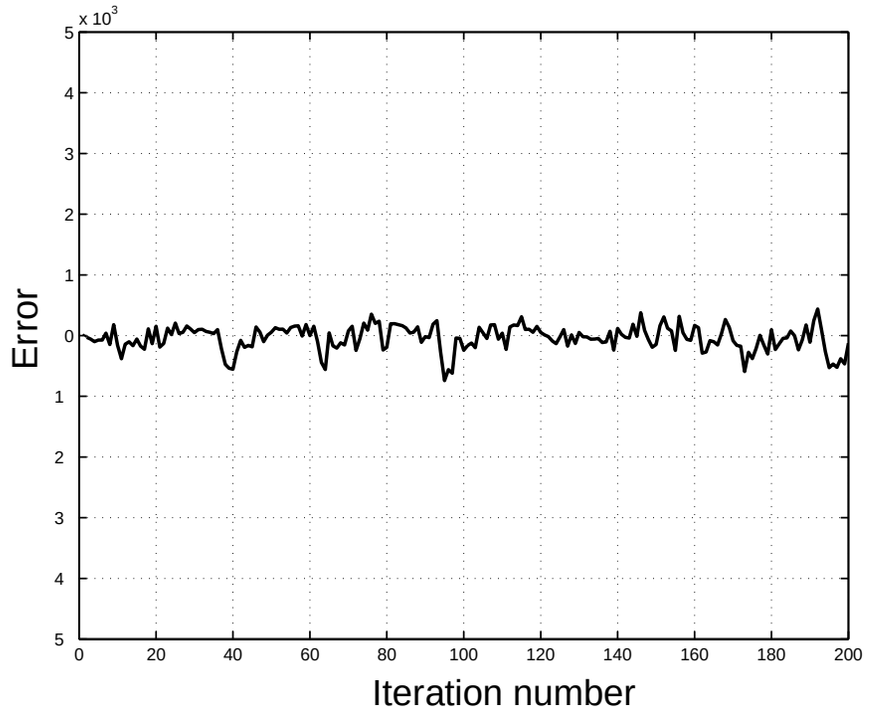


Figure 3.27: The test error (Plant 1)

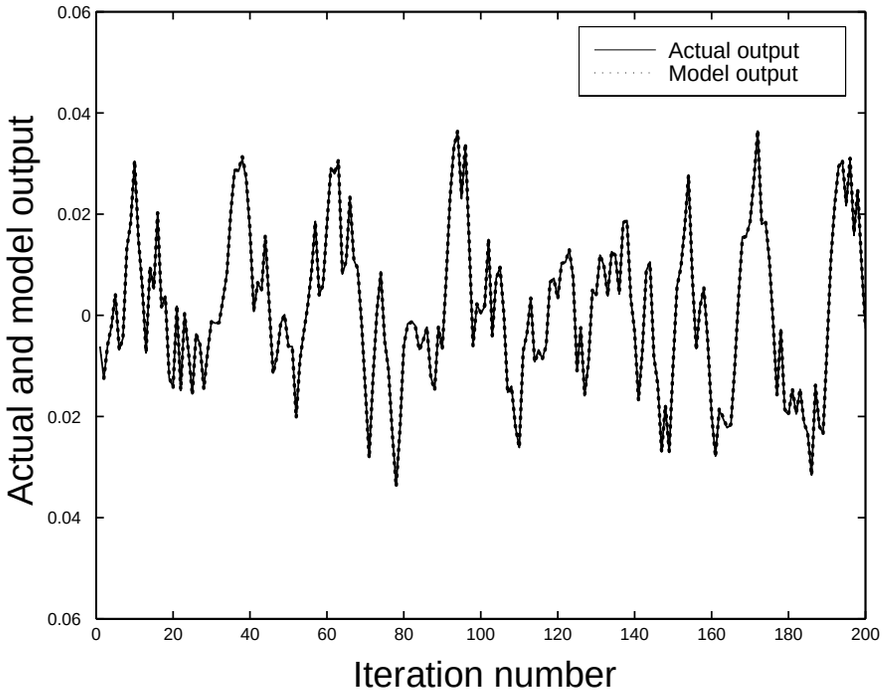


Figure 3.28: The actual and evolved outputs of the plant for the test data set (Plant 1)

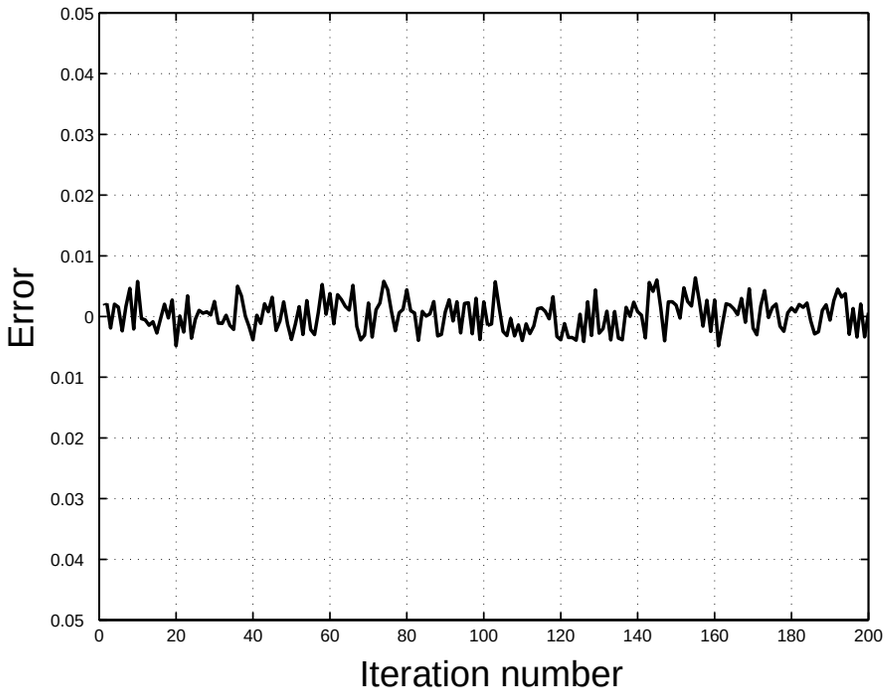


Figure 3.29: The test error (Plant 2)

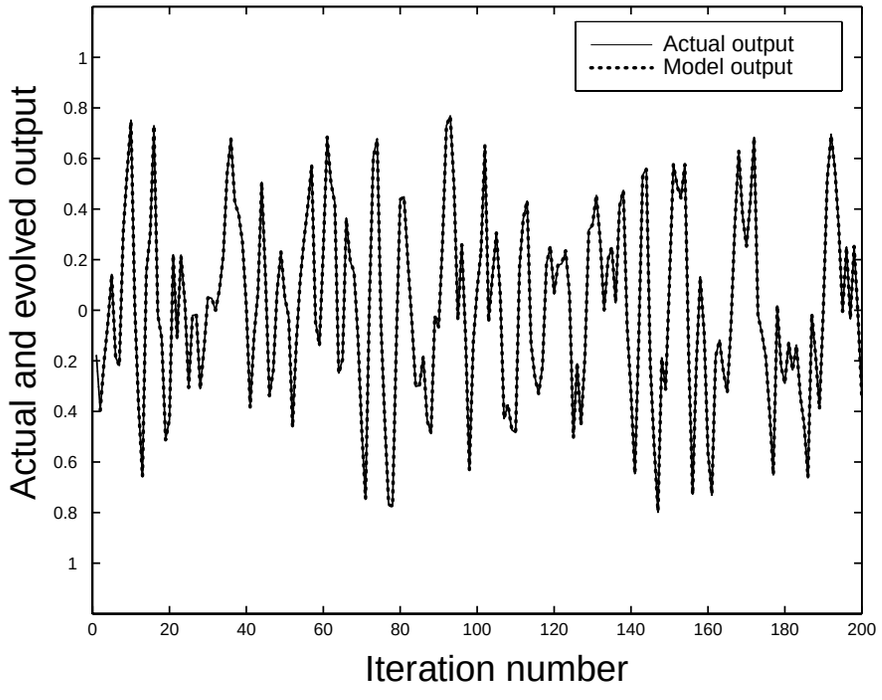


Figure 3.30: The actual and evolved outputs of the plant for the test data set (Plant 2)

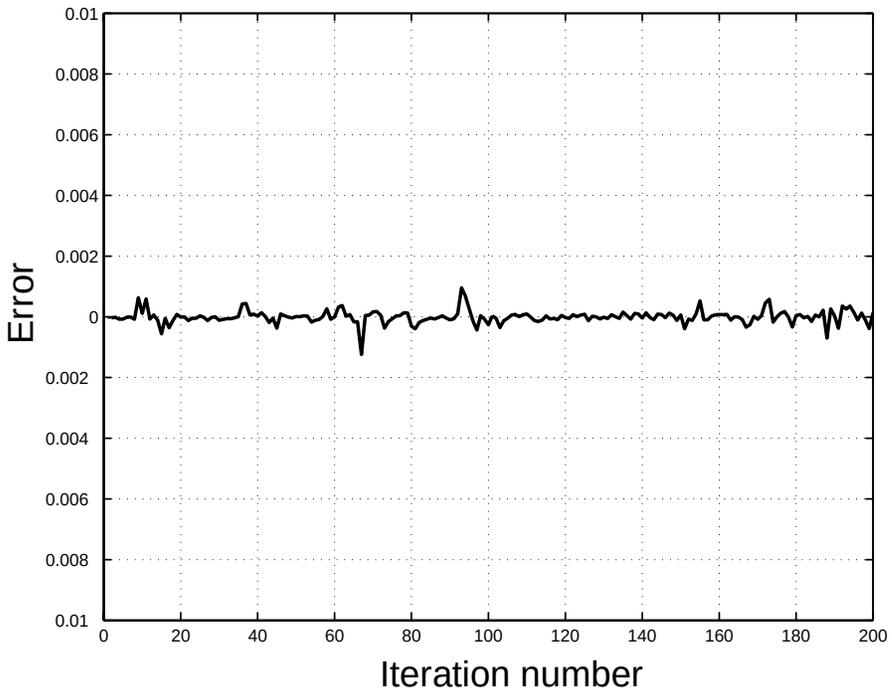


Figure 3.31: The test error (Plant 3)

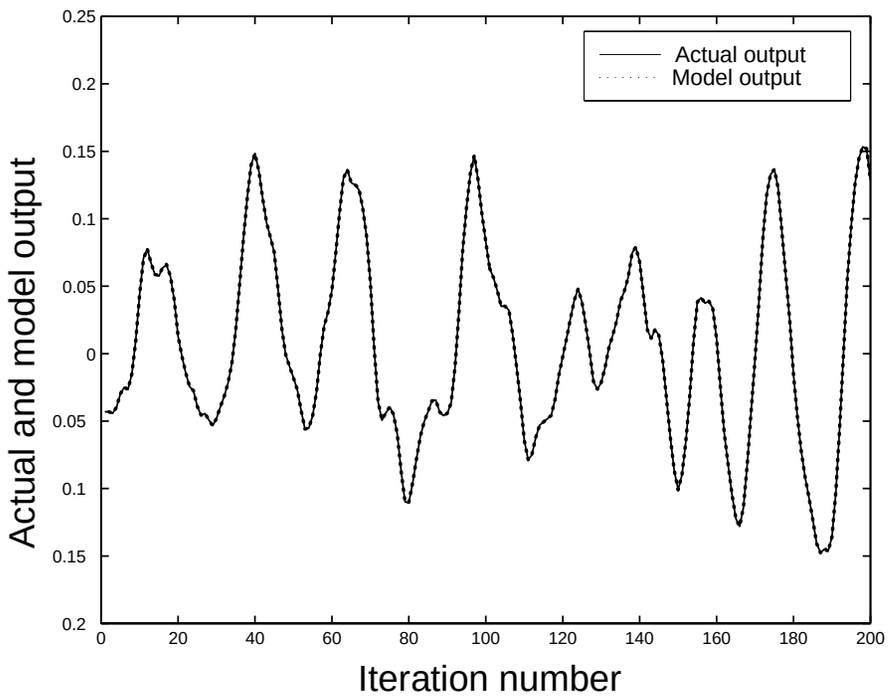


Figure 3.32: The actual and evolved outputs of the plant for the test data set (Plant 3)

Table 3.5: The comparison of the MSE values for modified Elman nets [21], modified Jordan nets [21], wavelet neural networks (WNN)[138] and hierarchical T-S fuzzy models (HFTS).

Example	Elman	Jordan	WNN	HFTS
1	0.0000548	0.0000492	0.000000459	0.0000000432
2	0.0004936	0.0003812	0.000002728	0.0000007065
3	-	-	0.000000732	0.0000000473

3.5.

From above simulation results, it can be seen that the proposed method works very well for generating a proper hierarchical T-S fuzzy models.

3.5.4 Section summary

The hierarchical architecture and inputs of the hierarchical fuzzy systems can be selected and optimized using MPIPE algorithm, and the parameters used in hierarchical T-S fuzzy models are optimized by a modified EP algorithm. Simulation results shown that the evolved hierarchical T-S fuzzy models are effective for the identification of nonlinear systems.

The simulations are constructed under the assumption of that the hierarchical layer is three and each input variable is divided into two fuzzy sets. But generalize the proposed method to the complex case is direct.

3.6 Conclusion in this Chapter

Based on a novel representation and calculation of the hybrid soft computing models, a unified framework for evolving the hybrid soft computing models is proposed in this chapter. In this flexible unified framework, various hybrid soft computing models and various parameter tuning strategies can be combined in order to find a proper soft computing model efficiently.

Four kinds of hybrid soft computing models have been evolved successfully by using the proposed method. The key points of this technique are those as follows:

- Almost of all the soft computing models can be represented and calculated by the type constrained sparse tree with pre-specified data structure which is attached to the node of the tree.

- In this sense, the soft computing models are created automatically not pre-designed, therefore, the difficulties in determining of the architecture of soft computing models can be avoided to some extend.

- It is also very important to mention that based on this idea the architecture and parameters used in the hybrid soft computing models can be evolved and optimized by using proper learning algorithm, i.e., the architecture of the hybrid soft computing models can be evolved by MPIPE or genetic programming and the parameters can be optimized by many parameter optimization techniques.

Simulation results show that the proposed method works very well for the nonlinear system identification problems.

Chapter 4

Hybrid Soft Computing for Control

To *control* a system is to make it behavior in a desired manner [268]. How to express this desired behavior depends primarily on the task to be solved, but the dynamics of the system, the actuators, the measurement equipment, the available computational power, etc., influence the formulation of the desired behavior as well. Two basic types of control problems are those as follows:

- *Regulation problems*. The fundamental desired behavior is to keep the output of the system at a constant level regardless of the disturbances acting on the system.
- *Servo problems*. The fundamental desired behavior is to make the output of the system follow a reference trajectory closely.

As it is known that many control applications involve significant nonlinearities. Although linear control design methods can sometimes be applied to nonlinear systems over limited operating regions through the process of linearization, the level of performance desired in other applications requires that the nonlinearities be directly addressed in the control system design. The challenge of addressing nonlinearities during the control design process is further complicated when the description of the nonlinearities involves significant uncertainty. In such applications, the level of achievable performance may be enhanced using off- or on-line function approximation techniques to increase the accuracy of the model of the nonlinearities. So far, a number of off- or on-line function approximators, such as those discussed in Section 2 and 3, have been proposed under the framework of soft computing methodologies.

There are in general two approaches of approximation based control, named as direct control and indirect control. In the first approach, the approximator is used to estimate the unknown nonlinearities of the plant. Based on this estimate, the control law is computed by treating the estimates as if they were the true part of the unknown plant according to

the certainty equivalence principle. In the second approach, the approximator is used to estimate directly the unknown nonlinear controller functions.

In this chapter, some of soft computing based control schemes are investigated. The design principle and the performances of soft computing based control schemes are given in detail.

4.1 Soft Computing Model Based Control: Some Design Principles

4.1.1 First-Order Taylor Approximation Based Design

This scheme is a generalization of the method proposed in [265] in which the used approximator is a diagonal recurrent neural network. But other soft computing models can also be used in order to formulate a unified controller design framework.

Let a MISO nonlinear process be described by

$$y(k+1) = f(y(k), y(k-1), \ldots, y(k-n_y), u(k), u(k-1), \ldots, u(k-n_u)) \tag{4.1}$$

where $y(k+1)$ and $u(k) = [u_1(k), u_2(k), \ldots, u_m(k)]^T$ represent the process output and the process inputs, respectively, $f(\cdot)$ is a nonlinear function vector, and n_y and n_u are the process orders. Eq. (4.1) can be rewritten compactly as

$$y(k+1) = f(Y(k), u(k), U(k-1)) \tag{4.2}$$

where $Y(k)$ and $U(k-1)$ are the vectors consisting of the process outputs $y(k)$ to $y(k-n_y)$ and the process inputs $u(k-1)$ to $u(k-n_u)$, respectively.

The control objective is to determine $u(k)$ such that the following quadratic function, consisting of the one-step-ahead-tracking error and the norm of the change in the control inputs, is minimized:

$$J(u(k)) = |y^*(k+1) - y(k+1)|^2 + \lambda \parallel u(k) - u(k-1) \parallel^2 \tag{4.3}$$

where $y^*(k+1)$ is the process desired output and λ is a weighting factor.

This is a nonlinear optimization problem and it is not easy to find the optimal solution for Eq. (4.3). One simple method for finding the solution can be described as follows. Eq. (4.2) can be expanded by the Taylor series with respect to the argument $u(k)$ at the

point $u(k-1)$, neglecting the higher-order terms. Then we have

$$y(K+1) \approx f(Y(k), u(k-1), U(k-1)) + \frac{\partial f^T}{\partial u(k)}|_{u(k)=u(k-1)}[u(k) - u(k-1)] \quad (4.4)$$

Substituting Eq.(4.4) into Eq. (4.3) and then minimizing Eq. (4.3), we can obtain

$$u(k) = u(k-1) + \frac{\partial f/\partial u(k)|_{u(k)=u(k-1)}}{\lambda + \|\partial f/\partial u(k)|_{u(k)=u(k-1)}\|^2}$$

$$\times (y^*(k+1) - f(Y(k), u(k-1), U(k-1))) \quad (4.5)$$

Remark 4.1: With a linear process, the control law (4.5) becomes the well-known generalized minimum variance control.

Remark 4.2: Note that the first-order Taylor approximation is only accurate if the higher-order terms of the Taylor expansion can be neglected.

Remark 4.3: The weighting factor λ may be chosen to provide a compromise between the closed-loop performance and stability. A small λ generally improves the performance but may result in stability problem. A large λ normally has better closed-loop stability, but more sluggish output response.

Remark 4.4: If $f(\cdot)$ is unknown, soft computing models can be used to approximate the function off-line or on-line. Therefore, a soft computing based control scheme can be formulated by estimating the sensitivity function $\partial f/\partial u(k)$ and the quasi-one-step-ahead predictive output of $f(Y(k), u(k-1), U(k-1))$.

4.1.2 Linear Feedback Design After Cancellation of Nonlinearities

This design scheme is a generalization of the method proposed in [266], in which the chaotic dynamical system is decomposed into a sum of linear and nonlinear part, and a radial basis function network is used to approximate the nonlinear part of the system. The resulting system then dominated by the linear system, therefore, linear controller design methods can be easily used for controlling the systems. We argue that it is valuable to design and compare the hybrid soft computing models based design method in order to formulate a unified controller design framework finally.

Consider a dynamic system of the form

$$\dot{x} = f(x) = f_L(x) + f_N(x) \quad (4.6)$$

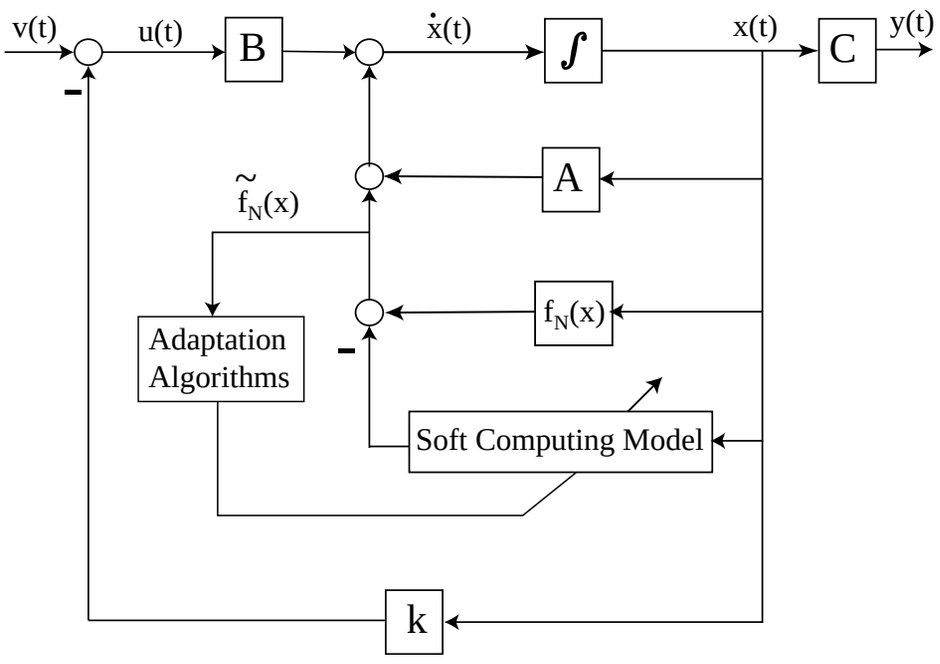


Figure 4.1: A linear state feedback controller in which a soft computing model is used to approximate the nonlinear part of dynamic system

where $f_L(x) = Ax$ is the linear part and $f_N(x)$ is the nonlinear part of the system dynamics $f(x)$, respectively. In Eq. (4.6), $f_L(s)$ is assumed to be known so that the controller for it can be designed and $f_N(x)$ is assumed unknown but the inputs and outputs of it can be measured.

Assume that the $f_N(x)$ can be approximated by a soft computing model $\hat{f}_N(x)$ with the approximation error $\tilde{f}_N(x) = f_N(x) - \hat{f}_N(x)$. If the approximator $\hat{f}_N(x)$ is subtracted from system (4.6), then is result in

$$\dot{x} = Ax + f_N(x) - \hat{f}_N(x) = Ax + \tilde{f}_N(x) \tag{4.7}$$

If the approximation error identically zero, then the resulting dynamics is purely linear for which a linear controller can be used

$$\dot{x} = Ax + bu(t) \tag{4.8}$$

where $\{A, b\}$ is assumed to be controllable and $u(t)$ is a scalar control input of the form

$$u(t) = -k^T x + v(t) \tag{4.9}$$

in which $v(t)$ is an external input (reference signal) and $k = [k_1, k_2, \ldots, k_n]$ is a constant

feedback gain vector to be designed. k can be obtained by the standard state feedback, i.e., pole-placement method. With this control law, the closed-loop control system is

$$\dot{x} = A_C x + bv(t) \tag{4.10}$$

where $A_C = A - bk^T$ is a stable matrix obtained by a suitable choice of the k. If the approximation error is not zero, then the closed-loop control system is

$$\dot{x} = A_C x + bv(t) + \tilde{f}_N(x) \tag{4.11}$$

Without loss generality, assume that the equilibrium point of the controlled system is the origin when $v(t) = 0$. Then, the following well-known stable condition result holds for Eq. (4.11).

Theorem The zero equilibrium of Eq. (4.11) is asymptotically stable if

$$\frac{\|\tilde{f}_N(x)\|}{\|x\|} \rightarrow 0 \; as \; \|x\| \rightarrow 0, \tag{4.12}$$

where $\|\cdot\|$ denotes the Euclidean norm of a vector.

Proof. Refer to Theorem 4.3 of [224].

Remark 4.5: In this control scheme, the plant to be controlled can be decomposed into a sum of a linear and a nonlinear part. This is not the always cases for arbitrary nonlinear systems.

Remark 4.6: If the nonlinearities can be cancelled by an approximation based soft computing model, a simple linear state-feedback controller can be devised to force the system response to a desired one. So, it is important to design a proper approximator, especially on-line trained soft computing model.

4.1.3 Sliding Mode Control Based Design

Consider a SISO nth-order nonlinear system described by following differential equation

$$x^{(n)} = f(x, \dot{x}, \ldots, x^{(n-1)}) + g(x, \dot{x}, \ldots, x^{(n-1)})u + d(t); \; y = x \tag{4.13}$$

where f and g are unknown continuous functions, d is a bounded unknown external disturbance; $u \in R$, $y \in R$ are the input and output of the system, respectively. Set $x_1 = x$, $x_2 = \dot{x}$, ..., $x_n = x^{(n-1)}$ and $x = [x_1, x_2, \ldots, x_n]^T$, then Eq.(4.13) can be expressed

as state and output equation as follows:

$$\dot{x} = Ax + b[f(x) + g(x)u + d(t)] \tag{4.14}$$

$$y = cx \tag{4.15}$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \tag{4.16}$$

$$b = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{4.17}$$

$$c = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix} \tag{4.18}$$

The control problem is to design an adaptive control law such that the output $y(t)$ of the closed-loop system can asymptotically track a desired signal $y_d(t)$, i.e.,

$$\lim_{t \to \infty} [y(t) - y_d(t)] = 0 \tag{4.19}$$

In order for system (4.14) to be controllable following assumptions are usually needed.
Assumption 1: $g(x) \geq \alpha > 0$, $\forall x \subset Q \subset R^n$, where Q is a neighborhood of zero.
Assumption 2: $|d(t)| \leq d_0$, $\forall t \geq 0$, where d_0 is regarded as an unknown constant.
Define the tracking error:

$$\epsilon(t) = y(t) - y_d(t) \tag{4.20}$$

and $e_i(t) = \epsilon^{(i-1)}(t)$, $1 \leq i \leq n$. Then $e(t) = [e_1, e_2, \dots, e_n]^T \in R^n$. It follows from (4.14)

that

$$\dot{e} = Ae + b[f(x) + g(x)u + d(t) - y_d^{(n)}] \tag{4.21}$$

Define the generalized error

$$s(t) = (\frac{d}{dt} + \lambda)^{n-1}\epsilon(t) = \Lambda^T e(t) \tag{4.22}$$

where $\lambda > 0$, $\Lambda = [\Lambda_1, \Lambda_2, \ldots, \Lambda_{n-1}, 1]$, and $\Lambda_j \in R$, $1 \leq j \leq n-1$, are determined once the parameter λ has been specified. Obviously, on the sliding surface, $s = 0$, the tracking error $\epsilon(t) \to 0$, as $t \to \infty$. Differentiation of s with respect to t, yields

$$\dot{s} = \Lambda^T \dot{e} = \bar{\Lambda}^T e + (f + gu + d - y_d^{(n)}) = f - w + d + gu \tag{4.23}$$

where $\bar{\Lambda} = [0, \Lambda_1, \Lambda_2, \ldots, \Lambda_{n-1}]^T$, $w = -\bar{\Lambda}^T e + y_d^{(n)}$.

If f and g were known, and the external disturbance d were zero, the equivalent control

$$u = -\frac{f - w}{g} \tag{4.24}$$

would render $\dot{s} = 0$. However, since f and g are unknown, some proper approximators, $\hat{f}$, $\hat{g}$, for f and g, respectively, should be employed and the certainty equivalent control law would be the form

$$\hat{u} = -\frac{\hat{f} - w}{\hat{g}}. \tag{4.25}$$

Remark 4.7: In this design scheme, the main problems are that how to select a proper approximator, tuning the parameters of the approximator off-line or on-line, and analysis the stability of the closed-loop system.

Remark 4.8: So far, a number of approximation based sliding mode controller have been successfully designed. The most used approximators are neural networks [225]-[228] and fuzzy logic systems [229]-[231].

4.1.4 Backstepping Design

Consider a strict-feedback nonlinear system in the following form:

$$\dot{x_i} = f_i(x_1, x_2, \ldots, x_i) + g_i(x_1, x_2, \ldots, x_i)x_{i+1}, 1 \leq i \leq n-1$$
$$\dot{x_n} = f_n(x) + g_n(x)u$$

$$y = x_1 \tag{4.26}$$

where $x = [x_1, x_2, \ldots, x_n] \in R^n$, $u \in R$, $y \in R$ are the state variables, system input and output, respectively; $f_i(\cdot)$ and $g_i(\cdot)$, $i = 1, 2, \ldots, n$ are unknown smooth functions.

The control objective is to design an adaptive controller for system (4.26) such that the output y follows a desired trajectory y_d, while all signals in the closed-loop systems are bounded. From now onwards, the following assumptions are satisfied.

Assumption 1. The signs of $g_i(\mathbf{x_i})$ are known, and there exist constants g_{i0}, and known smooth functions $G_i(\mathbf{x_i})$ such that $G_i(\mathbf{x_i}) \geq |g_i(\mathbf{x_i})| \geq g_{i0}$, $\forall \mathbf{x_i} \in R^i$, where $\mathbf{x_i} = [x_1, x_2, \ldots, x_i]^T$.

Assumption 2. The desired trajectory vectors $\mathbf{x}_{di}$ with $i = 2, \ldots, n+1$ are continuous and available, where $\mathbf{x}_{d(i+1)} = [y_d, \dot{y_d}, \ldots, y_d^{(i)}]^T$.

Adaptive control for first-order systems:

Let us firstly consider the Eq.(4.26) when $i = 1$, i,e.,

$$\dot{x} = f_1(x_1) + g_1(x_1)u_1 \tag{4.27}$$

with u_1 as control input, and define $z_1 = x_1 - y_d$, $\beta_1(x_1) = G_1(x_1)/g_1(x_1)$ and a smooth scalar function

$$V_{z1} = \int_0^{z1} \sigma\beta_1(\sigma + y_d)d\sigma \tag{4.28}$$

Let $\sigma = \theta z_1$, the V_{z1} can be rewritten as $V_{z1} = z_1^2 \int_0^1 \theta\beta_1(\theta z_1 + y_d)d\theta$. According to Assumption 1, the following condition holds,

$$1 \leq \beta_1(\theta z_1 + y_d) \leq G_1(\theta z_1 + y_d)/g_{10} \tag{4.29}$$

Then,

$$\frac{z_1^2}{2} \leq V_{z1} \leq \frac{z_1^2}{g_{10}} \int_0^1 \theta G_1(\theta z_1 + y_d)d\theta \tag{4.30}$$

This means that V_{z1} is a positive-define function with respect to z_1, therefore, it can be used as a candidate Lyapunov function. Furthermore, if the control signal u_1 is selected as follows

$$u_1^* = \frac{1}{G_1(x_1)}(-k(t) - h_1(Z_1)) \tag{4.31}$$

where the smooth function $h_1(Z_1) = \beta_1(x_1)f_1(x_1) - \dot{y_d}\int_0^1 \beta_1(\theta z_1 + y_d)d\theta$, $Z_1 = [x_1, y_d, \dot{y_d}]^T$,

therefore, we have

$$\dot{V}_{z1} = z_1 \left[G_1(x_1)u_1 + \beta_1(x_1)f_1(x_1) - \dot{y_d} \int_0^1 \beta_1(\theta z_1 + y_d)d\theta \right]. \tag{4.32}$$

Substituting $u_1 = u_1^*$ into Eq.(4.32), we obtain $\dot{V}_{z1} = -k(t)z_1^2 \leq -k * z_1^2 \leq 0$. Hence, V_{z1} is a Lyapunov function and the tracking error $z_1 \to 0$ as $t \to \infty$ asymptotically.

If the functions $f_1(x_1)$ and $g_1(x_1)$ are unknown, the desired controller u_1^* is not available due to the unknown function $h_1(Z_1)$. However, $h_1(Z_1)$ is a smooth function and may be approximated by a soft computing model. Therefore, a soft computing model based controller can be realized as follows

$$u_1 = \frac{1}{G_1(x_1)}[-k_1(t)z_1 - SOFT_1] \tag{4.33}$$

By carefully selecting of the parameter tuning strategy of the soft computing model, above adaptive controller can be implemented with guaranteed stability (see, for e.g., a used soft computing model is a NN and related weight tuning formula in [209]).

In order to control the higher-order strict-feedback nonlinear system (4.26), a back-stepping design scheme can be used as follows.

Backstepping Design:

Step 1: The system (4.26) for $i = 1$ is given by

$$\dot{x} = f_1(x_1) + g_1(x_1)x_2 \tag{4.34}$$

x_2 can be viewed as a virtual control input, and define error variable $z_2 = x_2 - \alpha_1$ with $\alpha_1 = u_1$ defined in Eq.(4.33), then

$$\dot{z}_1 = f_1(x_1) + g_1(x_1)(z_2 + \alpha_1) - \dot{y}_d \tag{4.35}$$

Taking V_{z1} given in Eq.(4.28) as a Lyapunov function candidate, then we have

$$\dot{V}_{z1} = z_1[G_1(x_1)(z_2 + \alpha_1) + h_1(Z_1)]. \tag{4.36}$$

Step 2: The system (4.26) for $i = 2$ is given by

$$\dot{x}_2 = f_2(\mathbf{x_2}) + g_x(\mathbf{x_2})x_3. \tag{4.37}$$

Again, by viewing x_3 as a virtual control, we may design a control input α_2 for Eq.(4.37).

Define $z_3 = x_3 - \alpha_2$, then

$$\dot{z}_2 = \dot{x}_2 - \dot{\alpha}_1 = f_2(\mathbf{x_2}) + g_2(\mathbf{x_2})(z_3 + \alpha_2) - \dot{\alpha}_1 \tag{4.38}$$

Choosing a Lyapunov candidate

$$V_{s2} = V_{z1} + \int_0^{z_2} \sigma\beta_2(\mathbf{x_1}, \sigma + \alpha_1)d\sigma \tag{4.39}$$

Its time derivative becoms

$$\dot{V}_{s2} = \dot{V}_{s1} + z_2\beta_2(\mathbf{x_2})\dot{z}_2 + \int_0^{z_2} \sigma \left[\frac{\partial\beta_2(\mathbf{x_1}, \sigma + \alpha_1)}{\partial \mathbf{x_1}}\dot{\mathbf{x}}_1 + \frac{\partial\beta_2(\mathbf{x}_1, \sigma + \alpha_1)}{\partial\alpha_1}\dot{\alpha}_1\right] d\sigma \tag{4.40}$$

Selecting the controller α_2 as

$$\alpha_2 = \frac{1}{G_2(x_2)}[-G_1(x_1) - k_2(t)z_2 - SOFT_2] \tag{4.41}$$

with an appropriate parameter tuning strategy of the soft computing model, the system (4.37) can be controlled with guaranteed stability.

Step n: For system (4.26), consider $z_n = x_n - \alpha_{n-1}$, we have

$$\dot{z}_n = \dot{x}_n - \dot{\alpha}_{n-1} = f_n(x) + g_n(x)u - \dot{\alpha}_{n-1}. \tag{4.42}$$

Taking the following Lyapunov function candidate

$$V_{sn} = V_{s(n-1)} + \int_0^{z_n} \sigma\beta_n(\mathbf{x_{n-1}}, \sigma + \alpha_{n-1})d\sigma \tag{4.43}$$

Its time derivative becoms

$$\begin{aligned}\dot{V}_{sn} &= \dot{V}_{s(n-1)} + z_n\beta_n(\mathbf{x}_n)\dot{z}_n \\ &+ \int_0^{z_n} \sigma \left[\frac{\partial\beta_n(\mathbf{x}_{n-1}, \sigma + \alpha_{n-1})}{\partial \mathbf{x}_{n-1}}\dot{\mathbf{x}}_{n-1} + \frac{\partial\beta_n(\mathbf{x}_{n-1}, \sigma + \alpha_{n-1})}{\partial\alpha_{n-1}}\dot{\alpha}_{n-1}\right] d\sigma\end{aligned} \tag{4.44}$$

Selecting the controller u as

$$u = \frac{1}{G_n(x)}[-G_{n-1}(\mathbf{x}_{n-1}) - k_n(t)z_n - SOFT_n] \tag{4.45}$$

with an appropriate parameter tuning strategy of the soft computing model, the system (4.26) can be controlled with guaranteed stability.

Remark 4.9: It can be seen that it is important for designing of appropriate Lyapunov function in order to guarantee the stability of the closed-loop system. In fact, the Lyapunov function is not unique for a given nonlinear system and therefore finding a proper Lyapunov function becomes a challenging problem in the backstepping design.

Remark 4.10: In each step of the backstepping design, a soft computing model is used for approximating nonlinear function, so it is important for selecting an appropriate parameter tuning strategy.

Remark 4.11: So far, the used soft computing models in approximation based backstepping controller design include neural networks [209][210][264] and fuzzy systems [232].

4.2 Nonlinear System Identification and Control by using PIPE Algorithm

4.2.1 Introduction

For the identification and control problem of nonlinear dynamic system, the evolutionary identification/control of nonlinear system has received much attention during the last few yeas. In these researches, a novel method in which the optimal control of nonlinear system can be directly derived by using tree-structure-based evolutionary algorithm (TSEA) was proposed. Andrew proposed a system identification method by using genetic programming [171] (GP). Howley used GP to get a sub-optimal control law for simulated specific spacecraft attitude maneuvers [172]. Dracopoulos used GP to derive an attitude control law to de-tumble a spinning satellite and discussed the stability by the Lyapunov method [173]. Dominic et al. used GP to design a discrete-time dynamic controller of chemical process that offers similar performance as PID controller [174]. Chellapilla used tree-structure-based evolutionary programs to derive nonlinear control laws for broom balancing problem and backing up a truck-and-trailer problem [175]. Angeline recently proposed multiple interacting program algorithm, which involved the simultaneous evolution of a set of equations represented as parse tree, and applied it to system identification of nonlinear system [176].

All these researches based on one key point, that is TSEA can offer an ideal candidate for system identification and controller design, by the direct matching of individual structures and control rules. However, there are some remained problems to be solved, e.g., how to design a TSEA identifier/controller, the stability, robustness and generalization ability of TSEA controller, and so on.

In this Section, based on the Probabilistic Incremental Program Evolution algorithm,

new methods for identification and control of nonlinear discrete-time system are proposed. Firstly, the PIPE Emulator is generated using PIPE algorithm, and secondly PIPE Emulator based inverse controller is designed.

4.2.2 PIPE Identifier

We discuss the possibility and effectiveness of identification of nonlinear system by using PIPE algorithm. Every individual of the PIPE population represents a candidate model of nonlinear system to be identified. The best individual will be found in which the sum of the error between plant and model outputs is minimum according to the probabilistic incremental learning algorithm. The advantage of this method is that the solution of PIPE Identifier is of symbolic expression and easily applied to the controller design of nonlinear system.

Consider a nonlinear SISO plant described by equation of the form:

$$\begin{aligned} x(k+1) &= F[x(k), u(k)] \\ y(k) &= O[x(k)] \end{aligned} \tag{4.46}$$

where $u(\cdot)$, $x(\cdot)$ and $y(\cdot)$ are discrete time sequence, and satisfy the stringent observability condition, i.e., the state of the system can be reconstructed from the I/O data. The plant can also be represented as:

$$y(k+1) = f[y(k), \cdots, y(k-n+1); u(k), \cdots, u(k-m+1)] \tag{4.47}$$

where $[u(k), y(k)]$ represents the I/O pair of plant at time k and m $\leq$ n.

The architecture of PIPE-based identification system is shown in Fig. 4.2. The inputs of PIPE model are present input and output signals, previous input and output signals. The output of the induced plant model is compared with the actual output of the plant to determine the error for the model. The error is used to update the plant model's performance by PIPE algorithm.

Numerical Experiment

For the identification and control of nonlinear system, a benchmark problem is here considered which was also discussed in [177] [178]. In [177], a neural network based scheme is proposed for the identification and control of the plant. In [178], a recurrent fuzzy neural network is employed for same purpose. The plant is given by the following

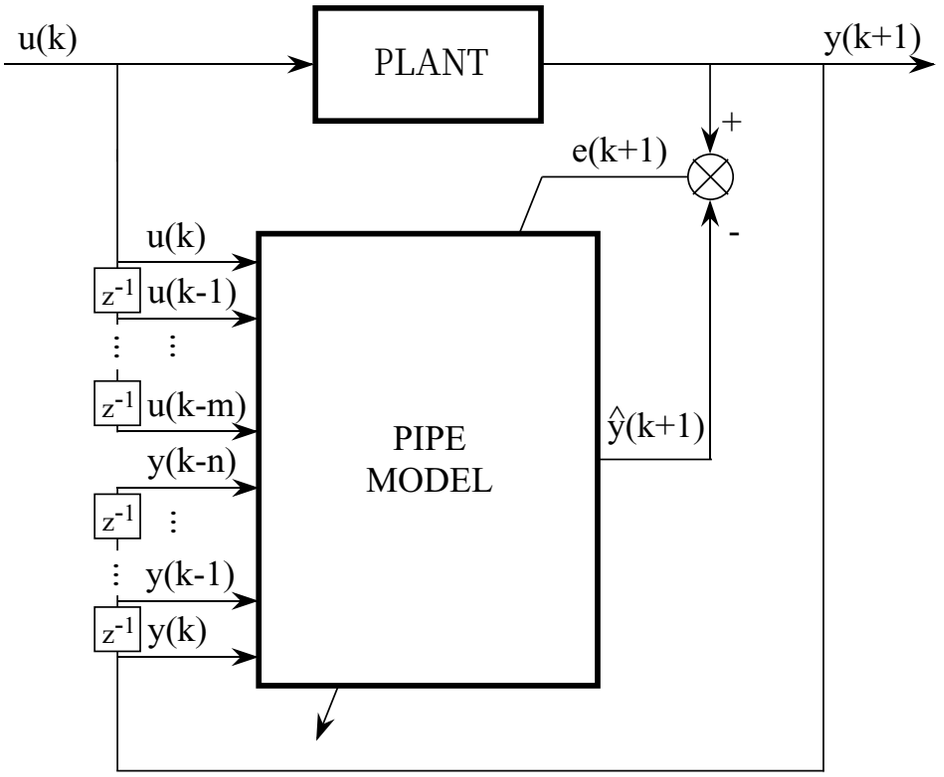


Figure 4.2: PIPE Emulator/Identifier

equation:

$$y\left(k+1\right)=f\left[y\left(k\right),y\left(k-1\right)\right]+u\left(k\right) \tag{4.48}$$

$$f\left[y\left(k\right),y\left(k-1\right)\right]=\frac{y\left(k\right)y\left(k-1\right)\left[y\left(k\right)+2.5\right]}{1+y^{2}\left(k\right)+y^{2}\left(k-1\right)} \tag{4.49}$$

The evolved plant is evaluated using the following signal function $u(k)$ for 200 units of time starting at $k=0$, with $y\left(-1\right)=0$, $y\left(0\right)=0$.

$$u(k)=sin\left(2k\pi/25\right) \tag{4.50}$$

The task specific language used for this experiment includes the function set ($+$, $-$, $*$, %, sin, cos), and the terminal set $(x0, x1, x2, R)$, where R denotes the generic random constant. The other control parameters of PIPE for this experiment are same as those in table 2.1.

For function approximation and/or nonlinear system identification, the usually used objective/fitness functions in neural network and evolutionary computation community can be summarized as follows:

(1) The mean square error (MSE)

$$Fitness = \frac{1}{N}\sum_{i=1}^{N}(y^i - y_p^i)^2 \tag{4.51}$$

(2) The root mean square error (RMSE)

$$Fitness = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y^i - y_p^i)^2} \tag{4.52}$$

(3) The sum of absolute error (SAE)

$$Fitness = \sum_{i=1}^{N}|y^i - y_p^i| \tag{4.53}$$

(4) The mean absolute error (MAE)

$$Fitness = \frac{1}{N}\sum_{i=1}^{N}|y^i - y_p^i| \tag{4.54}$$

(5) The sum of relative error (SRE)

$$Fitness = \sum_{i=1}^{N}\frac{|y^i - y_p^i|}{y^i} \tag{4.55}$$

(6) The mean relative error (MRE)

$$Fitness = \frac{1}{N}\sum_{i=1}^{N}\frac{|y^i - y_p^i|}{y^i} \tag{4.56}$$

where y^i is the i-th model output and y_p^i is the i-th plant output. N is the number of data in the training data set. Up to date, there is no general solution in selecting of the fitness function for an identification problem. A number of experiments have been shown that the smaller error for training data set may have not good generalization for validation data set. Therefore, additional terms are usually added to above fitness function in order to cope with the generalization problem, i.e., plus a term named the norm of the weight vector in the neural network training can results in a good generalization of the neural network. In our current experiments, due to the ability of PIPE itself that among programs with equal fitness values, smaller ones are always selected, the additional term for generalization purpose is not used here. The used fitness function is the sum of absolute error which is selected according to our experiments in the use of PIPE for

function approximation.

The following symbolic expression was returned as the best individual found in the run after the 14166 generation with fitness 8.651981:

$$g(x0,x1,x2) = (-(+(+(*(cos(-(sin0.377678)0.083342))x1)(-(-$$
$$x2x2)x0))x1)(\%(sin(*(*(*(-(-x1x2)(+(-x10.583063)x1))(sin$$
$$(*(\%(sin(*x2(\%(sinx0)(-x2(*(*(cos0.114430)x0)(*(*(+x0(cos$$
$$0.275634))(*(+(cos(+0.445955(-0.340801(cos0.005139))))x0)$$
$$0.817219))x1))))))(+x0x0))x2)))(cos(sin(\%x2(+x2(cos(+(cos$$
$$x0)(\%x0(cos(-(*x2x2)x0))))))))))(cos(*(+(\%(+(sinx0)x2)x1)$$
$$(\%(*x0(-(*(*(sin0.911381)(+(*(cosx1)(+(cos(\%x1x1))(*(-x1$$
$$x1)x0)))(*(+0.723960(sinx2))(*(*(cosx0)0.344123)x2))))(*($$
$$*(*(\%(cos(cosx0))0.230661)x1)x1)0.963181))(-x1x2)))(cosx2$$
$$)))(-x1(+x1x0))))))(cos0.573655))) \tag{4.57}$$

where x0, x1, x2 and g(x0,x1,x2) represents $y(k-1)$, $y(k)$, $u(k)$, and $y(k+1)$, respectively. In order to test the generalization ability of PIPE Emulator/Identifier, the following equation is used to create the un-trained sample set:

$$u(k) = sin(k\pi/25) + 0.1sin(k\pi/32) + 0.1sin(k\pi/10) \tag{4.58}$$

The responses of the evolved plant and real plant for same input signal and same time duration is showed in Fig. 4.3 where the dotted and solid line denote the evolved and desired outputs of the plant. Fig. 4.4 shows the desired output (solid line) and evolved output (dotted line) of the plant for the un-trained sample set. From the simulation results we can see that the PIPE Identifier has better ability to approximate the nonlinear plant.

4.2.3 PIPE based controller design

The most common scheme used in GP-based control is that of inverse controller. In this case, GP was used to evolving a directly control law of nonlinear systems. But it is difficult to design a better fitness function that should reflect the characteristics of nonlinear system, and the general method to do this has not been found yet. In this paper,

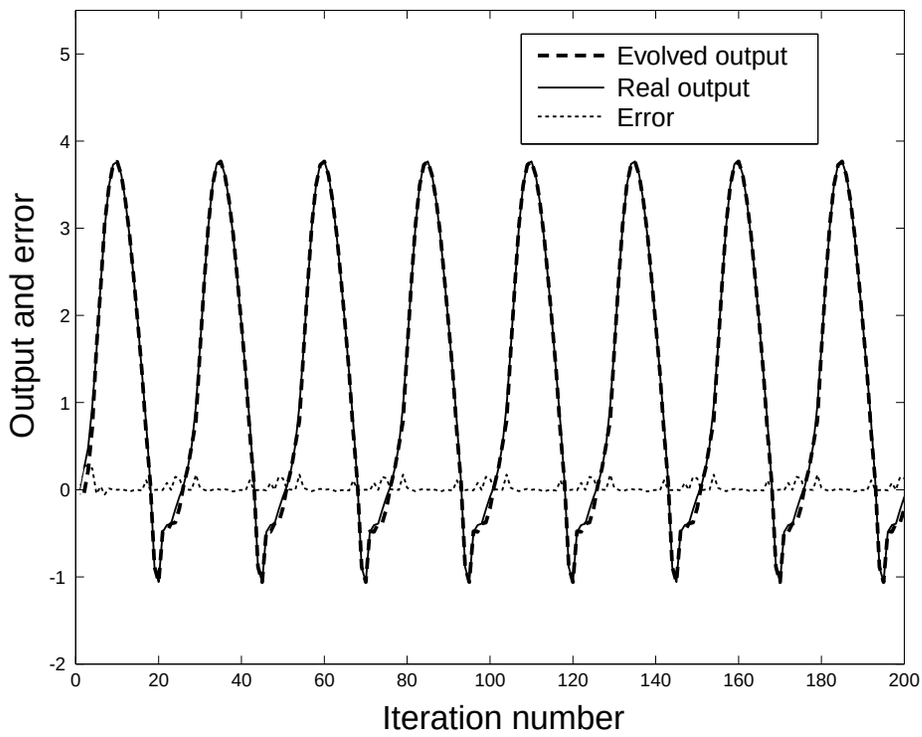


Figure 4.3: The desired plant output and the evolved plant output for training data set

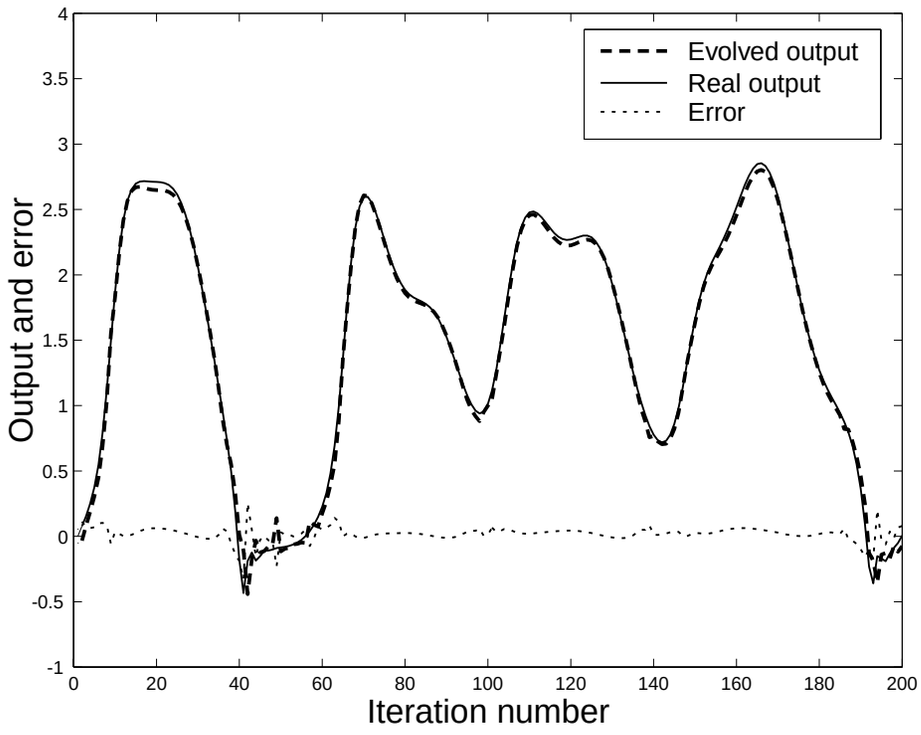


Figure 4.4: The desired plant output and the evolved plant output for un-trained samples

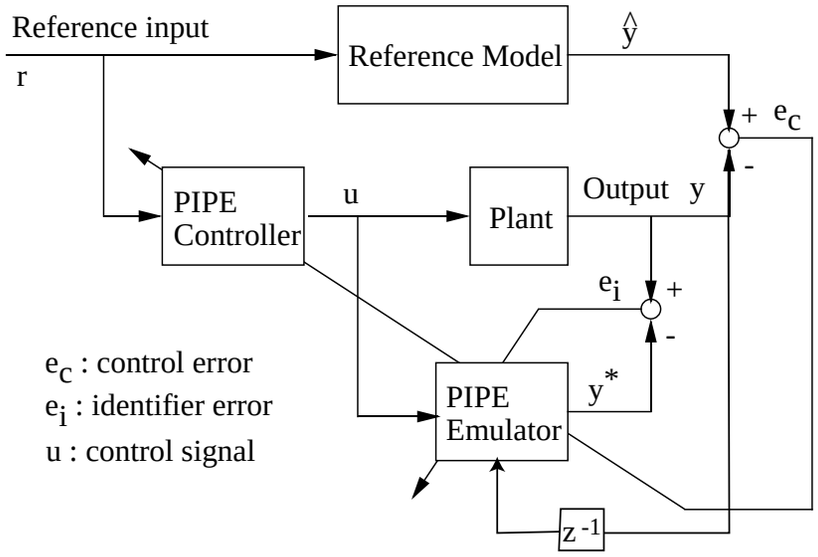


Figure 4.5: PIPE Emulator-Based Control System

we will propose a new inverse dynamic controller based on PIPE algorithm. The general architecture of the proposed PIPE-based control system is shown in Fig. 4.5. First, the approximation model of nonlinear system is constructed by using PIPE algorithm and is called a PIPE Emulator. PIPE Emulator is offline trained by the forward plant model. Secondly, PIPE controller is designed by two ways

- by the mathematical inverse of the plant equation.
- by designing the inverse plant model using PIPE algorithm. Through as the example, we illustrate the first case in the following.

Example: The plant to be controlled is given by the Eq.(4.48) and Eq.(4.49), where the function $f[\cdot]$ is assumed to be unknown, and $u(k)$ is additive. A PIPE model of this plant has been established as Eq.(4.57). The aim of control is to determine a controller $u(k)$, based on the PIPE Emulator, such that the output of the closed-loop system may follow the one of the reference model:

$$y_m(k+1) = 0.6y_m(k) + 0.2y_m(k-1) + r(k) \tag{4.59}$$

where $r(k) = sin(2k\pi/25)$. That is, the error $e(k) = y(k) - y_m(k)$ must converge to zero as k goes to infinity. If the function $f[\cdot]$ is exactly known, we can construct a controller based on the principle of inverse control as

$$u(k) = -f[y(k), y(k-1)] + 0.6y(k) + 0.2y(k-1) + r(k) \tag{4.60}$$

which, when applied to (4.48), results in

$$y(k+1) = 0.6y(k) + 0.2y(k-1) + r(k) \tag{4.61}$$

Combining (4.59) with (4.61), we have

$$e(k+1) = 0.6e(k) + 0.2e(k-1) \tag{4.62}$$

thus, we have $lim_{k\to\infty} e(k) = 0$. However, since $f[\cdot]$ is unknown, the controller (4.60) cannot be implemented. To cope with this, we replace the $f[\cdot]$ in (4.48) by a PIPE emulator: we use the following controller

$$u(k) = -g[y(k), y(k-1)] + 0.6y(k) + 0.2y(k-1) + r(k) \tag{4.63}$$

where $g[\cdot]$ is of the form of (4.57). Then we can get the equation of closed-loop system as

$$y(k+1) = f[y(k), y(k-1)] - g[y(k), y(k-1)]$$

$$+0.6y(k) + 0.2y(k-1) + r(k) \tag{4.64}$$

The overall structure of PIPE Emulator-based control system is shown in Fig. 4.6. In the figure, r, u, f, g, A_m, y, y_m, e_i and e_c denote the reference input, control input, nonlinear part of plant, an estimation of f, coefficients of reference model, plant output, output of reference model, identification error and control error, respectively. Fig. 4.7 shows a comparison of the output of $y(k+1)$ of the closed-loop system with the reference model output $y_m(k+1)$. From the simulation result, it can be seen that the proposed method works very well in producing accurate tracking control using the above scheme.

4.2.4 Section summary

This Section discussed the applicability of PIPE algorithm to the identification and control of nonlinear system. The evolved symbolic expression is competitive with those obtained using neural network and/or multiple interacting program algorithm, and the symbolic expression model of nonlinear system may be amenable to analysis than neural network model. But same as GP, the symbolic expression obtained by PIPE is usually redundant and very long in length. This problem can be addressed by selecting the proper architecture of sparse tree and instruction set in the further research.

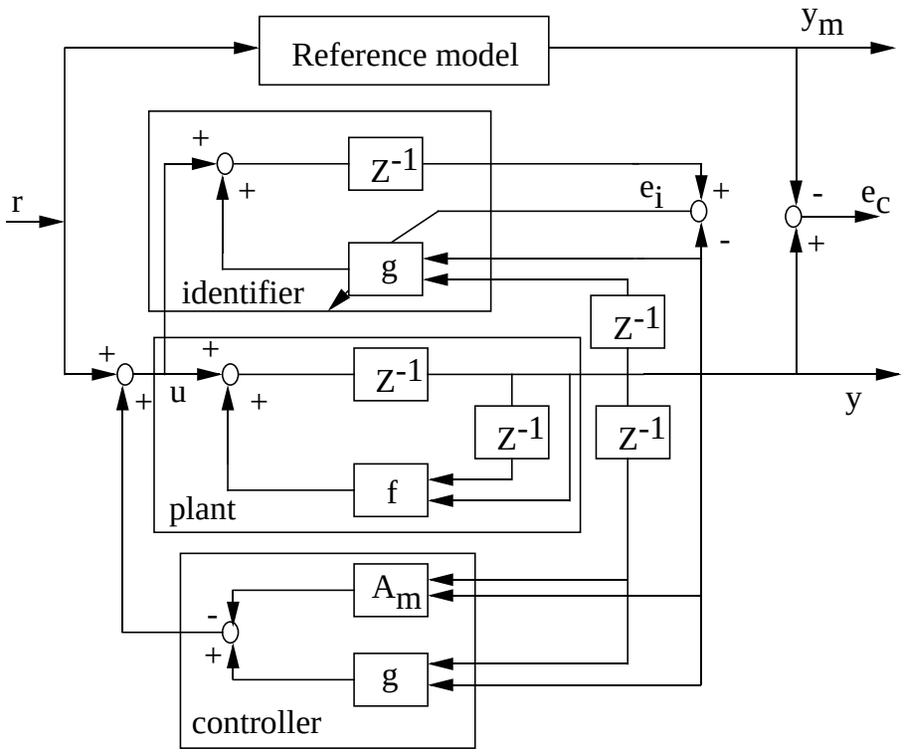


Figure 4.6: PIPE Emulator-based control system

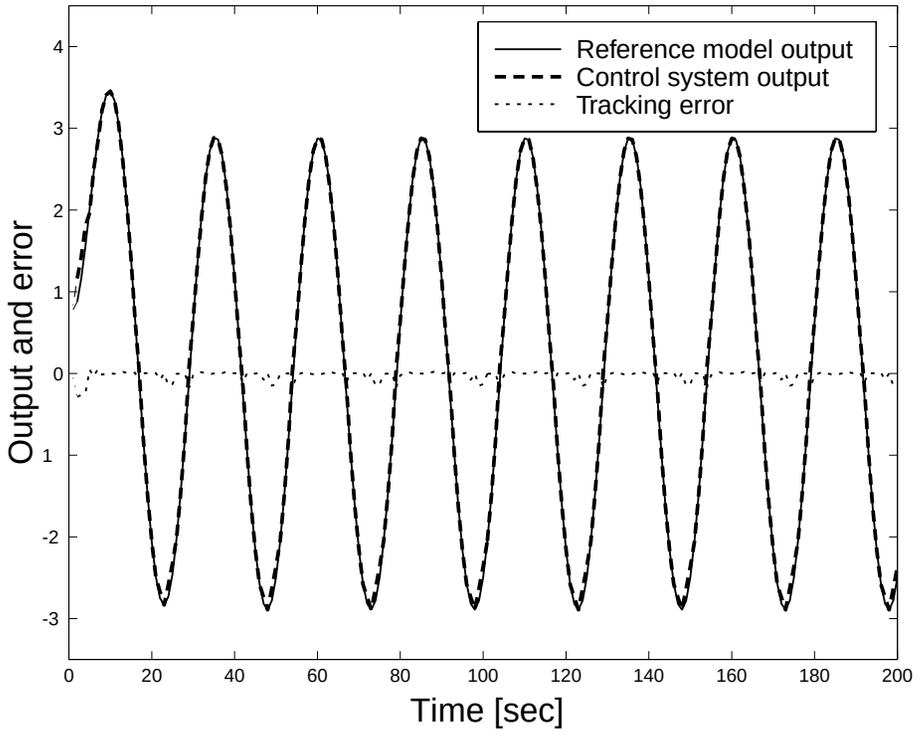


Figure 4.7: The Desired Plant Output and the Evolved Plant Output

In addition, it is easy to insert the traditional control methods into the PIPE Emulator/Identifier based control system. Our future work will concentrate on improving the performance of PIPE controller by embedding PID technique in order to apply easily the proposed method to the practical control problems.

4.3 Approximation Based Direct Control

4.3.1 Adaptive Soft Computing Model Based Inverse Control

Design Principle

Basically, the adaptive soft computing based inverse controllers can be categorized into two main schemes, i.e., direct approach and indirect approach. In direct soft computing based inverse control scheme, there are two kinds of learning schemes known as general learning depicted in Fig. 4.8 and specialized learning depicted in Fig. 4.9. The soft computing based controllers produce the control inputs through learning the inverse mapping relationship of the nonlinear plant or minimum some cost function. Explained in brief, the basic principle is as follows.

Assuming that the system to be controlled can be described by

$$y(t+1) = g[y(t), \ldots, y(t-n+1), u(t), \ldots, u(t-m)] \tag{4.65}$$

The desired soft computing model is then the one that isolates the most recent control input, $u(t)$,

$$\hat{u}(t) = \hat{g}^{-1}[y(t+1), \ldots, y(t+n-1), u(t), \ldots, u(t-m)] \tag{4.66}$$

Assuming such a soft computing model has been somehow obtained, it can be used for controlling the system by substituting the output at time $t+1$ by the desired output, the reference, $r(t+1)$. If the soft computing model represents the exact inverse, the control input produced by it will thus drive the system output at time $t+1$ to $r(t+1)$.

Generalized Training

The most straightforward way of training a soft computing model as the inverse of a system is to approach the problem as a system identification problem. In doing so, the soft computing model can be trained off-line by using any of the training methods presented in the previous chapters of the thesis. Note that soft computing model based

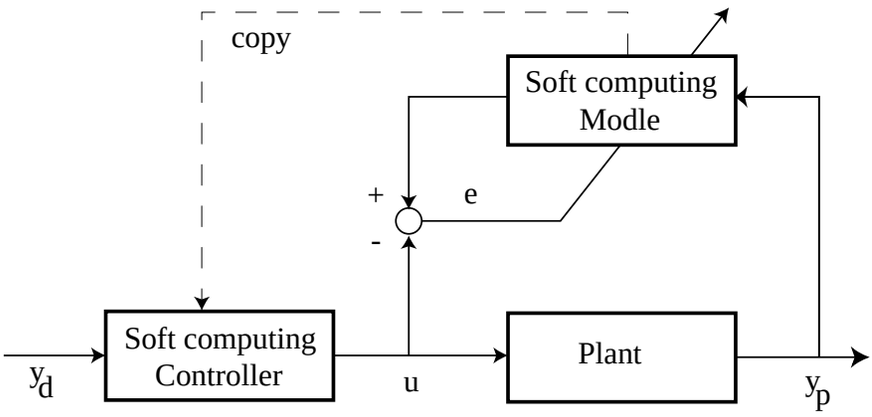


Figure 4.8: Adaptive inverse soft computing control with generalized learning

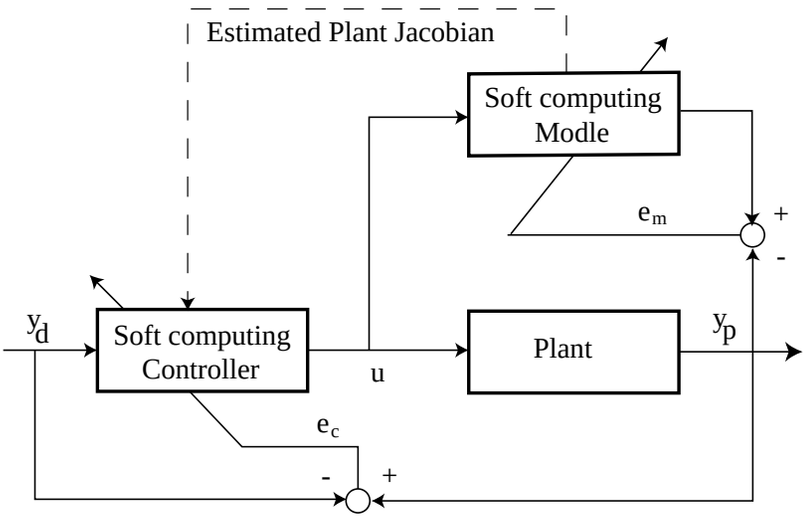


Figure 4.9: Adaptive inverse soft computing control with specialized learning

direct inverse control with generalized training is not a model-based design method due to the controller is inferred directly from a set of data without requiring an actual model of the system.

Example: Neural Net Based Direct Inverse Control with Generalized Learning

Assume that the plant to be controlled given by

$$y(k+1) = 0.9y(k) - 0.001y^2(k-1) + u(k) + sin(u(k-1)) \tag{4.67}$$

Firstly, the neural network direct inverse controller needs to be trained off-line by randomly selecting the input signal $u(k)$ at interval $[-2, 2]$, and measuring the output of the system $y(k+1)$. The input and output for training the direct neural network

inverse controller are $[y(k+1), y(k), y(k-1), u(k-1)]$ and $u(k)$. Secondly in the open-loop control stage, The input and output of the direct neural network inverse controller are $[r(k+1), y(k), y(k-1), u(k-1)]$ and $u(k)$. Where $r(k+1)$ is reference signal. A simulation result is shown in Fig. 4.10, which presented the tracking performance of the neural network direct inverse controller.

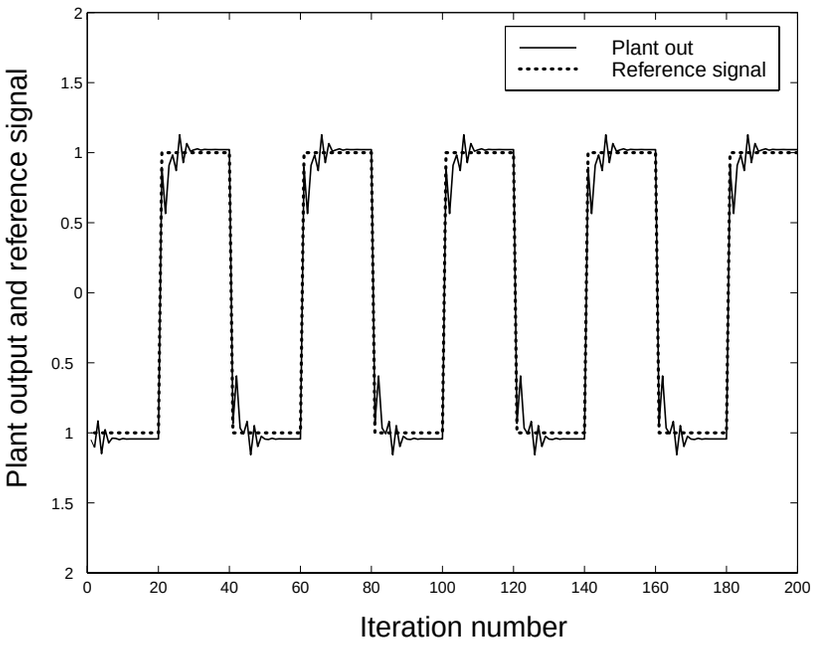


Figure 4.10: Tracking performance of the NN direct inverse controller with generalized learning

Note that in this control scheme the cost function for training the inverse dynamic neuro-controller is not directly based on the plant output error. So control performance cannot be improved directly by training unless learning has been carried out in such a way that good generalization through the control space can be expected.

Specialized Training

From the viewpoint of real-time applications, the specialized training method depicted in Fig.4.9, is relatively more practical than the generalized training method mentioned above. However, because the uncertain or unknown nonlinear plant lies between the controller and the output error, the standard gradient method for tuning the controller parameters are not available. This is the main obstacle of the specialized learning method for training the direct soft computing model based inverse controller.

In what follows, a neural network based direct inverse control algorithm with the specialized training scheme are given.

Consider the discrete-time SISO unknown nonlinear systems described by

$$y_p(k+1) = g(y_p(k), \ldots, y_p(k-n+1), u(k), \ldots, u(k-m+1)) + e(k) \tag{4.68}$$

where u, y_p and e are control input, plant output and the process noise, respectively. g is a continuous nonlinear function, n and m are the known system orders. The plant input is limited in amplitude for considering the real applications, i.e., there are u_m and u_M such that $u_m \leq u(k) \leq u_M$ for any k.

The goal of control task is to learn inverse dynamics of the plant in order to follow a specific a specified reference $y_d(k)$ as close as possible.

Estimation of the Plant Jacoban. In order to implement the direct neural inverse controller, the plant Jacoban should be estimated firstly. Some methods for approximating the plant Jacoban are listed as follows:

- Estimate the plant Jacoban by comparing the changes from the previous iteration, i.e.,

$$J(k+1) \approx \frac{y_p(k+1) - y_p(k)}{u(k) - u(k-1)} \tag{4.69}$$

  Such an approach can sometimes result in undesirable tracking. This undesirable effect is due to fast variation of J when the plant output and control input change suddenly, particularly at the zero-cross point. Moreover for tracking a constant reference signal, Eq.(4.69) may have the problem of division by zero.

- Change Eq.(4.69) into following form

$$J(k+1) \approx sign(y_p(k+1) - y_p(k)) \cdot sign(u(k) - u(k-1)) \tag{4.70}$$

  The problem of division by zero is avoided, but it cannot be used in real applications because it is sensitive to noise.

- Calculate the plant Jacobian through a model on-line.

  Let $x_m(k) = [x_{m,1}(k), \ldots, x_{m,n+m}(k)] = [y_p(k), \ldots, y_p(k-n+1)]$. Then the output of single hidden layer neuro-emluator for plant (4.68) can be expressed by

$$y_m(k+1) = f_{mo}\left(\sum_{j=1}^{Nm} \omega_j^o(k)\left[f_{mj}^h\left(\sum_{i=1}^{m+n} \omega_{mij}^h(k) x_{m,i}(k) + b_{mj}^h(k)\right)\right] + b_{mo}(k)\right) \tag{4.71}$$

  where ω_{mj}^o and ω_{mij}^h are the weights for the output and hidden layers respectively,

b_m is the bias, x_m is the input, N_m is the number of hidden units, f^h_{mj} and f_{mo} are the activation functions for the hidden and output layer respectively. Assume that the activation function

$$f(x) = \frac{1 - exp(-x)}{1 + exp(-x)} \tag{4.72}$$

is used in hidden layer and a linear function $f_{mo}(x)$ is used in output layer. Then we have $f^{'}_{mo} \equiv 1$ and

$$f^{'} = \frac{1}{2}(1 - f^2(x),\ f^{''}(x) = -\frac{1}{2}f(x)(1 - f^2(x)) \tag{4.73}$$

Thus, an estimated plant Jacobian can be calculated directly through the neural network emulator at each time step, that is

$$\hat{J(k)} = \frac{\partial y_m(k+1)}{\partial u(k)} = \sum_{j=1}^{N_m} \omega^o_{m,j}(k)\omega^h_{m,n+1,j}(k)f^{'}(k) \tag{4.74}$$

where

$$f^{'}(k) = \frac{1}{2}(1 - f^2(k)) = \frac{1}{2}\left(1 - f^2\left(\sum_{i=1}^{m+n} \omega^h_{mij}(k)x_{mi}(k) + b^h_{mj}(k)\right)\right) \tag{4.75}$$

- Enhancing estimation of the plant Jacobian [267]

  So far, as the estimated plant Jacobian is provided by using the neuro-emulator model, almost of all results are based on the following objective function to on-line train the plant neuro-estimator, that is

$$E(k) = \frac{1}{2}(y_m(k) - y_p(k))^2 \tag{4.76}$$

  Although the neuro-emulator trained by Eq.(4.74) did provide an on-line estimation of plants Jacobian, it cannot be expected to conduct a satisfied solution due to the lack of richness of teaching signals. A direct reason is no information on the plant Jacobian involved in neuro-emulator training. Therefore, an enhanced version for estimating plant Jacobian using the following new objective function is proposed.

$$E_\lambda(k) = \frac{1}{2}\left[(y_m(k) - y_p(k))^2 + \lambda\left(\frac{\partial y_m(k)}{\partial u(k-1)} - J_p(k)\right)^2\right] \tag{4.77}$$

  where λ is a weighted coefficient, $J_p(k)$ is an estimated plant Jacobian with process

noise which is given by

$$J_p(k) = \begin{cases} J_B, if J_p^o(k) > J_B \\ -J_B, if J_p^o(k) < -J_B \\ J_p^o(k), otherwise \end{cases} \tag{4.78}$$

where $J_B > 0$ is a certain threshold given by designers and

$$J_p^o(k) = \frac{y_p(k) - y_p(k-1)}{\varepsilon(k) + u(k-1) - u(k-2)} \tag{4.79}$$

where $\varepsilon(k) = \varepsilon_0 sign(u(k-1) - u(k-2))$, ε_0 is a sufficient enough small positive number for avoiding the problem of division by zero. The on-line updating rule for training the neuro-emulator then becomes

$$\begin{aligned} \triangle\omega_m(k+1) = e_m(k+1)\frac{\partial y_m(k+1)}{\partial \omega_m(k)} + \frac{\lambda \partial^2 y_m(k+1)}{\partial u(k)\partial \omega_m(k)} \\ \times \left(\frac{\partial y_m(k+1)}{\partial u(k)} - J_p(k+1) \right) \end{aligned} \tag{4.80}$$

where $\omega_m(k)$ denots the generic weights in the neuro-emulator model at time step k, $\triangle\omega_m(k) = \omega_m(k) - \omega_m(k-1)$, $e_m(k) = y_m(k) - y_p(k)$ and

$$\frac{\partial y_m(k+1)}{\partial \omega_j^o(k)} = f(k) \tag{4.81}$$

$$\frac{\partial y_m(k+1)}{\omega_{ij}^h(k)} = \omega_j^o(k) x_{m,i}(k) f^{'}(k) \tag{4.82}$$

$$\frac{\partial^2 y_m(k+1)}{\partial u(k)\partial \omega_j o(k)} = \omega_{n+1,j}^h(k) f^{'}(k) \tag{4.83}$$

$$\frac{\partial^2 y_m(k+1)}{\partial u(k)\partial \omega_{ij}^h(k)} = \delta_{i,n+1}\omega_j^o(k) f^{'}(k) + \omega_j^o(k)\omega_{n+1,j}^h(k) x_{m,i}(k) f^{''}(k) \tag{4.84}$$

where $\delta_{i,n+1} = 1$ for $i = n+1$ and 0 for otherwise.

Finally if the plant Jacobian is successfully estimated, a on-line direct inverse controller

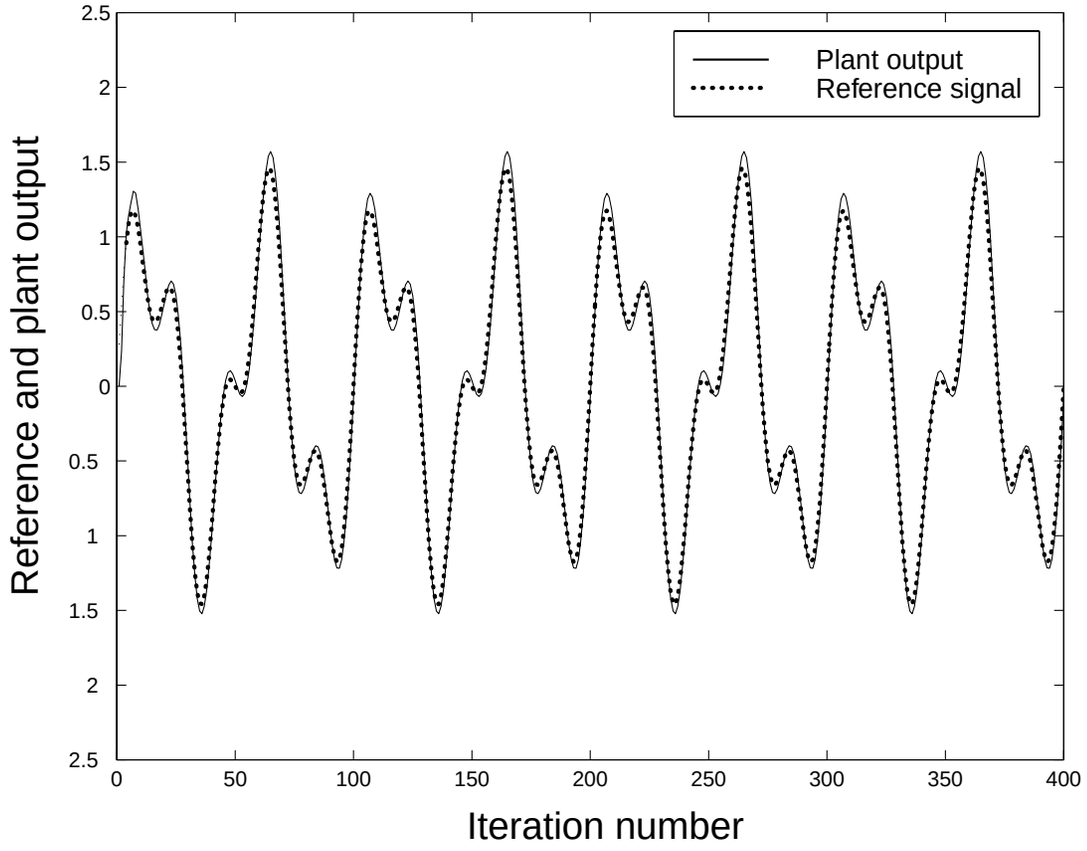


Figure 4.11: Tracking performance of the NN direct inverse controller with specialized learning

can be implemented with the following weight update rule.

$$\triangle\omega_c(k) \propto (y_p(k) - y_d(k))\hat{J}(k)\frac{\partial u(k)}{\partial \omega_c(k-1)} \tag{4.85}$$

where $\omega_c(k)$ denotes the generic weight in the neuro-controller at time step k, $\triangle\omega_c(k) = \omega_c(k) - \omega_c(k-1)$, $\hat{J}(k)$ is estimated plant Jacobian. The term $\partial u(k)/\partial\omega_c(k-1)$ can be calculated by BP algorithm.

Simulation Studies

The plant to be controled is given by

$$\begin{aligned} y(k+1) = 0.2y^2(k) + 0.2y(k-1) + 0.4sin(0.5 * (y(k) + y(k-1))) \\ cos(0.5 * (y(k) + y(k-1))) + 1.2u(k) \end{aligned} \tag{4.86}$$

The reference signal is given by

$$r(k) = sin(\pi k/50) + 0.5sin(\pi k/25) \tag{4.87}$$

The simulation result is shown in Fig.4.11.

4.3.2 Soft Computing Model Assistant Self-Tuning PID Control

The architecture of self-tuning soft computing model assistant PID controller is shown in Fig.4.9, in which the soft computing model can be any one of the neural network, fuzzy system or hybrid soft computing model discussed in Chapter 3.

Assume that the used soft computing model is a neural network, then a self-tuning neuro-PID controller can be derived as follows. The discrete-time PID controller can be described by

$$\begin{aligned} u(k) &= u(k-1) + K_p\{e(k) - e(k-1)\} \\ &+K_i e(k) + K_d\{e(k) - 2e(k-1) + e(k-2)\} \end{aligned} \tag{4.88}$$

$$e(t) = y_m(t) - y(t) \tag{4.89}$$

where, $u(k), y(k)$ and $y_m(k)$ are the output of PID controller, output of the plant and the reference model output respectively, and K_p, K_i, and K_d are the PID gains. In order to derive the self-tuning algorithm of the Neuro-PID controller, we define the error function E which should be minimized as

$$E = \frac{1}{2}e^2(t+1) \tag{4.90}$$

Using a three-layered neural network, we can realize the learning rule to find the suitable PID gains. The inputs of neural network are the reference signal, the input of the plant, the present and previous output of the plant. The outputs of neural network are K_p, K_i and K_d which are represented by $O(1), O(2)$ and $O(3)$, respectively. According to the gradient method, the following learning algorithm of the self-tuning Neuro-PID controller can be easily given.

1) Set the initial values of W_{kj}, W_{ji}, θ_k, θ_j, η and α, where W_{kj}, W_{ji}, θ_k, θ_j, η and α denote the connection weights of input layer and hidden layer, connection weights of hidden layer and output layer, the bias of hidden layer, the bias of output layer, the learning rate and momentum term, respectively.

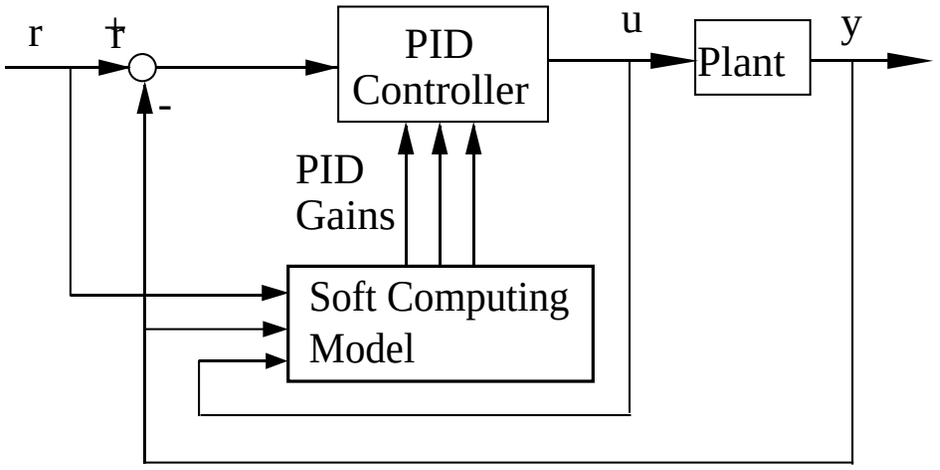


Figure 4.12: The architecture of self-tuning soft computing asistant PID controller

2) Calculate $e(t+1)$ and δ_k by

$$e(t+1) = y_m(t+1) - y(t+1) \tag{4.91}$$

$$\delta_k = e(t+1) * \frac{\partial y(t+1)}{\partial u(t)} * O(k) * (1 - O(k)) \frac{\partial u(t)}{\partial O(k)} \qquad (k = 1, 2, 3) \tag{4.92}$$

where

$$\frac{\partial u(t)}{\partial O(k)} = \begin{cases} e(t) - e(t-1) & (k=1) \\ e(t) & (k=2) \\ e(t) - 2e(t-1) + e(t-2) & (k=3) \end{cases} \tag{4.93}$$

3) Calculate

$$\triangle W_{kj}(t+1) = \eta \delta_k O_j + \alpha \triangle W_{kj}(t) \tag{4.94}$$

4) Calculate δ_j by

$$\delta_j = \sum \delta_k W_{kj} O_j (1 - O_j) \tag{4.95}$$

5) Calculate

$$\triangle W_{ji}(t+1) = \eta \delta_j O_i + \alpha \triangle W_{ji}(t) \tag{4.96}$$

6) t=t+1 and go to step 2.

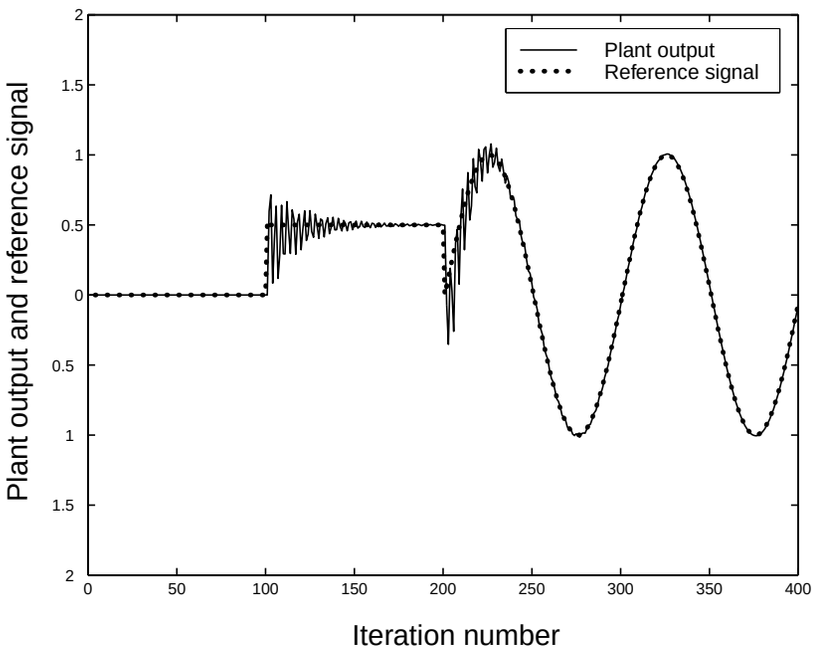


Figure 4.13: Tracking performance of the NN direct inverse controller with specialized learning

Simulation Studies

The plant to be controled is given by

$$y(k+1) = 0.9y(k) - 0.001y(k-1) + u(k) + sin(u(k-1)) \tag{4.97}$$

The simulation result is shown in Fig. 4.13.

4.3.3 Adaptive Soft Computing Control by State Feedback

Introduction

The main problems in designing of soft computing feedback controller are that the proofs of stability and convergence of the algorithms, the robustness of the controller, and the ability of the controller for dealing with the uncertainties and disturbances. So far with the use of Lyapounov synthesis method, a number of results have been obtained in the literature [147]-[170] for solving these problems, but they only can be applied to a classical

of affine nonlinear systems.

$$\begin{cases} \dot{x}_i = x_{i+1}, i = 1, 2, \ldots, n-1 \\ \dot{x}_n = a(x) + b(x)u \\ y = x_1 \end{cases} \tag{4.98}$$

where $x = [x_1, x_2, \ldots, x_n]^T \in R^n$, $u \in R$, $y \in R$ are the state variables, system input and output, respectively. In addition, the uniformly systemic method for the designing of neural/fuzzy feedback controller has not been recognized commonly.

Most recently, S. S. Ge, C. C. Hang and T. Zhang proposed a directly adaptive feedback control method by using RBF neural networks [170]. It is valuable to mention that the approaches can be applied to the control of non-affine nonlinear systems. This section is the further research about the feedback control of nonlinear affine/non-affine systems by using the basis function networks.

The basis function networks have been successfully applied to identification of nonlinear systems discussed in Chapter 3. In this Section, we focus on the purpose of control by using the basis function networks.

Problem Statement

Given a typical nonlinear system

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \vdots \\ \dot{x}_n = f(x, u) \\ y = x_1 \end{cases} \tag{4.99}$$

where $x = [x_1, x_2, \ldots, x_n]^T \in R^n$ is state vector and $u \in R$, $y \in R$ are input and ouput of the system. If $f(x, u)$ can be described by $f(x, u) = a(x) + b(x)u$, then the system (4.99) is called an affine nonlinear system; otherwise, it is called a non-affine nonlinear system.

The control objective is that given a desired output, $y_d(t)$, find a control u, such that the output of the system tracks the desired trajectory with an acceptable accuracy, while all the states and the control remain bounded.

In order to control the system (x), the following assumptions are needed.

Assumption 1: $f(x, u)$ is C^1 for $(x, u) \in R^{n+1}$ and $f(x, u)$ is a smooth function with respect to input u.

Assumption 2: $\partial f(x, u)/\partial u \neq 0$ for all $(x, u) \in R^{n+1}$, and the sign of $\partial f(x, u)/\partial u$ is

known.

Assumption 3: The reference signal $y_d(t), y_d^{(1)}(t), y_d^{(2)}(t), \ldots, y_d^{(n)}(t)$ are smooth and bounded.

Assumption 4: On a compact set $\omega_z \in R^{n+1}$, the ideal basis function network weights W^* satisfies

$$\| W^* \| \leq w_m \tag{4.100}$$

where w_m is a positive constant.

Define vector x_d and ζ as

$$x_d = [y_d(t), \dot{y}_d(t), \ldots, y_d^{(n-1)}(t)]^T \tag{4.101}$$

$$\zeta = x - x_d \tag{4.102}$$

and a filtered tracking error as

$$e = [\Lambda \; 1]^T \zeta \tag{4.103}$$

where $\Lambda = [\lambda_1, \lambda_2, \ldots, \lambda_{n-1}]^T$ is an appropriately chosen coefficient vector so that $\zeta(t) \rightarrow 0$ as $e(t) \rightarrow 0$, (i.e. $s^{n-1} + \lambda_{n-1} s^{n-2} + \ldots + \lambda_1$ is Hurwitz). Then, the time derivative of the filtered tracking error can be written as

$$\dot{e} = f(x, u) - y_d^{(n)}(t) + [0 \; \Lambda^T]\zeta \tag{4.104}$$

Direct Adaptive Control of Nonlinear System by State Feedback

A directly adaptive state feedback control algorithm is given for the affine and/or non-affine nonlinear systems, in which each of the different basis function networks is used as a direct controller and the stability of the closed control system are guaranteed according to the following two theorems.

Theorem 1: Suppose system (3) satisfying Assumptions 1-3, then, there exist an ideal control input, u^*, such that

$$\dot{e} = -k_v e - k_v sat(e) \tag{4.105}$$

where $sat(e)$ is a continuous function defined as

$$sat(e) = \begin{cases} 1 - exp(-e/\gamma), e > 0 \\ -1 + exp(e/\gamma), e \leq 0 \end{cases} \tag{4.106}$$

and k_v is a positive constant. Subsequently, Eq.(4.105) leads to

$$lim_{t\to\infty} \mid y(t) - y_d(t) \mid = 0 \tag{4.107}$$

Proof: See Ref.[170].

Remark 4.12: Theorem 1 guarantee the existence of an ideal control input for the system (3), but it is unknown that how to construct this ideal control input. In the following, it is supposed that the ideal control input is approximated using the different basis function networks and the weights update law of the basis function networks is given.

Theorem 2: For system (3), if the controller is given by each of the following equation

(a) $u = U_{GBRF}$ (the output of the GBRF network)

(b) $u = U_{VPBF}$ (the output of the VPBF network)

(c) $u = U_{B-Spline}$ (the output of the B-spline basis function network)

d) $u = U_{Fuzzy}$ (the output of the fuzzy basis function network)

(e) $u = U_{Wavelet}$ (the output of the wavelet basis function network)

and the corresponding weights of the basis function networks are updated by

$$\dot{W} = -(k_0\|W\| + k_1|e| + k_2)S(z)e - \delta(\|W\| + |e| + 1)\|S(z)\||e|W \tag{4.108}$$

and with

(1) Assumptions 1-4 being satisfied, and

(2) the existence of two compact sets D_w and D_e such that $W(0) \in D_w$ and $e(0) \in D_e$,

then, for a suitably chosen design parameter K_v, the filtered tracking error e, the weights of the basis function networks and all system states are semi-globally uniformly ultimately bounded. In addition, the tracking error can be made arbitrarily small by increasing the controller gains and the nodes number of the basis function networks.

Proof: Similar to Ref.[170].

Remark 4.13: It is valuable to mention that only the weights of the basis function networks are adjusted (the other parameters used in the basis function networks are fixed in the adaptive tuning of the controller). In addition, $z = [x^T, \nu]^T \in R^{n+1}$ is the input

vector of the basis function networks and ν is calculated as

$$\nu = k_v e + k_v sat(e) - y_d^{(n)}(t) + [0\ \Lambda^T]\zeta. \tag{4.109}$$

Simulation Studies

Two examples are presented to verify the effectiveness of the proposed methods for the adaptive control of nonlinear affine and non-affine systems. Fourth order Runge-Kutta algorithm is used in the simulation for solving the nonlinear differential equation.

Example 1

The first plant to be controlled is an affine nonlinear system (Inverted Pendulum Problem) given by the following equation.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \frac{gsin(x_1) - \frac{mlx_2^2 cos(x_1) sin(x_1)}{M+m}}{l(\frac{4}{3} - \frac{mcos^2 x_1}{M+m})} + \frac{\frac{cos(x_1)}{M+m}}{l(\frac{4}{3} - \frac{mcos^2 x_1}{M+m})} u \\ y = x_1 \end{cases} \tag{4.110}$$

where M is the mass of the cart, m is the mass of the rod, $g = 9.8\frac{m}{sec^2}$ is the acceleration due to gravity, l is the half length of the rod, and u is the control input. In this example, it is assumed that $M = 1kg$, $m = 0.1kg$, and $l = 0.5m$. Our control objective is to control the state $x1$ of the system to track the reference trajectory $y_d = \frac{\pi}{40.0} sin(t)$. The initial state of the system is $x_1(0) = -\frac{\pi}{50.0}, x_2(0) = 0.0$. The sampling time is 0.005s, the maximum simulation time is 20s.

The fixed parameters are $k_0 = k_1 = k_2 = 10.0$, $\Lambda = 10.0$ and $\gamma = 0.03$ for each basis function network controller. The other parameters used in the basis function network controller are shown in Table 4.1.

The control performances of different basis function networks are shown in Fig. 4.14-Fig.4.18, respectively.

Example 2

The second plant to be controlled is a non-affine nonlinear system given by the following equation.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \frac{x_1}{1+x_2^2} + u^5 + u^3 + ue^{u^2} \\ y = x_1 \end{cases} \tag{4.111}$$

The control objective is to control the state x_1 of the system to track the reference trajectory $y_d = 0.2 * sin(0.5 * t) + 0.1 * cos(t)$. The initial state of the system is $x_1(0) =$

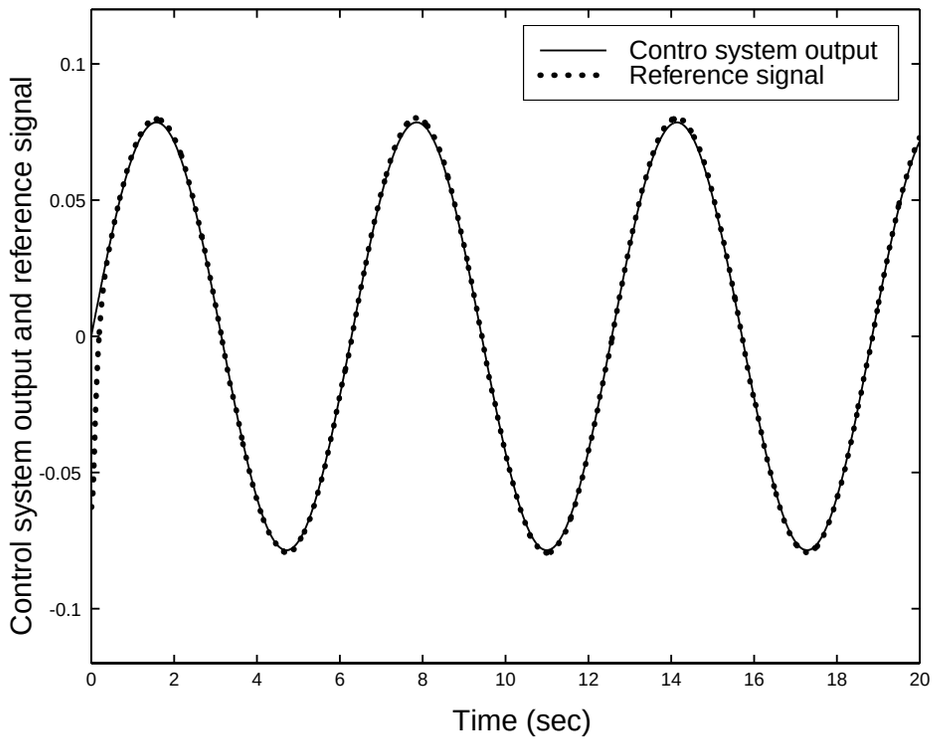


Figure 4.14: Tracking performance of state feedback control (by RBF neural networks)

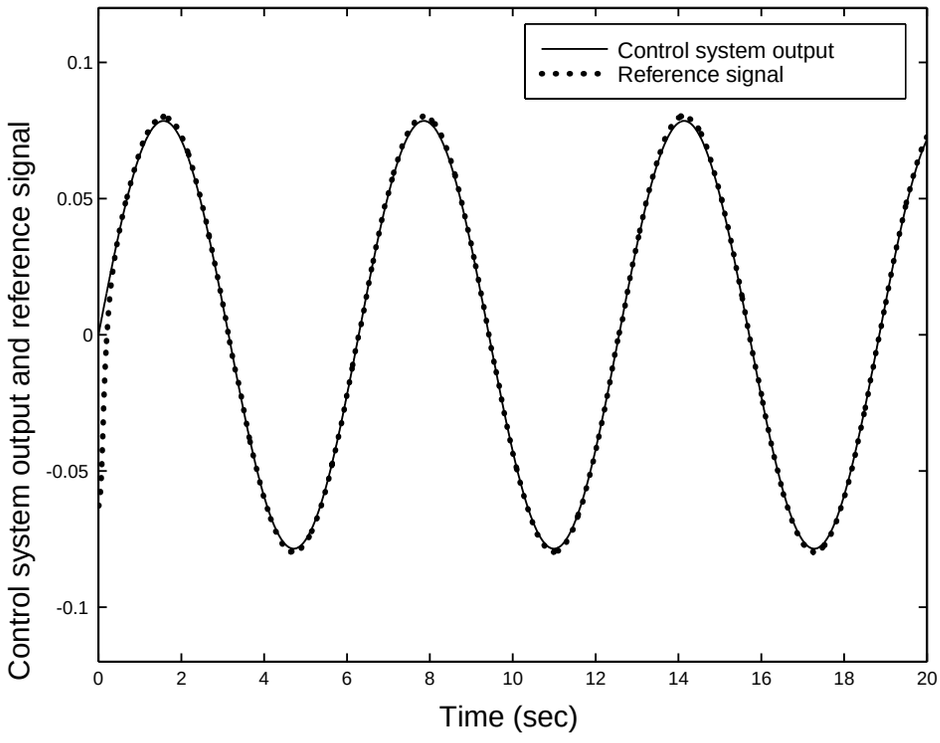


Figure 4.15: Tracking performance of state feedback control (by B-spline basis function networks)

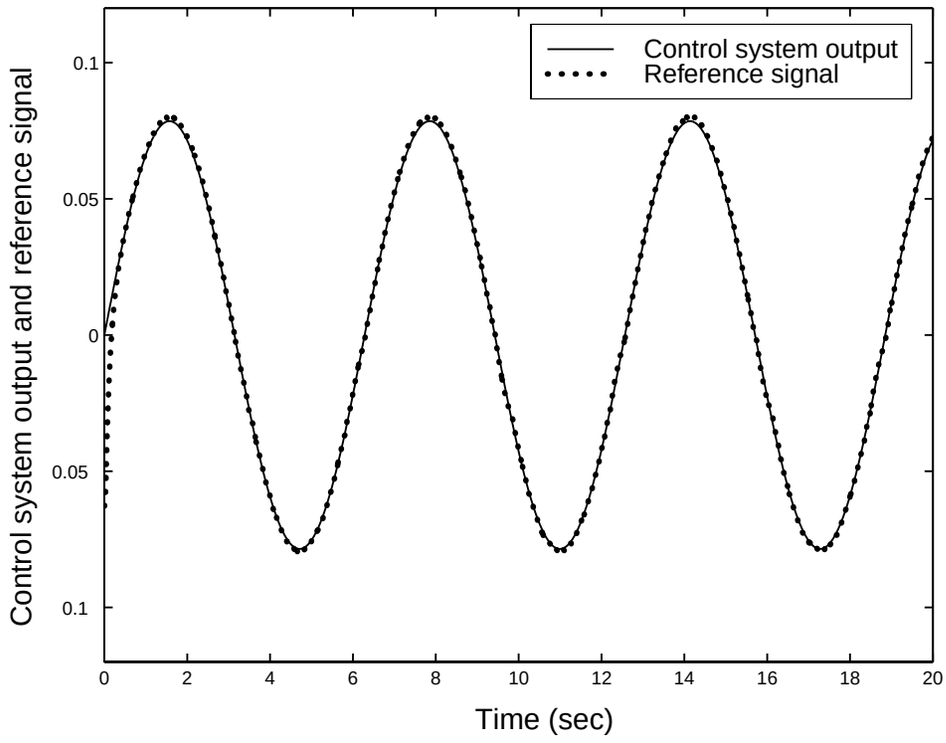


Figure 4.16: Tracking performance of state feedback control (by fuzzy basis function networks)

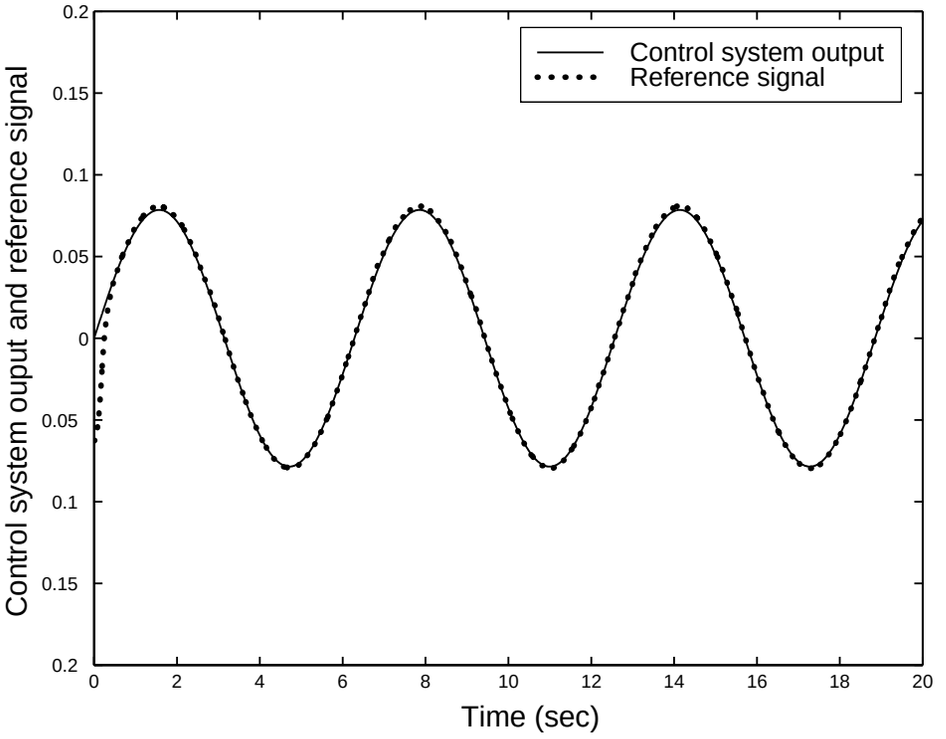


Figure 4.17: Tracking performance of state feedback control (by VPBF networks)

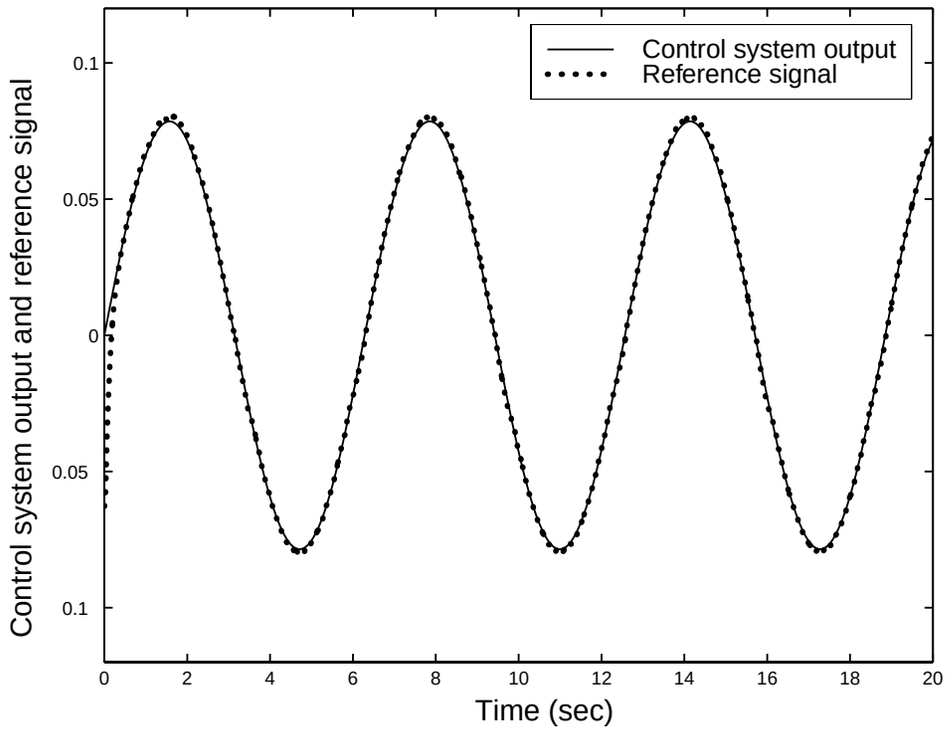


Figure 4.18: Tracking performance of state feedback control(by wavelet basis function networks)

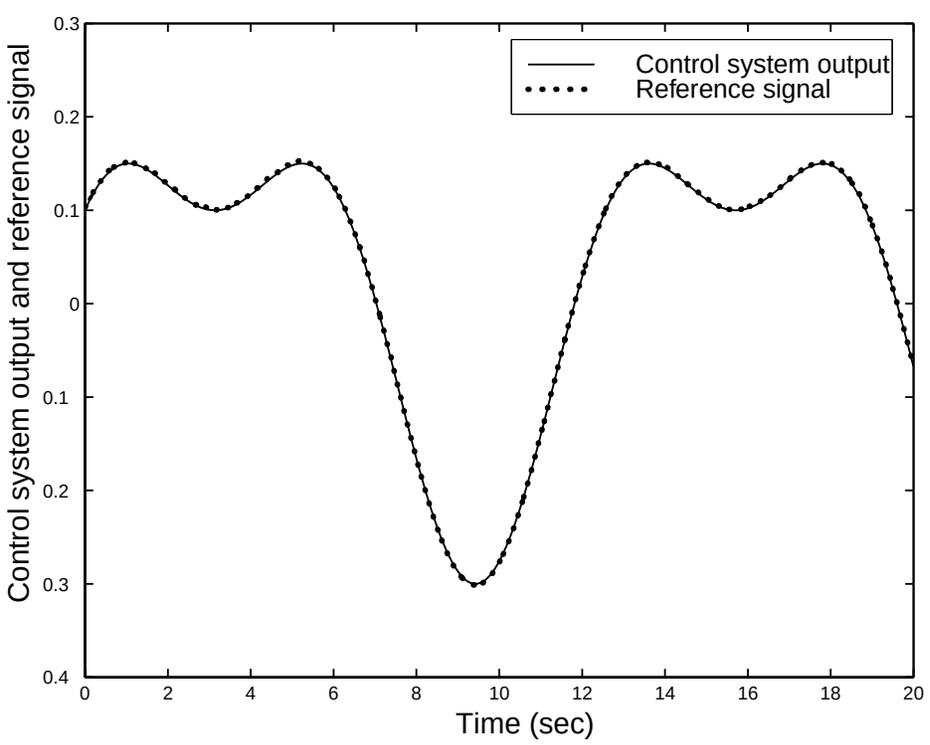


Figure 4.19: Tracking performance of state feedback control (by RBF neural networks)

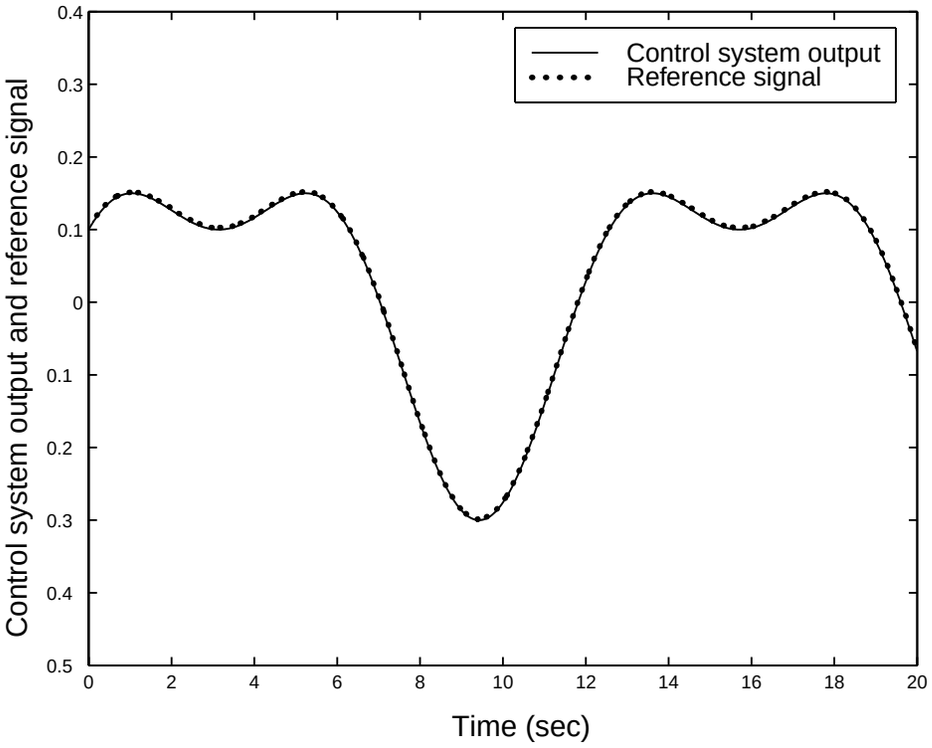


Figure 4.20: Tracking performance of state feedback control (by B-spline basis networks)

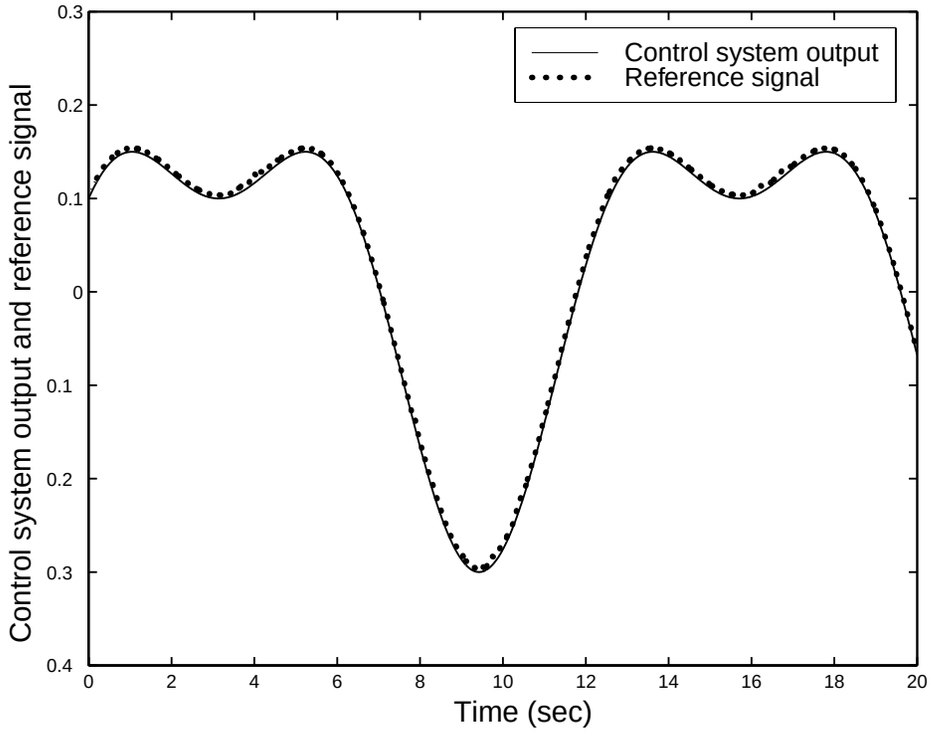


Figure 4.21: Tracking performance of state feedback control (by fuzzy basis networks)

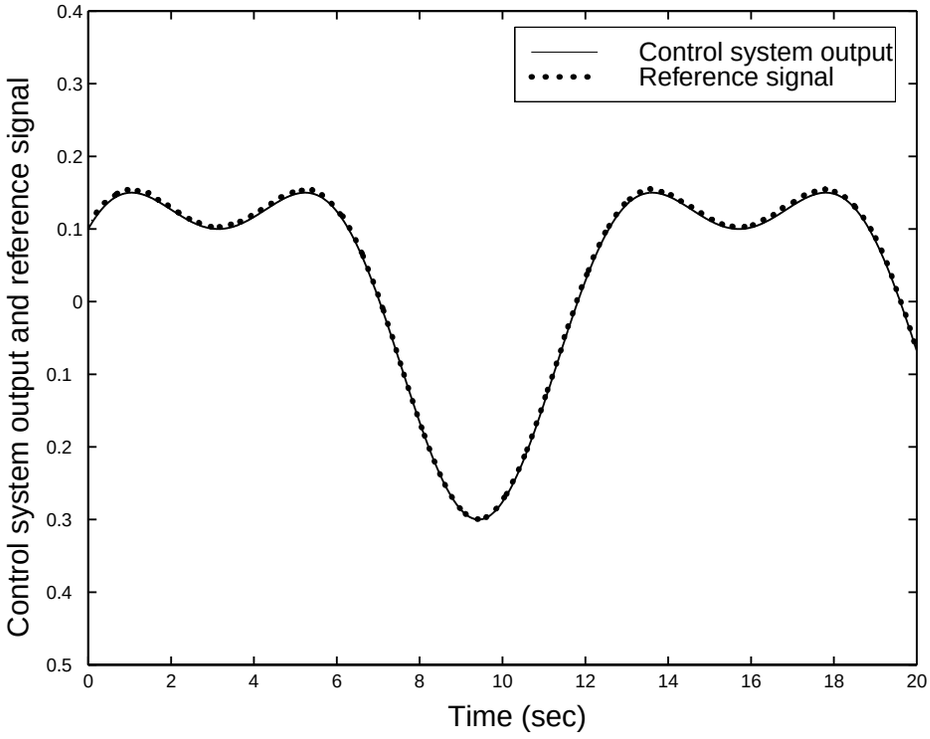


Figure 4.22: Tracking performance of state feedback control (by VPBF networks)

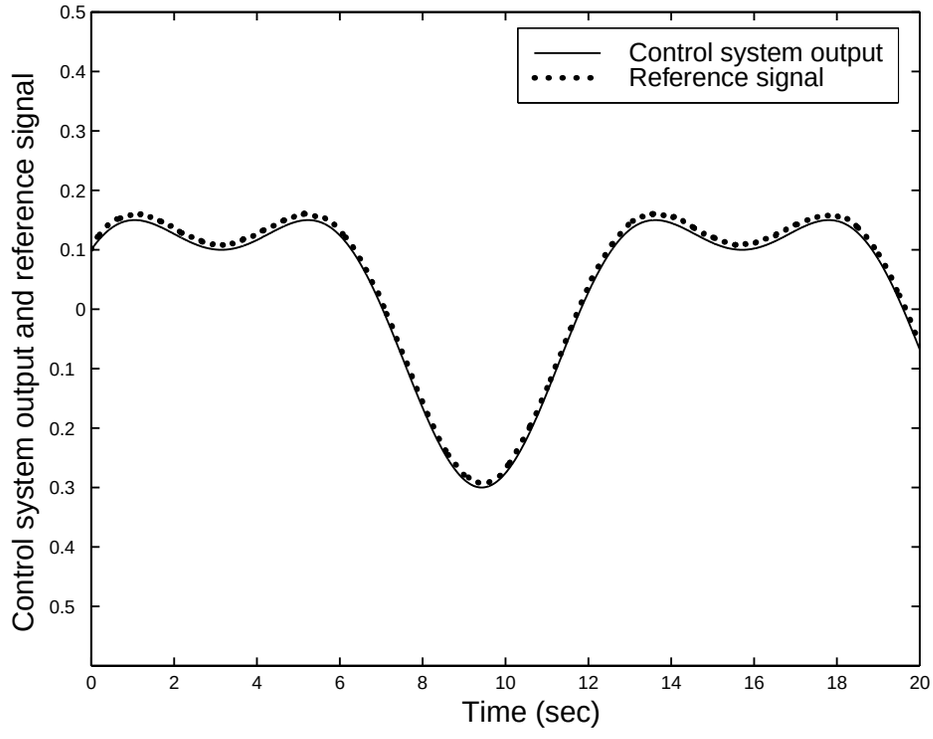


Figure 4.23: Tracking performance of state feedback control(by wavelet basis networks)

Table 4.1: Parameters used in Example 1

BFN	No. of hidden	k_v	δ	initial a	initial b	initial weight w
GRBF-NN	12	5	10	0.01	0.0	rand(0,1)
VPBF-NN	10	2	5	No	No	0.1
B-spline-NN	8	2	4	80.0	0.0	rand(0,1)
Fuzzy-NN	10	2	4	10.0	0.0	rand(0,1)
Wavelet-NN	12	2	4	60.0	0.1	0.0

Table 4.2: Parameters used in Example 2

BFN	No. of hidden	k_v	δ	initial a	initial b	initial weight w
GRBF-NN	8	4	20	0.01	0.0	0.0
VPBF-NN	10	1.5	13.5	No	No	rand(0,1)
B-spline-NN	10	4	10	80.0	0.0	rand(0,1)
Fuzzy-NN	20	4	15	10.0	0.0	0.0
Wavelet-NN	20	5	25	60.0	0.01	0.0

$0.1, x_2(0) = 0.2$. The sampling time is 0.005s, the maximum simulation time is 20s.

The fixed parameters are $k_0 = k_1 = k_2 = 10.0$, $\Lambda = 10.0$ and $\gamma = 0.03$ for each basis function network controller. The other parameters used in the basis function network controller are shown in Table 4.2.

The control performances of different basis function networks are shown in Fig. 4.19-Fig. 4.23, respectively.

From above simulation results it can be seen that each of the basis function networks can be effectively used as a directly adaptive controller for the control of nonlinear affine/non-affine systems.

Section Summary

The proposed control algorithms can be implemented online and there is no requirement for persistent excitation condition for tracking convergence.

It is also valuable to mention that only the weights of each basis function networks are update to force the systems to follow the desired trajectory. The parameters within the each basis function networks are fixed in the closed loop control process. Therefore there is no need to pre-determine the shape and parameters of the basis function networks and to pre-partition the input space. But the initial values of the parameters used in the basis function networks must be suitable chosen. One possible principle for selecting proper initial values of the parameters within the each of basis function networks is that the

initial values can be selected such that there is no overflow in the processing of weights update.

In addition, it is unknown that if it is benefit to allow the parameters used in the basis function networks (a and b) adjustable for the complex dynamic nonlinear systems. This is an interesting problem needed for further research.

5 basis function networks and its calculation architectures are given, and a directly adaptive state feedback control approach based on above basis function networks is proposed in this Section.

Simulation study and experimental comparison show that the proposed method is efficient for the control of nonlinear affine/non-affine systems. In addition, the current research show that it is possible to construct a uniformly framework (theory) for control nonlinear affine/non-affine systems by directly adaptive state feedback control approaches.

4.4 Conclusion in this Chapter

Some of hybrid soft computing model based or assistant adaptive control schemes, i.e., PIPE based evolutionary control and a unified framework of the basis function network based adaptive state feedback control scheme are developed in this research. The effectiveness of the proposed control algorithms is confirmed by simulation studies.

Some soft computing model based or assistant controller design principles are also discussed, i.e., sliding mode control based design and backstepping design. Currently, in these design methods the used soft computing models are neural networks and/or fuzzy systems. Other soft computing models can also be used for designing of these controllers in principle. It is valuable to implement and compare the performances of various soft computing models based control schemes in order to construct a unified framework of hybrid soft computing based or assistant control scheme finally.

Chapter 5

Application to the Drilling System

5.1 Identification of the Thrust Force in the Drilling Process

5.1.1 Introduction

The drilling process has a great importance for the production technology due to its widespread use in the manufacturing industry. In the drilling process, the spindle speed and feed rate are usually pre-determined according to the type and the variety of the workpiece materials. In order to prevent the drill from the damage and enhance a maximum production rate of the drilling system, it is important to monitor and control the drilling system.

Artificial neural networks have been successfully applied to a number of scientific and engineering fields recently, especially for the identification and control of nonlinear systems. A number of researches also shown that complex mechanical systems can be identified and controlled by various on- or off-line trained neural networks, e.g., Neural networks based DC motor control, induction motor control and robot manipulator control. In the researches of neural network based system identification and control, there have been two key directions. The one is that try to improve or create new algorithm for training the neural network in order to satisfy the need of engineering practice, i.e., the convergence speed of learning algorithm and the approximation accuracy of the neural model. The other direction is that trying to improve or find advanced neural identification and control technique, and apply it to the complex industrial plant.

There have been a number of researches in the optimization, estimation and control of drilling process recently [179]-[186]. Soft computing approach has been a popular research fields in the tool condition monitoring and control. Cutting parameter selection

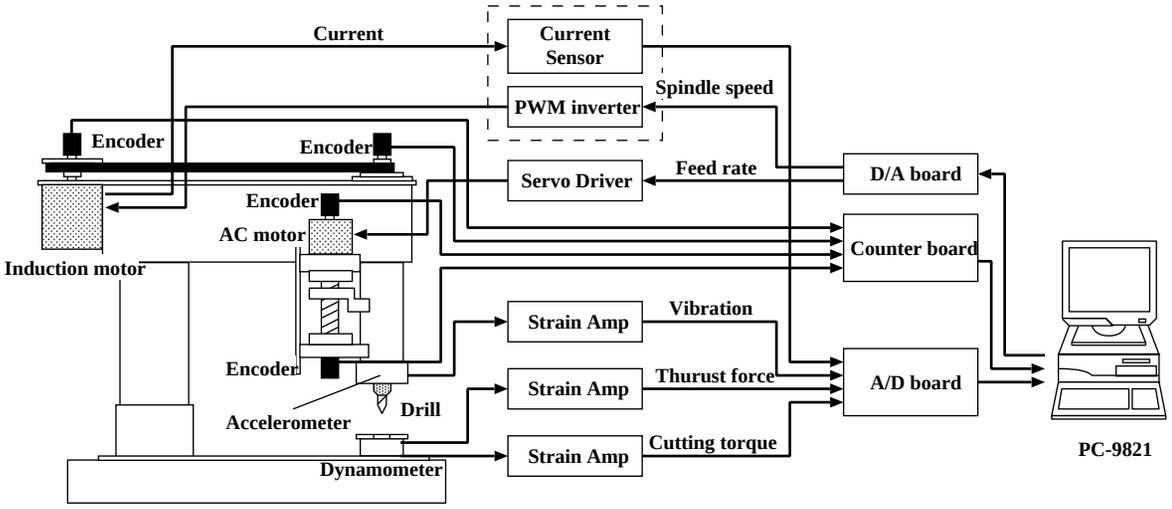


Figure 5.1: The drilling system

or regulation based on neural network and genetic algorithm has been presented [179][180]. The tool ware estimation by using neural or neuro-fuzzy techniques can be found in [186],[187]-[189]. A number of machining force control schemes have also been discussed in the previous works, e.g., traditional adaptive control schemes [189][190], linearisation control [191], robust control [192], sliding mode control [193], neural network control [194], and fuzzy control [195].

The thrust force and cutting torque in the drilling process are usually measured by sensor. But as it is pointed that the problems i.e., vibration, will occur when the sensor is attached to the drilling machine.

In this research, an estimation and control method of the thrust force is proposed by using neural network technique. In order to obtain a good neural model of thrust force in the drilling process under a variety of conditions, the data set for training neural network is selected by randomly setting the values of spindle speed and feed rate at the pre-defined ranges. Based on the neural model of the thrust force, a neural controller can be trained to control the thrust force off-line, and then, this controller can be applied to the drilling machine to on-line control the thrust force with the recursive least square learning algorithm.

5.1.2 The Drilling Machine

The used drilling machine is shown in Fig. 5.1, which contains the following components: an induction motor, a AC motor, 4 encoders, a dynamometer, a current sensor, a PWM inverter, a servo driver, 3 strain amplifier, a counter board, a AD/DA converter and a

personal computer (PC-9821).

The signals of encode number, the current of spindle system, vibration, thrust force and cutting torque are sent to the personal computer at every sampling time (i.e., 6 $[msec]$) in order to monitor and analyze the dynamics of the drilling process. Meanwhile, the computer provides the control action to manipulate drilling system by spindle speed and feed rate.

The objective of this research is modeling and control the thrust force by using the neural networks. A number of factors, i.e., feed rate, cutting torque, spindle speed, hole depth and the current and voltage values of spindle system, may affect the thrust force in the drilling process. In this research, two thrust force prediction models are constructed in order to find a effective thrust force model. The one is based on the feed rate, the previous feed rate and thrust force. The other is based on the feed rate, the current and voltage of the spindle system and the previous thrust force.

5.1.3 Neural Network Model of the Thrust Force

Sampling Data

In order to obtain a general thrust force model at different dynamic cutting conditions, we set the spindle speed and feed rate as uniformly random values at pre-defined ranges. The spindle speed varied at interval 1200-2000 [rpm], and the feed rate varied at interval 0.01-0.08 [mm/rev]. The experimental setup of cutting conditions is shown in Table.5.1.

The feed rate, the phase voltage and the current signals of spindle system and the thrust force in the drilling process will be measured and sent to computer for the later use for the training of neural networks.

Neural Network Training Algorithm

The used neural network training algorithm for modeling is batch version of momentum backpropagation algorithm as discussed as in Chapter 2.

Identification Experiments

In this section, we present some simulation and real time implementation results to verify the effectiveness of the proposed methods for the estimation of thrust force in the drilling process.

The used parameters in the training of neural network: $\eta_1 = 0.01$, $\alpha_1 = 0.9$, $\eta_2 = 0.005$, $\alpha_2 = 0.95$, $\eta_3 = 0.002$, $\alpha_3 = 0.8$.

Model 1

The neural network has three inputs including the feed rate, the previous feed rate and the previous thrust force. The output is the current thrust force.

The neural network was trained off-line based on the above training algorithm. The model thrust force and real thrust force is shown in Fig. 5.2.

For the validation of the neural model of thrust force, two tests are performed. The one is by randomly re-setting the spindle speed and feed rate at the ranges of $1200-2000[rmp]$ and $0.01-0.08[mm/rev]$, respectively. The other is by setting the spindle speed and feed rate are $1800[rpm]$ and $0.03[mm/rev]$, respectively. The test results are shown in Fig. 5.3 and Fig. 5.4.

Model 2

The neural network has seven inputs including the feed rate, the previous feed rate, the current and voltage of spindle system and their previous values, and the previous thrust force. The output is the current thrust force.

The model thrust force and real thrust force is shown in Fig. 5.5.

Same as the method proposed in model 1, for validation the thrust force model, two tests are performed which are shown in Fig. 5.6 and Fig. 5.7, respectively.

From above results, it can be seen that the proposed method works very well for generating the thrust force models.

5.1.4 Neural Control of the Thrust Force

It is expected that maintaining a constant thrust force when the tip of drill is reached to the workpiece due to it can reduce the vibration, and enhance the production rate and quality of the hole. In this section, a neural network based constant thrust force control method is presented.

Neural Controller and Its Training Algorithm

The used neural controller is a three layer feedforward neural network with hyperbolic tangent activation functions in the input-to-output layer and linear activation functions in the output layer (see, Fig. 5.8).

The neural controller has three inputs, which are the reference thrust force, the previous feed rate and thrust force. The output of neural controller is the feed rate. From

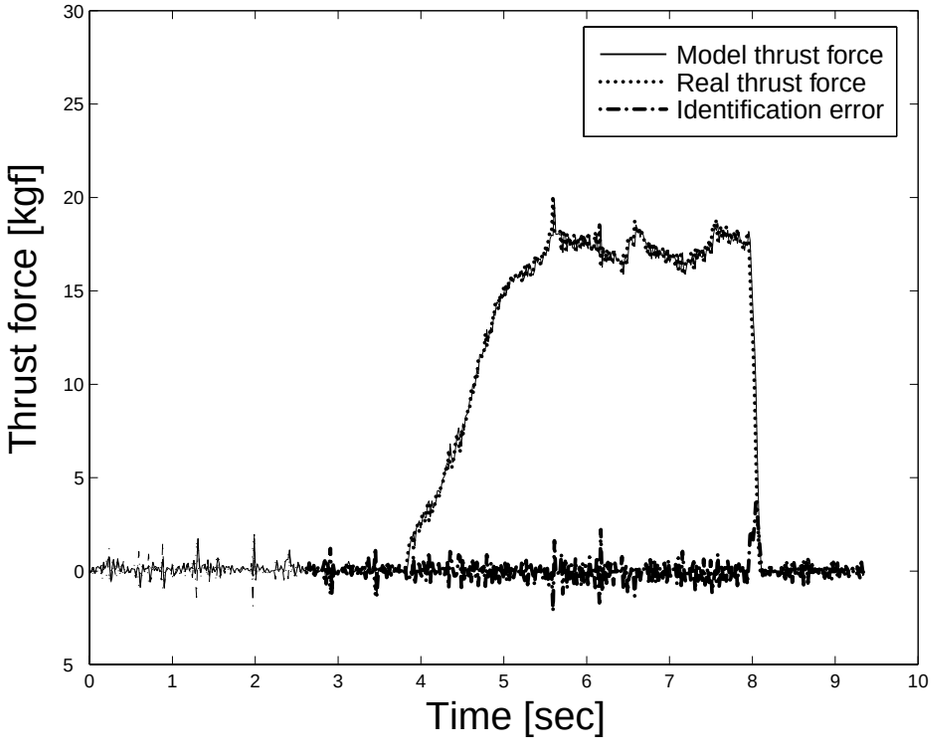


Figure 5.2: Model and real thrust force for training data set

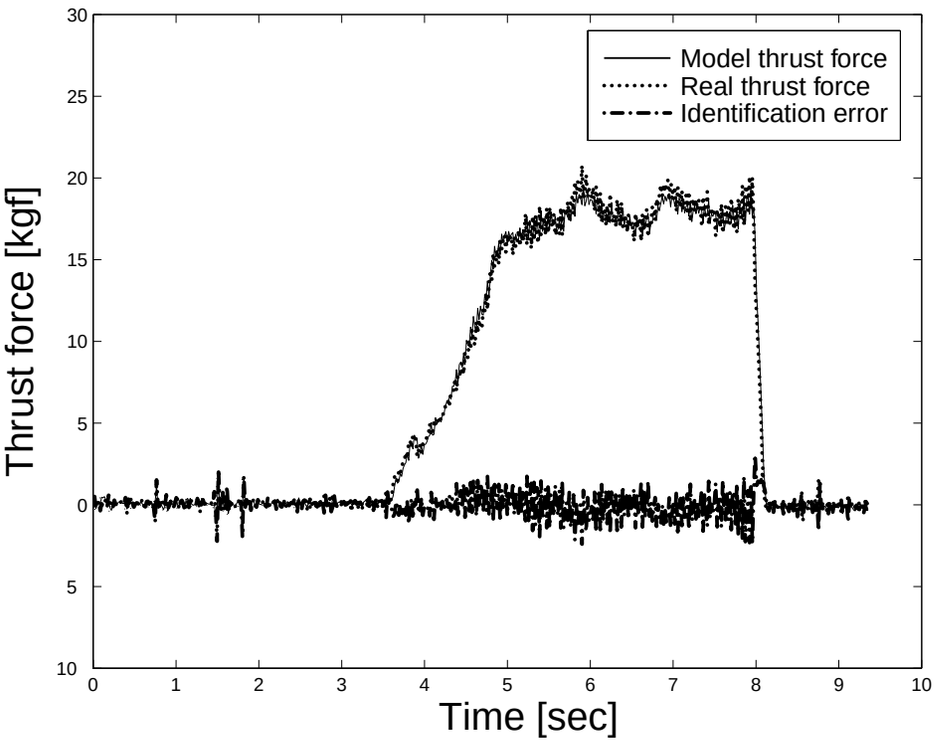


Figure 5.3: Test: model and real thrust force by randomly re-setting the spindle speed and feed rate

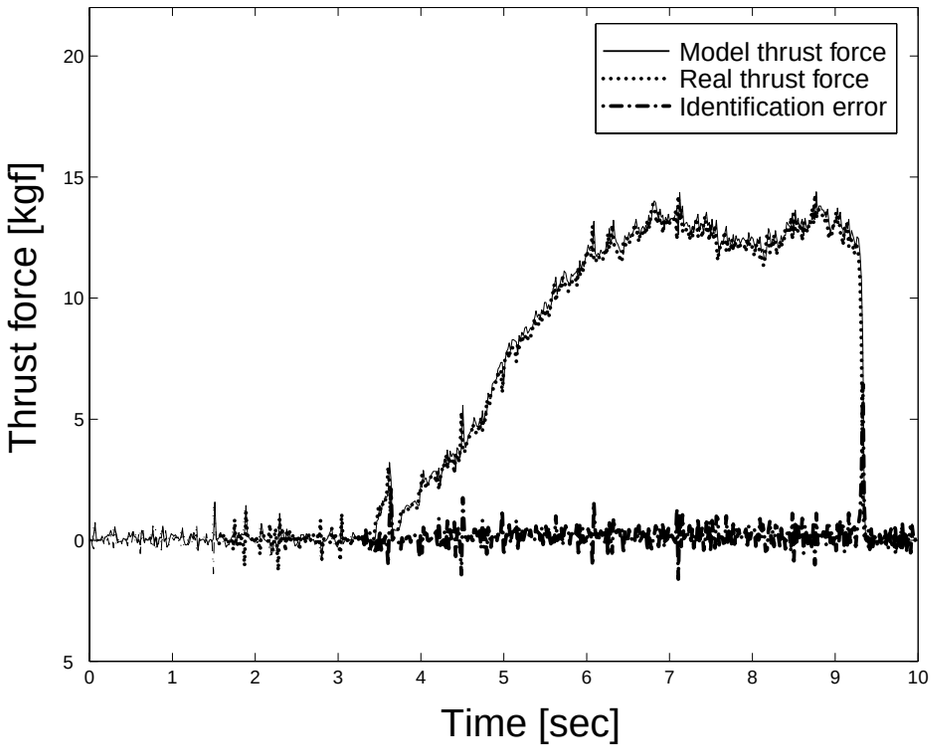


Figure 5.4: Test: model and real thrust force for spindle speed 1800[rpm] and feed rate 0.03[mm/rev]

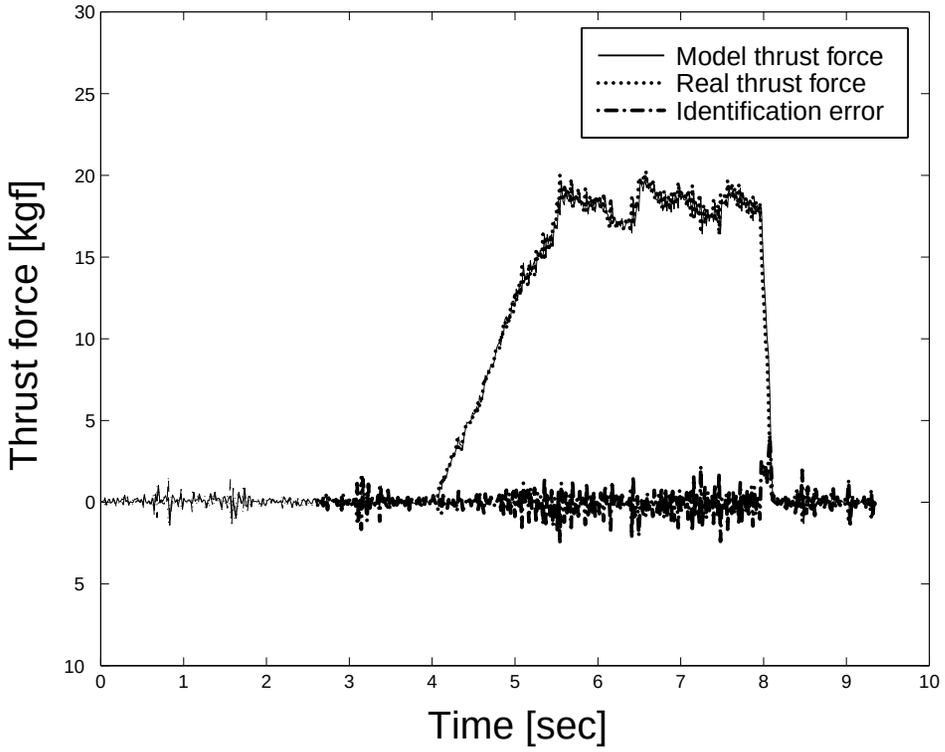


Figure 5.5: Model and real thrust force for training data set

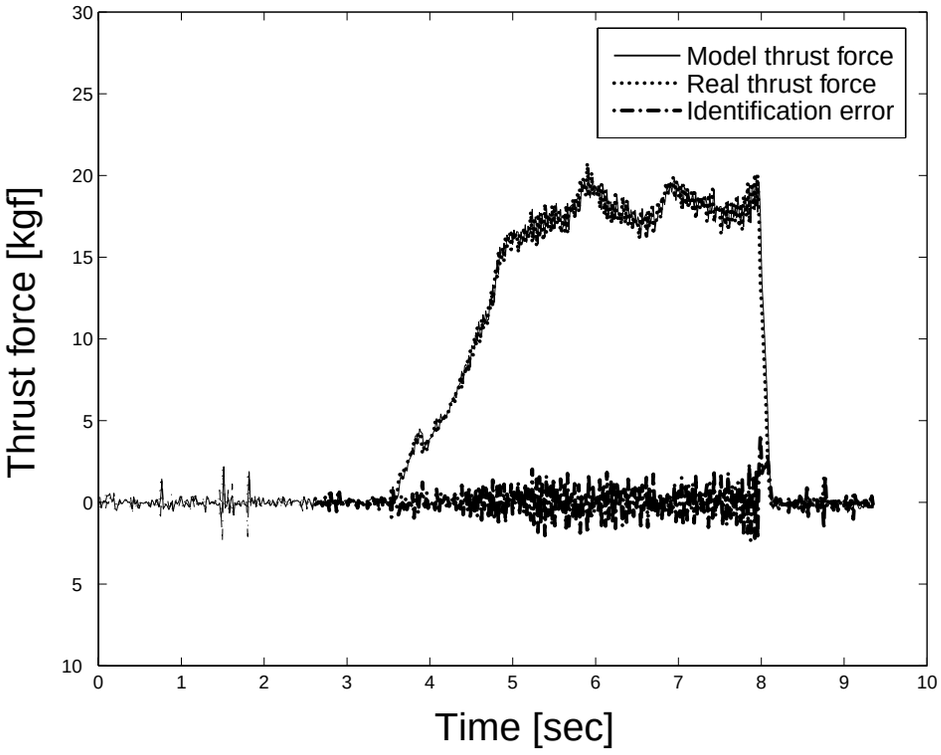


Figure 5.6: Test: model and real thrust force by randomly re-setting the spindle speed and feed rate

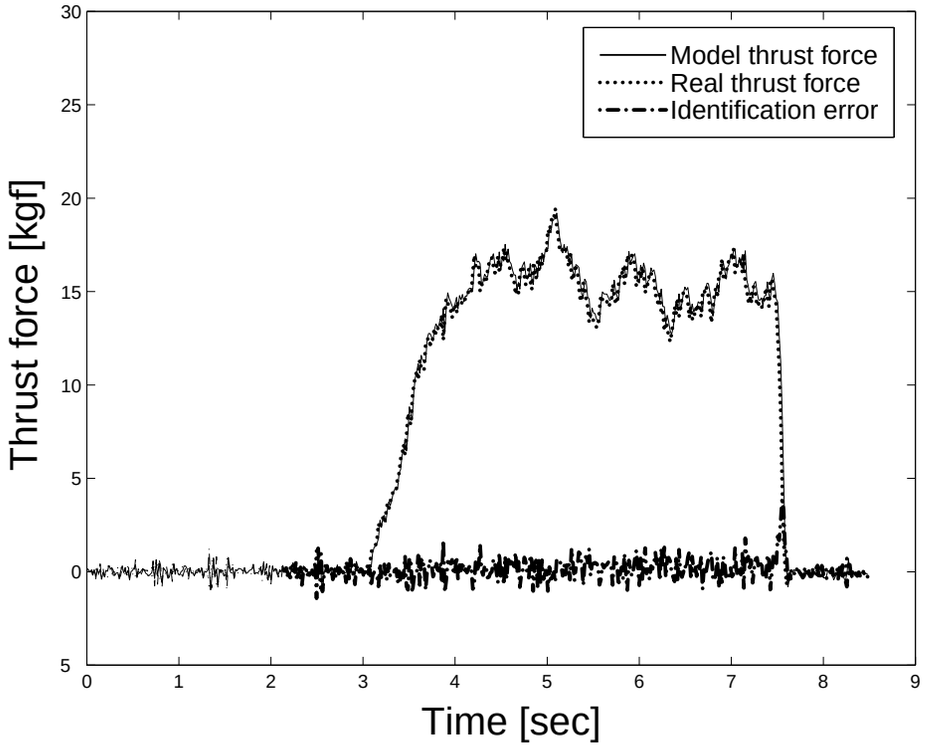


Figure 5.7: Test: model and real thrust force for spindle speed 1800[rpm] and feed rate 0.03[mm/rev]

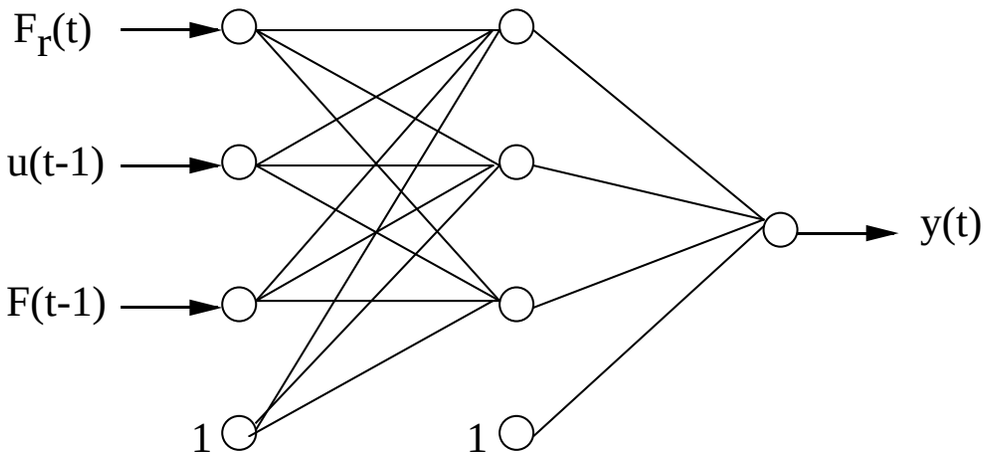


Figure 5.8: The thrust force neural controller

Fig.8, we can get the output of neural controller as

$$y(t) = \sum_{j=0}^{n_h} w_{1,j} h_j(t) = \sum_{j=1}^{n_h} w_{1,j} f_j(\sum_{l=0}^{n_i} w_{j,l} x_l(t)) + w_{1,0} \tag{5.1}$$

where $w_{1,j}$ denotes the hidden-to-output layer weights, 1 denotes that there is only one output of the neural controller. $h_j(t)$ denotes the outputs of the hidden layer.

The real output of neural controller in implementation is determined by scaling the output of the neural controller as

$$u(t) = y(t)(Umax - Umin) + Umin \tag{5.2}$$

where $Umax$ and $Umin$ are the maximum and minimum feed rate of the drilling machine.

The used training algorithm for neural controller is recursive least square. Define the objective function as

$$E(t) = \frac{1}{2} e^2(t) = \frac{1}{2}(F_r(t) - F(t))^2 \tag{5.3}$$

where $F(t)$ and $F_r(t)$ are the reference and measured thrust force at sampling time t.

Firstly, the regression vectors of input-to-hidden and hidden-to-output layer can be formed as

$$\phi_h^T(t) = [x_1(t), x_2(t), \ldots, x_{n_i}(t), 1] \tag{5.4}$$

$$\phi_o^T(t) = [h_1(t), h_2(t), \ldots, h_{n_h}(t), 1] \tag{5.5}$$

where n_i and n_h are the number of neurons in the input and hidden layer, respectively.

In addition, the weights in the input-to-hidden and the hidden-to-output cab be formed the following vector form

$$W_{h,l}^T(t) = [w_{h1l(t)}, w_{h2l}(t), \ldots, w_{hn_il}(t), w_{0,l}(t)] \tag{5.6}$$

$$W_{o,1}^T(t) = [w_{o11(t)}, w_{o21}(t), \ldots, w_{on_h1}(t), w_{o,0}(t)] \tag{5.7}$$

where h and o denote the hidden and output layer, respectively.

Then, the sum of the l-th hidden unit and the output unit are

$$S_{hl}(t) = \phi_h^T(t) W_{hl}(t) \tag{5.8}$$

$$S_{o1}(t) = \phi_o^T(t) W_{o1}(t) \tag{5.9}$$

Secondly, define the errors for the l-th hidden units and the output unit as

$$e_{hl}(t) = S_{hl}^*(t) - S_{hl}(t) \tag{5.10}$$

$$e_o1(t) = S_{o1}^*(t) - S_{o1}(t) \tag{5.11}$$

where $S_{hl}^*(t)$ and $S_{o1}^*(t)$ are the desired values and unknown. So the above errors cannot be calculated. But based on the plant Jacobian and the objective function (19), the errors can be approximated by

$$e_{o1}(t) \cong -\frac{\partial E(t)}{\partial S_{o1}(t)} \cong e(t) J(t) \tag{5.12}$$

where $J(t)$ is the plant Jacobian, it can be calculated in real implementation as

$$J(t) = \frac{\partial F(t)}{\partial u(t-1)} \cong sign(F(t) - F(t-1)) \cdot sign(u(t-1) - u(t-2)) \tag{5.13}$$

The error for the hidden layer can be obtained as

$$e_{hl}(t) \cong -\frac{\partial E(t)}{\partial S_{hl}(t)} \cong e_{o1}(t) w_{ol1}(t) f_{hl}^{'}(t) \tag{5.14}$$

where $f_{hl}^{'}$ is the derivative of the hidden layer activation functions.

Finally the weights of neural controller can be update according to

$$W_{hl}(t) = W_{hl}(t-1) + P_h(t) \phi_h(t) (e_{hl}(t)) \tag{5.15}$$

$$W_{o1}(t) = W_{o1}(t-1) + P_o(t) \phi_o(t) (e_{o1}(t)) \tag{5.16}$$

where P_h and P_o are the P-matrices of the hidden and output layers respectively. The P-matrix is update according to

$$P(t) = \frac{1}{\lambda}P(t-1)\{I_m - \frac{\phi(t)\phi^T(t)P(t-1)}{\lambda + \phi^T(t)P(t-1)\phi(t)}\} \tag{5.17}$$

Experimental Result of Thrust Force Control

The selected neural control scheme in the experiment is the combination of off- and on-line control. The consideration for selecting this control scheme is that a direct on-line neural control scheme may not be appropriate due to the drilling control process is a fast process. The off-line trained neural controller can provide a rough control action in the starting of the drilling process, meanwhile, the on-line neural controller force the plant output to a desired one in an adaptive way by using on-line parameter tuning algorithm.

Off-line training of neural controller

The thrust force neural model have been obtained in Section 5.1.3. In the off-line training stage of neural controller, the plant is replaced by the neural model of the thrust force in order to get the proper weight setting of the neural controller.

Based on above recursive least square training algorithm, the neural controller can be trained to control the thrust force following a desired thrust force. And then the weights of trained neural controller are saved to a file for the later use of on-line control.

On-line control of thrust force

In the on-line control stage, the trained thrust force neural controller is copied and embed in the real drilling control system.

The used on-line neural controller has three inputs, which are the reference thrust force, the previous feed rate and thrust force. The output of is current feed rate. The used parameter in neural controller are: $\lambda = 0.99$, $Umin = 0.01$ and $Umax = 0.9$, $P(0) = \alpha * I$, $\alpha = 100$, where I is an identity matrix with a proper order.

A result of the proposed control scheme is shown in Fig. 5.9. Form the experimental result, it can be seen that the proposed neural controller can regulate the thrust force to desired one.

5.1.5 Section Summary

In this section, we proposed the estimation and control methods of the thrust force in the drilling system by using neural network technique. The effectiveness of the proposed

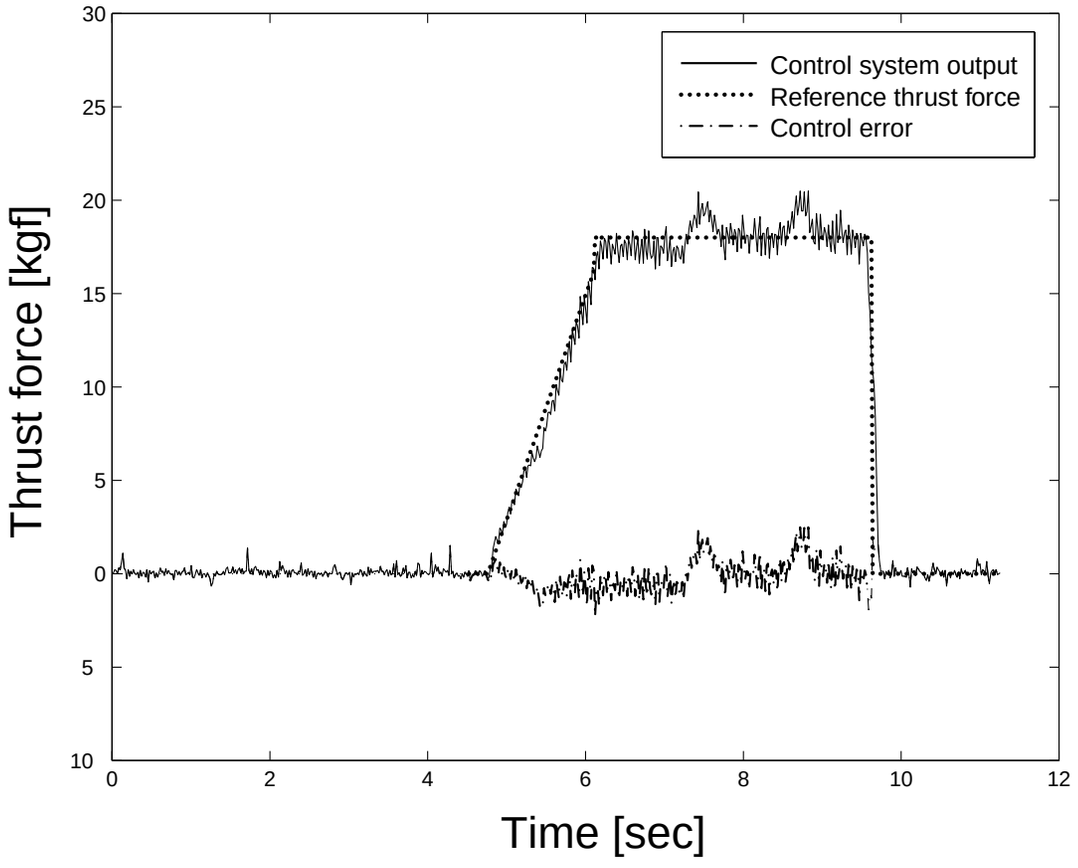


Figure 5.9: The proposed control system output and reference thrust force: the spindle speed is the random value at interval [1200, 2000] [mm/rev]

estimation method of the thrust force was confirmed by comparing the estimated value with the measured value by the thrust force sensor. Based on the neural thrust force model, a two stages method is used to design a neural thrust force controller. The experimental result shown that the proposed neural controller can successfully force the plant output follow the reference signal.

5.2 Identification of Cutting Torque in the Drilling Process

5.2.1 Introduction

Under the assumption that the rotational axis of the spindle system is rigid , the dynamics of the spindle system during cutting can be represented as

$$J\ddot{\theta}_s + D\dot{\theta}_s = K_T i_q - T_l \tag{5.18}$$

Table 5.1: Cutting conditions

Work piece	S45C
Drill	HSS Drill (ϕ2, L=56 [mm])
Sampling time	6 [msec]
Spindle speed	$1200 + 800 * rand()/rand\_max[rpm]$
Feed rate	$0.01 + 0.07 * rand()/rand\_max[mm/rev]$
Hole depth	6 [mm]

where i_q is the active current and calculated by so called d_q-transformation from a three phase alternating current. T_l denotes the cutting torque and $\dot{\theta}_s$ the angular velocity of the spindle motor. J and D are the inertia around and the viscous friction coefficient of the rotational axis respectively. K_T is the torque constant.

The objective of this research is modeling the cutting torque by using the PIPE and the recurrent fuzzy neural networks. Based on above physical model of the cutting process, the signals of the active current i_q and angular velocity of the spindle motor $\dot{\theta}_s$ are used for modeling the cutting torque by using PIPE and recurrent fuzzy neural networks.

The used PIPE and RFNN have been discussed as previous Chapter. So here we give the simulation and experimental results only.

5.2.2 Experiments

Data Pre-processing

In order to obtain a general cutting torque model at different dynamic cutting conditions, we set the spindle speed and feed rate as uniformly random values at pre-defined ranges. The spindle speed varied at interval 1200-2000 [rpm], and the feed rate varied at interval 0.01-0.08 [mm/rev]. The experimental setup of cutting conditions is shown in Table 5.1.

The signals of the active current, the velocity of spindle motor and the current cutting torque during cutting process are measured and send to the computer for the later use in the training of the PIPE and the RFNN.

The training data are rescaled within the range $0.15 - 0.85$ in order to speeding up the convergence speed of the training:

$$y_s = 0.7\frac{y - y_{min}}{y_{max} - y_{min}} + 0.15. \tag{5.19}$$

Results of Cutting Torque Estimation by PIPE

The parameters used in PIPE run are shown in Table.2. The used instruction sets are $I = \{+,-,*,\%,\sin,\cos,\exp, x_1, x_2, x_3, x_4, x_5\}$.

The input vector is $x = [x1, x2, x3, x4, x5]$, where x_1, x_2, x_3, x_4, x_5 are the one-step-ahead active current, the active current, the one-step-ahead velocity of the spindle motor, the velocity of the spindle motor, and the one-step-ahead cutting torque. The out put is the cutting torque.

The best individual is returned at generation 324500 with fitness: 4.211524. The evolved symbolic expression is:

```
(% (+ 0.143182 x4)(exp (* (sin (* x1(+ (% x2(% x3(+ x4 x0)))0.002242)))(
cos (% (+ (exp (+ (sin (% 0.337986(- x1 0.290509)))(cos x0)))(cos (%  x0
x3)))(- (% (% (- (+ x0 (sin x2))(- (- (sin(- x3 x3)) x0) x0))(% (- x2 x3)
x1))(+ (* 0.646081(* (sin (+ (- (* (cos (- (cos (% (+ 0.863717(cos 0.551
236))(* x1 (sin (+ 0.604877(- x3 x3))))))0.950600))(% (+ x3 (* x2 x3))(%
x1 x4)))(* (+ (sin x3)(- x1(sin (- x0 0.029851))))(cos (exp (* 0.040923(
exp (* (* (exp x0)(+ (exp (exp x3)) x4)) x1)))))))(% (* (% x3 (- x1( sin
0.041383)))x3)(cos x2))))0.569843))0.448670))x4))))))
```

The outputs of real plant and the evolved model for training data set are shown in Fig. 5.10, and the experiment test is shown in Fig. 5.11.

Results of Cutting Torque Estimation by RFNN

The parameters used in PIPE run are also shown in Table 2.1. The used instruction set are $I_0 = \{+_{10}, +_{11}, \ldots, +_{20}\}$, $I_1 = \{+_5\}$ and $I_2 = \{x_1, x_2, x_3, x_4, x_5\}$.

The input vector and the output are same as those described in above section.

The outputs of real plant and the evolved model for training data set are shown in Fig. 5.12, and the experiment test is shown in Fig. 5.13.

5.2.3 Section Summary

In this research, two estimation methods of cutting torque in the drilling process using PIPE and RFNN are proposed. The effectiveness of the proposed estimation method of the cutting torque was confirmed by comparing the estimated value with the measured value by the cutting torque sensor. From this research, it can be seen that the cutting torque can be estimated by either PIPE or RFNN with the properly pre-selected input signals.

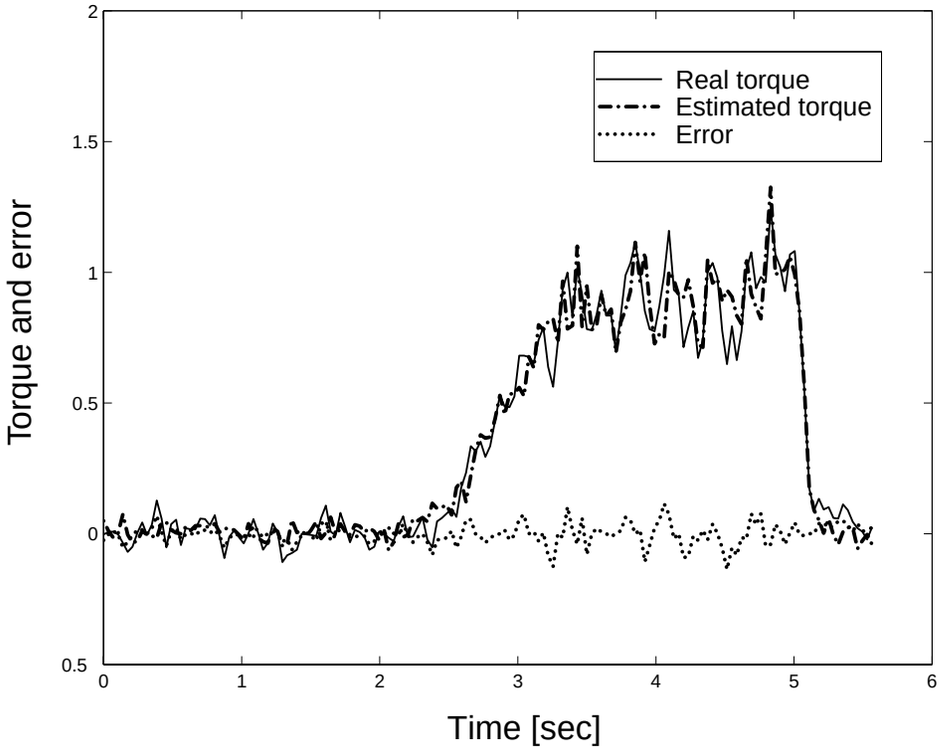


Figure 5.10: Cutting torque of the real plant and the PIPE model

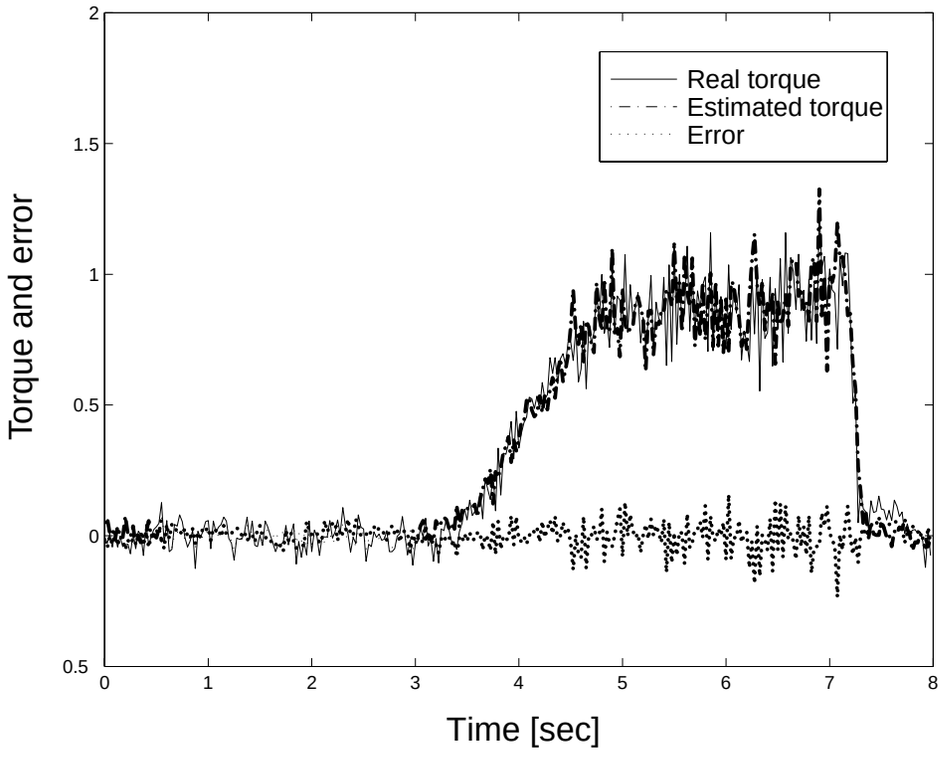


Figure 5.11: Test: Cutting torque of the real plant and the PIPE model by randomly re-setting the spindle speed and feed rate

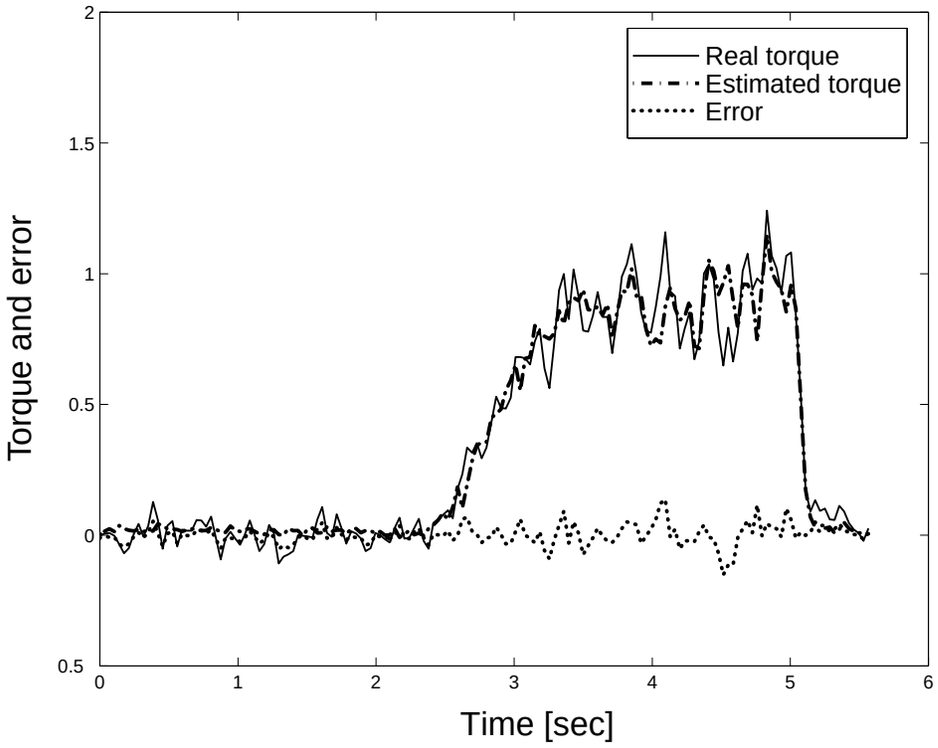


Figure 5.12: Cutting torque of the real plant and the RFNN model

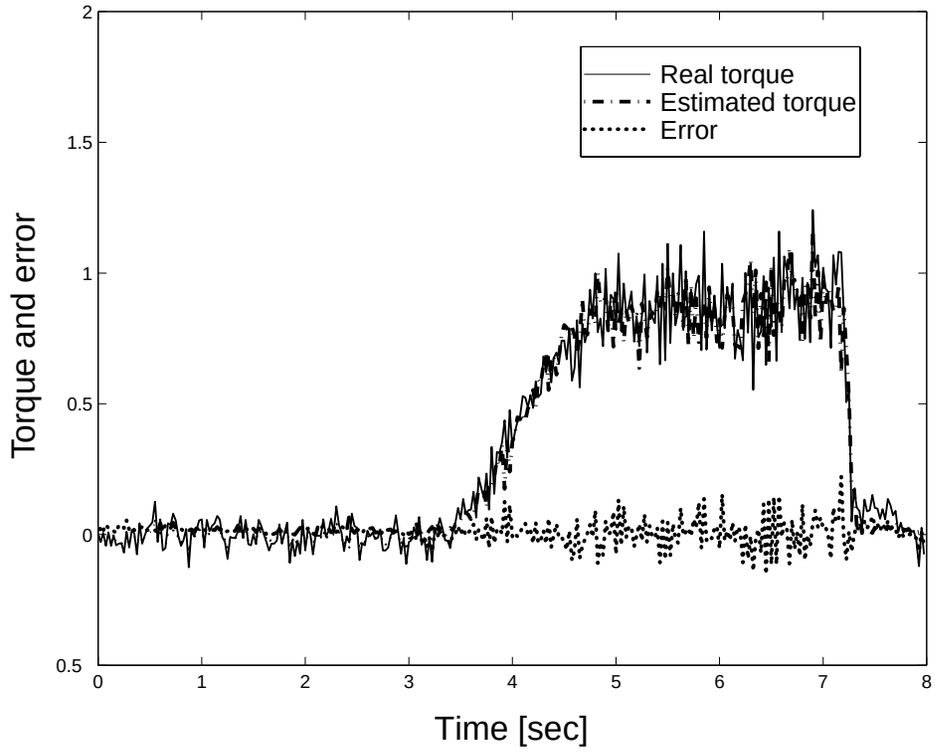


Figure 5.13: Test: Cutting torque of the real plant and the RFNN model by randomly re-setting the spindle speed and feed rate

Chapter 6

Conclusions

6.1 Conclusions

Soft computing approaches have been developed and applied to many scientific and engineering areas in recent years. There have also many successful researches for the identification and control of nonlinear systems by using various soft computing techniques with different computational architecture. The experiments gained over the past decade indicate that it can be more effective to use the various soft computing models in a combined manner. But there is no common recognition about how to combine them in an effective way, and a unified framework of hybrid soft computing models in which various soft computing models can be developed, evolved and evaluated has not been established.

In this research, a unified framework of hybrid soft computing models is proposed and it is applied to the identification and control of nonlinear systems. Under this framework, a number of soft computing models, i.e., the additive model, the non-regular MLP neural networks, the basis function networks and the hierarchical T-S fuzzy models can be evolved and optimized.

The key points of this technique are those as follows:

- Almost of all the soft computing models can be represented and calculated by the type constrained sparse tree with pre-specified data structure which is attached to the node of the tree.

- In this sense, the soft computing models are created automatically not pre-designed, therefore, the difficulties in determining of the architecture of soft computing models can be avoided to some extend.

- It is also very important to mention that based on this idea the architecture and parameters used in the hybrid soft computing models can be evolved and optimized

by using proper learning algorithm, i.e., the architecture of the hybrid soft computing models can be evolved by MPIPE or genetic programming and the parameters can be optimized by many parameter optimization techniques.

The evolved hybrid soft computing models have some advantages, i.e., an almost true polynomial model can be obtained by using the approach of evolutionary induction of additive model, the evolved flexible MLP neural network has non-regular architecture and more efficient than the usually used MLP networks, the evolved basis function networks have small size and the evolved hierarchical T-S fuzzy models are more flexible for approximating nonlinear systems. The disadvantage of the method is that it need more training time due to the search space increased in order to finding the optimal architecture and optimal parameters of the hybrid soft computing models simultaneously. Simulation and experimental studies have been shown that the proposed soft computing models are effective for the identification of nonlinear systems both for simulated plants and real plants (Drilling machine).

Some of hybrid soft computing model based or assistant adaptive control schemes, i.e., PIPE based evolutionary control and a unified framework of the basis function network based adaptive state feedback control scheme are developed in this research. The effectiveness of the proposed control algorithms are confirmed by simulation studies.

Some soft computing model based or assistant controller design principles are also discussed, i.e., sliding mode control based design and backstepping design. Currently, in these design methods the used soft computing models are neural networks and/or fuzzy systems. Other soft computing models can also be used for designing of these controllers in principle. It is valuable to implement and compare the performances of various soft computing models based control schemes in order to construct a unified framework of hybrid soft computing based or assistant control scheme finally.

6.2 Recommendations and Challenges

Some recommendations for further research are in order:

- A hybrid training method for designing of hybrid soft computing models has been proposed in Chapter 3, which is a off-line training method in general, possible improvement and generalization of the algorithm for the use of online training are needed.

- In any case, methods for speeding up the training speed of the hybrid soft computing models are needed to be further studied.

- Generalization of soft computing assistant PID controller and the soft computing based direct inverse controller, new control schemes and training algorithms will be developed. In this research, only a few of hybrid soft computing model based or assistant control schemes are implemented. Based on the soft computing based controller design principle discussed in Chapter 4, it is meaningful to evaluate and implement other soft computing based or assistant control schemes in order to gain some experiments for constructing a unified framework of the hybrid soft computing control schemes.

- We have successfully applied our methods to the identification and control of nonlinear systems, e.g., simulated plants and drilling machine. A natural next step is to evaluate the effectiveness of our methods in other industrial plants.

Reference

[1] L.J. Fogel, "Intelligence Through Simulated Evolution - Forty Years of Evolutionary Programming", *A Wiley-Interscience Publication, John Wiley and Sons press*, 1999.

[2] M. Hagan, H. Demuth, and M. Beale, "Neural Network Design", *Boston: PWS publishing*, 1996.

[3] L.A. Zadeth, "Fuzzy logic, neural networks, and soft computing", *Commun, ACM*, Vol.37, pp.77-84, 1994.

[4] Y. Diao and K.M. Passino, "Immunity-based hybrid learning methods for structure and parameter adjustment", *Submitted to IEEE Trans. on Syst., Man, and Cybern.*, July, 2000.

[5] S.A. Billings, "Identification of nonlinear systems–a survey", *IEE Proceedings, Part D, 127*, 272-285, 1980.

[6] H. Haber, and H. Unbehauen, "Structure identification of nonlinear dynamic systems–A survey on input/output approaches", *Automatica*, 26(4), 615-677, 1990.

[7] J. Sjoberg et al., "Nonlinear black-box modeling in system identification: a unified overview", *Automatica*, 31(12), 1691-1724, 1995.

[8] B. Anass et al., "Nonlinear dynamic system identification: a multi-model approach", *Int. J. of Control*, Vol.72, No.7/8, 591-604, 1999.

[9] S.A. Billings, "Neural networks and system identification", In K. Warwick et al. (Eds), *Neural networks and system control*, London: peter Peregrinus, 181-205, 1992.

[10] S. Chen S.A. Billings, "Nonlinear system identification using neural networks", *Int. journal of Control*, Vol.51, No.6, 1191-1214, 1990.

[11] M.J. Willis et al., "Artificial neural networks in process estimation and control", *Automatica*, Vol.28, No.6, 1181-1187, 1992.

[12] S.Z. Qin et al., "Comparison of four net learning methods for dynamic system identification", *IEEE Trans. on Neural Networks*, Vol.3, No.1, 122-130, 1992.

[13] K. S. Narendra et al., "Identification and Control of Dynamic System using Neural Networks", *IEEE Trans. on Neural Networks*, Vol.1, No. 2, Jan. 4-27, 1990.

[14] J.G. Kuschewski et al., "Application of feedforward neural networks to dynamic system identification and control", *IEEE Trans. Control Systems and Technology*, Vol.1, No.1, 37-49, 1993.

[15] Q. Song, "Robust training algorithm of multiplyered neural networks for identification of nonlinear dynamic systems", *IEE Proc. -Control Theory Appl.*, Vol.145, No.1, 41-46, 1998.

[16] B.K. Elias, "High-order neural network systems in the identification of dynamic systems", *Control and Dynamic Systems*, Academic Press, 279-305, 1998.

[17] G.P. Liu, V. Kadirkamanathan and Steve A. Billings, "On-line identification of nonlinear systems using Volterra polynomial basis function neural networks", *Journal of Neural Networks*, Vol.11, 1645-1657, 1998.

[18] G.P.Liu and V. Kadirkamanathan, "Multiobjective criteria for neural network structure selection and identification of nonlinear systems using genetic algorithms", *IEE Proc.-Control Theory Appl.*, Vol.146, No.5, 373-382, 1999.

[19] C.P. Jagdish et al., "Identification of nonlinear dynamic systems using functional link artificial neural networks", *IEEE Trans. on Syst. Man and Cybn.*, Vol.29, No.2, 254-262, 1999.

[20] L. Chun-Shin et al., "A sum-of-product neural network (SOPNN)", *Neurocomputing*, Vol.30, 273-291, 2000.

[21] D.T. Pham et al., "Training Elman and Jordan networks for system identification using genetic algorithms", *Artificial Intelligence in Engineering*, Vol.13, 107-117, 1999.

[22] D.T. Pham et al., "Identification of plant inverse dynamics using neural networks", *Artificial Intelligence in Engineering*, Vol.13, 309-320, 1999.

[23] L. Chun-Shin et al., "Nonlinear system identification using a Bayesian-Gaussian neural network for predictive control", *Neurocomputing*, Vol.28, 21-36, 1999.

[24] M. Onder Efe et al., "A comparative study of soft-computing methodologies in identification of robotic manipulators", *Robotics and Autonomous Systems*, Vol.30, 221-230, 2000.

[25] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its application to modeling and control", *IEEE Trans. Syst. Man, Cybern.*, Vol.15, 116-132, 1985.

[26] C. Xu and Y. liu, "Fuzzy model identification and self learning for dynamic systems", *IEEE Trans. on Syst. Man, Cybern.*, Vol.17, 683-689, 1987.

[27] S. Horikawa et al., "On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm", *IEEE trans. on Neural Networks*, Vol.3, No.5, 801-806, 1992.

[28] H.R. Berenji et al., "Fuzzy rules for guiding reinforcement learning", *Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Mallorca, 511-514, 1992.

[29] J.S. Jang et al., "Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence", Upper Saddle River, NJ:Prentice-Hall, 1997.

[30] R. Marco, "FuGeNeSys- A fuzzy genetic neural system for fuzzy modeling", *IEEE Trans. on Fuzzy Systems*, Vol.6, No.3, 373-388, 1998.

[31] S. Yuhui et al., "Implementation of evolutionary fuzzy systems", *IEEE Trans. on Fuzzy Systems*, Vol.7, No.2, 109-119, 1999.

[32] L. Chuan-Kai et al., "Fuzzy system identification using an adaptive learning rule with terminal attrctors", *Fuzzy Sets and Systems*, Vol.101, 343-352, 1999.

[33] S. Barada etal., "Generating Optimal Adaptive Fuzzy-Neural Models of Dynamical Systems with Applications to Control", *IEEE Trans. on SMC*, Vol.28, No.3, 371-390, 1998.

[34] N. Kasabov, "Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems", *Fuzzy sets and Systems*, Vol.82,No.2, 2-20, 1996.

[35] N. Kasabov et al., "FuNN/2- A Fuzzy Neural Network Architecture for Adaptive Learning and Knowledge Acquisition", *Information Science - Applications*, Vol.101, No.3/4, 155-175, 1997.

[36] N Kasabov et al., "Rule Insertion and Rule Extraction from Evolving Fuzzy Neural Networks: Algorithms and Applications for the Building Adaptive, Intelligent Expert Systems", *Proc. of Int. Conf. FUZZY-IEEE*, Seoul, 1999.

[37] K. Kristinn et al., "System identification and control using genetic algorithms", *IEEE Trans. on Systems, Man and Cybernetics*, Vol.22, No.5, 1033-1046, 1992.

[38] H.-X. Li, "Identification of Hammerstain models using genetic algorithms", *IEE Proc. -Control Theory Appl.*, Vol.146, No.6, 499-503, 1999.

[39] H.W. Andrew et al., "System Identification using Genetic Programming", *Proc. of ACEDC'96*, PEDC, University of Plymouth, UK, 1996.

[40] Y. Chen and S. Kawaji. Identification and Control of Nonlinear System using PIPE Algorithm. *Proc. of Workshop on Soft Computing in Industry'99 (IWSCI'99)*, Muroran, 60-65, 1999.

[41] Y. Chen and S. Kawaji. Evolutionary Control of Discrete-Time Nonlinear System using PIPE Algorithm. *Proc. of IEEE Int. Conf. on Systems, Man and Cybernetics (SMC'99)*, Tokyo, IV, 1078-1083, 1999.

[42] P. J. Angeline and D. B. Fogel, "An Evolutionary Program for the Identification of Dynamical System", *Proc. of SPIE (Volume 3077): Application and Science of Artificial Neural Networks III, S.Rogers (ed.), SPIE-The International Society for Optical Engineering*, Bellingham, WA., 409-417, 1997.

[43] Y. Pati et al., "Rational wavelets in model reduction and system identification", *Proc. IEEE Conf. on Decision and Control*, Lake Buena Vista, FL, 3394-3399, 1994.

[44] Q. Zhang et al., "Wavelet networks", *IEEE Trans. Neural Networks*, Vol.3, NO.6, 889-898, 1992.

[45] J. Zhang et al., "Wavelet neural networks for function learning", *IEEE Trans. Signal Process*, Vol.43, No.6, 1485-1497, 1995.

[46] A. Judisky et al., "Wavelets in Identification", *Fuzzy Logic and Experts Systems Applications*, Academic Press, 315-413, 1998.

[47] S. Rafal and S. Jurgen. Probabilistic Incremental Program Evolution. *Evolutionary Computation*, Vol.5, No.2, 123-141, 1997.

[48] J. Hu, et. al.. RasID - Random Search for Neural Network Training. *Journal of Advanced Computational Intelligence*, Vol.2, No.2, 134-141, 1998.

[49] N. Sureshbabu and J. Farrell, "Wavelet based system identification for nonlinear control applications", *IEEE Trans. Automatic Control*, Vol. 44, No.2, 412-417, 1999.

[50] Y. Oussar et al., "Training wavelet networks for nonlinear dynamic input-output modeling", *Neurocomputing*, Vol.20, 173-188, 1998.

[51] K.M. Bossley, "Neurofuzzy Modeling Approaches in System Identification", *PhD thesis*, University of Southampton, 1997.

[52] Y. Shi et al., "A new approach of neuro-fuzzy learning algorithm for tuning fuzzy rules", *Fuzzy Sets and Systems*, Vol.112, 99-116, 2000.

[53] Y. Shi et al., "Some considerations on conventional neuro-fuzzy learning algorithms by gradient descent method", Fuzzy Sets and Systems, Vol.112, 51-63, 2000.

[54] X. Hong and S.A. Billings, "Parameter estimation based on stacked regression and evolutionary algorithms", *IEE Proc.-Control Theory Appl.*, Vol.146, No.5, pp.406-414, 1999.

[55] P.J. Gregor et al., "Combinational evolution of regression nodes in feedforward neural networks", *Neural Networks*, Vol.12, 175-189,1999.

[56] S. Chen et al., "Combined genetic algorithm optimization and regularized orthogonal least square learning for radial basis function networks", *IEEE Trans. Neural Networks*, Vol.10, No.5, 1239-1243, 1999.

[57] B.-T. Zhang et al., "Evolutionary Induction of Sparse Tree", *Evolutionary Computation*, Vol.5, No.2, 213-230, 1997.

[58] Piero P. Bonissone et al., "Hybrid Soft Computing Systems: Industrial and Commercial Applications", *Proceedings of IEEE*, Vol.87, No. 9, September, 1999.

[59] Q. Gan and C. J. Harris, "Fuzzy local linearization and logic basis function expansion in nonlinear system modeling", *IEEE Trans. on Systems, Man, and Cybernetics-Part B*, Vol.29, N0.4, 1999.

[60] D.E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", *Addison-Wesley Publishing Company, Inc,*, 1989.

[61] J. Hu et al., "RasID- Random Search for Neural Network Training", *Journal of Advanced Computational Intelligence*, Vol.2, No.4, pp.134-141, 1998.

[62] J. Martyas, "Random Optimization", *Automation and Remote Control*, Vol.28, pp.244-251, 1965.

[63] F.J. Solis et al., "Minimization by random search techniques", *Mathematics of Operations Research*, Vol.6, PP.19-30, 1981.

[64] W.S. McCulloch and W. Pitts, "A Logical Calculus of the ideas immanent in the nervous activity", *Bull. Math. Biophys.*, Vol.5, 115, 1943.

[65] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain", *phych. Rev.*, Vol.65, 1958.

[66] M. Minsky and S. Papert, "Perceptron: An introduction to computational geometry", *Cambridge, MA: MIT Press*, 1969.

[67] K. Steinbush, "The Learning Matrix", *Kybernetik (Biol. Cybern.)*, pp.36-45, 1961.

[68] S.I. Amari, "Characteristics of randomly connected threshold-element network system", *Proc. of IEEE*, Vol.59, pp.35-47, 1971.

[69] K. Fukushima, "Cognition: A self-organizing multilayer neural network", *Biol. Cybern.*, Vol.20, pp.121-136, 1975.

[70] J.J. Hopfield, "neural network and physical systems with emergent collective computational abilities", *Proc. Nat. Acad. Sci. USA*, Vol.79, pp.2554-2558, 1982.

[71] M. Moreira et al., "Neural networks with adaptive learning rate and momentum terms", *Technique Report 95-04*, IDIAP, Martigny, Switzerland, October, 1995.

[72] S.E. Fahlman, "Fast-learning variations on back-propagation: an empirical study", *Proc. of the 1988 Connectionist Models Summer School*, Pittsberg, pp. 38-51, 1988.

[73] M. Riedmiller at al., "A direct adaptive method for faster backpropagation learning : the RPROP algorithm", *Proc. of the IEEE Int. Conf. on Neural Networks*, pp. 586-591, San Francisco, 1993.

[74] J.K. Kruschke, et al., "Benefits of gains: speeded learning and minimal hidden layers in backpropagation networks", *IEEE Trans. on Systems, Man, and Cybernetics*, Vol.21, No.1, pp.273-280, 1991.

[75] R.A. Jacobs, "Increase rates of convergence through learning rate adaptation", *Neural networks*, Vol.1, pp.295-307, 1988.

[76] G.D. Magoulas at al., "Improving the convergence of the backpropagation algorithm using learning rate adaptation methods", *Neural Computation*, Vol.11, pp.1769-1796, 1999.

[77] St. Maruster, "The stability of gradient-like methods", *Applied Mathematics and Computation*, Vol.117, pp.103-115, 2001.

[78] M. Onder and K. Okyay, "Stable training of computationally intelligent systems by using variable structure systems technique", *IEEE Trans. on Industrial Electronics*, Vol.47, No.2, pp.487-496, 2000.

[79] M. Onder and K. Okyay, "On stability of gradient-based training strategies for computationally intelligent systems", *IEEE Trans. on Fuzzy Systems*, Vol.8, No.5, pp.564-575, 2000.

[80] C.C. Chuang et al., "The annealing robust backpropagation (ARBP) learning algorithm", *IEEE Trans. on Neural Networks*, Vol.11, No.5, 2000.

[81] J. Bilski et al., "A fast training algorithm for neural networks", *IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing* , Vol.45, No.6, pp.749-753, June, 1998.

[82] G.W.Ng, "Applications of neural networks to adaptive control of nonlinear systems", *Research Studies Press Ltd.*, 1997.

[83] L.M. Sachenberger et al., "Neural networks: a new tool for predicting thrift failures", *Decision Science*, Vol.23, pp.899-916, 1992.

[84] X. Yao, "Evolving artificial neural networks", *Proceedings of IEEE*, Vol.87, pp.1423-1447, 1999.

[85] L. David, "Handbook of genetic algorithms", *Van Nostrand Reinhold*, New Yeak, 1991.

[86] Z. Michalewicz, "Genetic algorithm+Data Structure=Evolutionary Programming", *Springer*, Berlin, 1992.

[87] D.B. Fogel and A. Ghozeil, "A note on representation and variation operators", *IEEE Trans. on Evolutionary Computation*, Vol.1, pp.159-161, 1997.

[88] D.J. Montana et al., "Training feedforward neural networks using genetic algorithms", *Proc. of the Third Int. Conf. on Genetic Algorithms*, San Mateo, pp. 379-384, 1989.

[89] R.S. Sexton et al., "Toward global optimization of neural networks: a comparison of genetic algorithm and backpropagation", *Decision Support Systems*, Vol.22, pp.171-186, 1998.

[90] V.W. Porto et al., "Alternative neural network training methods", *IEEE Expert*, Vol.10, pp.16-22, 1995.

[91] N.Saravanan et al., "A comparison of methods for self-adaptation in evolutionary algorithms", *BioSystems*, Vol.36, pp.157-166, 1995.

[92] R.S. Sexton etal., "Comparative evaluation of genetic algorithm and backpropagation for training neural networks", *Information Science*, Vol.129, pp.45-59, 2000.

[93] Y. Tan et al., "Neural network based direct optimizing predictive control with on-line PID gradient optimization", *Journal of Intelligent Automation and Soft Computing*, 2000.

[94] G.P. Gustavo et al., "Neural networks learning with sliding mode control: the sliding mode backpropagation algorithm", *International Journal of Neural Systems*, Vol.9, No. 3, pp.187-193, 1999.

[95] J.J. Buckley, "Sugeno type controller are universal controller", *Fuzzy Sets and Systems*, Vol.53, pp.299-303, 1993.

[96] J.J. Buckley et al., "Fuzzy input-output controller are universal approximators", *Fuzzy Sets and Systems*, Vol.58, pp.273-278, 1993.

[97] S.G. Cao et al., "Fuzzy modeling and identification for a class of complex dynamic systems", *Proc. of the Pacific-Asia conf. on Expert Systems*, Huangshan, China, pp.212-217, 1995.

[98] S.G. Cao et al., "Anakysis and design of uncertain fuzzy control systems. Part I: Fuzzy modeling and identification", *Proc. of the Fifth IEEE Int. Conf. on Fuzzy Systems*, New OrleansVol.1, pp.640-646, 1996.

[99] H. Ying, "General Takagi-Sugeno fuzzy systems with simplified linear rule consequent are universal controllers models and filters", *Information Science*, Vol.108, pp.91-107, 1998.

[100] H. Ying, "Theory and application of a novel fuzzy PID controller using a simplified Takagi-Sugeno rule scheme", *Information Science*, Vol.123, pp.281-293, 2000.

[101] C. Xu and Y. liu, "Fuzzy model identification and self learning for dynamic systems", *IEEE Trans. on Syst. Man, Cybern.*, Vol.17, pp.683-689, 1987.

[102] Q. Gan and C.J. Harris, "Fuzzy local linearization and logic basis function expansion in nonlinear system modeling", *IEEE Trans. on Systems, Man, and Cybernetics-Part B*, Vol.29, N0.4, 1999.

[103] S. Yuhui et al., "Implementation of evolutionary fuzzy systems", *IEEE Trans. Fuzzy Systems*, Vol.7, No.2, pp.109-119, 1999.

[104] C-K. Lin and S-D Wang, "Fuzzy system identification using an adaptive learning rule with terminal attractors", *Journal of Fuzzy Sets and Systems*, pp.343-352, 1999.

[105] K. Sin-Jin et al., "Evolutionary design of fuzzy rule base for nonlinear system modeling and control", *IEEE Trans. Fuzzy Systems*, Vol.8, No.1, pp.37-45, 2000.

[106] H. Yo-Ping et al., "Designing a fuzzy model by adaptive macroevolution genetic algorithms", *Fuzzy Sets and Systems*, Vol.113, pp.367-379, 2000.

[107] W. Baolin et al., "Fuzzy modeling and identification with genetic algorithm based learning", *Fuzzy Sets and Systems*, Vol.113, pp.352-365, 2000.

[108] D. Maurizio et al., "Learning fuzzy rules with tabu search-an application to control", *IEEE Trans. on Fuzzy Systems*, Vol.7, No.2, pp.295-318, 1999.

[109] K. Balakrishnan et al.. Evolutionary design of neural architectures. *Tech.Rep. No. CS-TR-95-01*, Ames, IA: Iowa State University, Department of Computer Science, Artificial Intelligence Laboratory, 1995

[110] R.K. Belew et al.. Evoving networks: using genetic algorithms with connectionist learning. *In C.G. Langton, C. Taylor, J.D. Farmer, & S. Rasmussen (Eds.), Artificial Life II, SFI studies in the science of complexity*, Vol X. Reading, MA: Addison-Wesley, 1991

[111] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. em Complex Cybernetics 4, 1990:461-476

[112] F. Gruau. Genetic synthesis of modular neural networks. *In S. Forrest (Eds.), Proc. of the Fifth Int. Conf. on Genetic Algorithms*, 1993:318-325

[113] Farrell, J and M. Polycarpou, "Neural, Fuzzy, and On-line Approximation Based Control", In T. Samad, (ed.), Perspectives in Control: New Concepts and Applications, IEEE Press. Accepted February 1999. [Invited Chapter], In Press.

[114] J. Farrell, "On Performance Evaluation in On-line Approximation for Control", IEEE Transactions on Neural Networks, Vol.9, No.5, pp. 1001-1007, 1998.

[115] K.M. Bossley et al., "Neurofuzzy high-dimensional modeling", In Alfred Walle, editor, Adaptive Computing: Neural Networks, pp.297-332, 1995.

[116] C.J. Harris et al., "Neurofuzzy state estimators and its applications", Annual Reviews in Control, Vol.23, pp.149-158, 1999.

[117] C. Xu and Y. liu, "Fuzzy model identification and self learning for dynamic systems", *IEEE Trans. on Syst. Man, Cybern.*, Vol.17, pp.683-689, 1987.

[118] Q. Gan and C.J. Harris, "Fuzzy local linearization and logic basis function expansion in nonlinear system modeling", *IEEE Trans. on Systems, Man, and Cybernetics-Part B*, Vol.29, N0.4, 1999.

[119] S. Yuhui et al., "Implementation of evolutionary fuzzy systems", *IEEE Trans. Fuzzy Systems*, Vol.7, No.2, pp.109-119, 1999.

[120] C-K. Lin and S-D Wang, "Fuzzy system identification using an adaptive learning rule with terminal attractors", *Journal of Fuzzy Sets and Systems*, pp.343-352, 1999.

[121] K. Sin-Jin et al., "Evolutionary design of fuzzy rule base for nonlinear system modeling and control", *IEEE Trans. Fuzzy Systems*, Vol.8, No.1, pp.37-45, 2000.

[122] H. Yo-Ping et al., "Designing a fuzzy model by adaptive macroevolution genetic algorithms", *Fuzzy Sets and Systems*, Vol.113, pp.367-379, 2000.

[123] W. Baolin et al., "Fuzzy modeling and identification with genetic algorithm based learning", *Fuzzy Sets and Systems*, Vol.113, pp.352-365, 2000.

[124] D. Maurizio et al., "Learning fuzzy rules with tabu search-an application to control", *IEEE Trans. on Fuzzy Systems*, Vol.7, No.2, pp.295-318, 1999.

[125] G.V.S. Raju et al., "Hierarchical fuzzy control", *Int. J. Contr.*, Vol.54, No.5, pp.1201-1216, 1991.

[126] L.X. Wang, "Analysis and design of hierarchical fuzzy systems", *IEEE Trans. Fuzzy Systems*, Vol.7, No.5, pp.617-624, 1999.

[127] L.X. Wang, "Universal approximation by hierarchical fuzzy systems", *Fuzzy Sets and Systems*, Vol.93, pp.223-230, 1998.

[128] O. Huwendiek et al., "Function approximation with decomposed fuzzy systems", *Fuzzy Sets and Systems*, Vol.101, pp.273-286, 1999.

[129] K. Hiroaki et al., "Functional completeness of hierarchical fuzzy modeling", *Information Science*, Vol.110, No.1-2, pp.51-60, 1998.

[130] H. Rainer, "Rule generation for hierarchical fuzzy systems", *Proc. of the annual conf. of the North America Fuzzy Information Processing*, pp.444-449, 1997.

[131] K. Sun-Yuan et al., "Synergistic modeling and applications of hierarchical fuzzy neural networks", *Proceedings of IEEE*, Vol.87, No.9, 1550-1574, 1999.

[132] M. Brown et al., "High dimensional neurofuzzy systems: overcoming the curse of dimensionality", *Proc. 4th Int. Conf. on Fuzzy Systems*, pp.2139-2146, 1995.

[133] C. Wei and Li-Xin Wang, "A note on universal approximation by hierarchical fuzzy systems", *Information Science*, Vol.123, pp.241-248, 2000.

[134] L. C. Lin et al., "Hierarchical fuzzy control for C-axis of CNC tuning centers using genetic algorithms", *Journal of Intelligent and Robotic Systems*, Vol.25, No.3, pp.255-275, 1999.

[135] S. Rafal and S. Jurgen, "Probabilistic Incremental Program Evolution", Evolutionary Computation, Vol.2, No.5, pp. 123-141, 1997.

[136] Y. Chen and S. Kawaji, "Identification and Control of Nonlinear System using PIPE Algorithm", *Proc. of Workshop on Soft Computing in Industry'99 (IWSCI'99)*, Muroran, pp.60-65, 1999.

[137] Y. Chen and S. Kawaji, "Evolutionary Control of Discrete-Time Nonlinear System using PIPE Algorithm", *Proc. of IEEE Int. Conference on Systems,Man and Cybernetics*, pp.1078-1083, Tokyo, 1999.

[138] Y. Chen and S. Kawaji, "Evolving Wavelet Neural Networks for System Identification", *Proc. of The International Conference on Electrical Engineering (ICEE'2000)*, pp.279-282, Kitakyushu, 2000.

[139] Y. Chen and S. Kawaji, "Evolving ANNs by Hybrid Approaches of PIPE and Random Search algorithm for System Identification", *Proc. of The Third Asian Control Conference (ASCC'2000)*, pp. 2206-2211, Shanghai, July, 2000.

[140] D.T. Pham et al., "Training Elman and Jordan Networks for System Identification using genetic algorithms", *Artificial Intelligence in Engineering*, Vol.13, pp.107-117, 1999.

[141] D.B. Fogel, K. Chellapilla, and P.J. Angeline, "Inductive Reasoning and Bounded Rationality Reconsidered", *IEEE Transactions on Evolutionary Computation*, Vol. 3:2, pp. 142-146, 1999.

[142] D.B. Fogel, E.C. Wasson, and E.M. Boughton, "Evolving Neural Networks for Detecting Breast Cancer", *Cancer Letters*, Vol. 96, pp. 49-53, 1995.

[143] P.J. Angeline, "A Historical Perspective on the Evolution of Executable Structures", *Fundamenta Informaticae*, 36 (1-4), August, pp. 179-195, 1998.

[144] X. Yao, "Evolving artificial neural networks", *Proceedings of the IEEE*, 87(9), pp. 1423-1447, September, 1999.

[145] X. Yao, Y. Liu and G. Lin, "Evolutionary programming made faster", *IEEE Transactions on Evolutionary Computation*, 3(2), pp. 82-102, July 1999.

[146] H. Ying, "Theory and application of a novel fuzzy PID controller using a simplified Takagi-Sugeno rule scheme", *Information Science*, Volume 123, Issues 3-4, pp. 281-293, 2000.

[147] F.-C.Chen et al., "Adaptive control of a class of nonlinear discrete time system using neural networks", *IEEE Trans. Automat. Contr.*, Vol.40, No.5, pp.791-801, 1995.

[148] F.L.Lewis et al., "Neural net robot controller with guaranteed tracking performance", *IEEE Trans. Neural Networks*, Vol.6, No.3, pp.703-715, 1995.

[149] G.A.Rovithakis et al., "Adaptive control of unknown plants using dynamical neural networks", *IEEE Trans. Syst., Man, Cybern.*, Vol.24, pp.400-412, 1994.

[150] F.L.Lewis et al., "Neural network control of robot manipulators and nonlinear systems", *Taylor and Francis publisher*, 1999.

[151] Q. Song et al., "Robust backpropagation training algorithm for multilayered neural tracking controller", *IEEE Trans. Neural Networks*, Vol.10, No.5, 1999.

[152] M.J. Willis et al., "Artificial neural networks in process estimation and control", *Automatica*, Vol.28, No.6, pp.1181-1187, 1992.

[153] George A. Rovithakis, "Tracking control of multi-input affine nonlinear dynamical systems with unknown nonlinearities using dynamical neural networks", *IEEE Trans. Syst., Man and Cybern.*, Vol.29, No.2, 1999.

[154] Guoping P. Liu et al., "Variable neural networks for adaptive control of nonlinear systems", *IEEE Trans. Syst., Man, and Cybern.*, Vol.29, No.1, pp.34-43, 1999.

[155] Richard B. Mclain, "Direct adaptive control of partially known nonlinear systems", *IEEE Trans. Neural Networks*, Vol.10, No.3, pp.714-721, 1999.

[156] Jeffrey T. Spooner and Kevin M. Passino, "Decentralized adaptive control of nonlinear systems using radial basis neural networks", *IEEE Trans. Automatic Control*, Vol.44, No.11, pp.2050-2057, 1999.

[157] N. Kwanghee, "Stabilization of feedback linearizable systems using a radial basis function networks", *IEEE Trans. Automatic Control*, Vol.44, No.3, pp.1026-1031, 1999.

[158] R.M. Sanner et al., "Gaussian networks for direct adaptive control", *IEEE Trans. Neural Netw.*, Vol.3, No.6, pp.837-863, 1992.

[159] S. Jagannathan, "Discrete-time CMAC NN control of feedback linearizable nonlinear systems under a persistence of excitation", *IEEE Trans. Neural networks*, Vol.10, No.1, pp.128-137, 1999.

[160] M. A. Brdys et al., "Recurrent networks for nonlinear adaptive control", *IEE Proc.-Control Theory Appl.*, Vol.145, No.2, pp.177-188, 1999.

[161] L. Behera et al., "Application of self-organising neural networks in robot tracking control", *IEE Proc.-Control Theory Appl.*, Vol.145, No.2, pp.135-140, 1999.

[162] Y. Hao, "Analytical analysis and feedback linearization tracking control of the general Takagi-Sugeno fuzzy dynamic systems", *IEEE Trans. Syst., Man, and Cybern.*, Vol.29, No.1, pp.290-298, 1999.

[163] C. Federico et al., "stability analysis of nonlinear multivariable Takagi-Sugeno fuzzy control systems", *IEEE Trans. Fuzzy Systems*, Vol.7, No.5, pp.508-520, 1999.

[164] O. Raul et al., "Stable multi-input multi-output adaptive fuzzy/neural control", *IEEE Trans. Fuzzy Systems*, Vol.7, No.3, pp.345-353, 1999.

[165] L. Behera et al., "Guaranteed tracking and regulatory performance of nonlinear dynamic systems using fuzzy neural networks", *IEE Proc.-Control Theory Appl.*, Vol.146, No.5, pp.484-491, 1999.

[166] L. Yih-Guang et al., "Robust adaptive fuzzy-neural controllers for uncertain nonlinear systems", *IEEE Trans. on Robotics and Automation*, Vol.15, No.5, pp.805-817, 1999.

[167] W. -S.Lin and C.-H.Tsai, "Neurofuzzy-model-following control of MIMO nonlinear systems", *IEE Proc.-Control Theory Appl.*, Vol.146, No.2, pp.157-164, 1999.

[168] L. Yih-Guang et al., "Oberver-based adaptive fuzzy-neural control for unknown nonlinear dynamical systems", *IEEE IEEE Trans. Syst., Man*, and Cybern., Vol.29, No.5, pp.583-591, 1999.

[169] F. Mark and R. Eric, "Input/output stability theory for direct neuro-fuzzy controllers", *IEEE Trans. on Fuzzy Systems*, Vol.6, No.3, pp.331-345, 1999.

[170] S.S. Ge, C.C. Hang and T. Zhang, "Adaptive neural network control of nonlinear systems by state and output feedback", *IEEE Trans. Syst., Man, and Cybern.*, Vol.29, No.6, pp.818-828, 1999.

[171] A. H. Watson et al., "System Identification of Fluid Systems using Genetic Programming", In Proc. of the Second Online Workshop on Evolutionary Computation (WEC2), 45-48, (1996)

[172] B. Howley, "Genetic Programming of Near minimum Time Spacecraft Attitude Manoeuvres", Proc. of the 1st Annual Conference on Genetic Programming, July 28-31, 98-106, (1996).

[173] D.C. Dracopoulos, "Evolutionary Control of a Satellite", Proc. of the 2st Annual Conference on Genetic Programming, July 13-16, 77-81, (1997).

[174] S. Dominic, "Chemical Process Controller Design using Genetic Programming", Proc. of the 3st Annual Conference on Genetic Programming, July 13-16, 77-81, (1998).

[175] K. Chellapilla, "Automatic generation of nonlinear optimal control laws for broom balancing using evolutionary programming", In Proc. of the IEEE World Congress on Computational Intelligence , 195-200, (1998)

[176] P. J. Angeline and David B. Fogel, "An Evolutionary Program for the Identification of Dynamical System", In Proc. of SPIE (Volume 3077): Application and Science of Artificial Neural Networks III, S.Rogers (ed.), SPIE-The International Society for Optical Engineering, Bellingham, 409-417, (1997).

[177] K. S. Narendra et al., "Identification and Control of Dynamic System using Neural Networks", IEEE Trans. on Neural Networks, 1, 4-27, Jan. (1990).

[178] C-H. Lee, C-C. Teng, "Identification and Control of Dynamic Systems using Recurrent Fuzzy Neural Networks", IEEE Trans. on Fuzzy Systems, Vol.8, No.4, pp.349-366, Augest, 2000.

[179] B.Y. Lee et al., "Cutting-parameter selection for maximizing production rate or minimizing production cost in multistage tuning operations", *Journal of Materials Processing Technology*, Vol.105, pp.61-66, 2000.

[180] T.J. Ko et al., "Autonomous cutting parameter regulation using adaptive modeling and genetic algorithms", *Precesion Engineering*, Vol.22, No.4, pp.243-251, 1998.

[181] C.R. Peres et al., "Fuzzy model and hierarchical fuzzy control integration: an approach for milling process optimization", *Computer in Industrial*, Vol.39, pp.199-207, 1999.

[182] A. Ghasempoor et al., "Real time implementation of on-line tool condition monitoring in turning", *Int. J. of Machine Tools and Manufacture*, Vol.39, pp.1883-1902, 1999.

[183] S. Kawaji et al., "Estimation of cutting torque using disturbance observer", *Proc. of Japan-U.S.A. Symposium on Flexible Automation-Aa pacific Rim Conference*, pp.249-252, 1994.

[184] S. Kawaji et al., "Control of cutting torque in the drilling process using disturbance observer", *Proc. of the American Control Conference*, Vol.1, pp.723-728, 1995.

[185] Yung-Chin Chang et al., "Cutting force estimation of spindle motor", *Journal of Control Systems and Technology*, Vol.3, No.2, pp. 145-152, 1995.

[186] R.J. Kuo, "Multi-sensor integration for on-line tool wear estimation through artificial neural networks and fuzzy neural network", *Engineering Application of Artificial Intelligence*, Vol.13, pp.249-261, 2000.

[187] Q. Liu et al., "On-line monitoring of flank ware intuning with multilayered feedforward neural network", *Int. J. of Machine Tools and Manufacture*, Vol.39, PP.1945-1959, 1999.

[188] J.H. Lee et al., "One-step-ahead prediction of flank ware using cutting force", *Int. J. of Machine Tools and Manufacture*, Vol.39, PP.1747-1760, 1999

[189] L.K. Danshmend at al., "Model reference adaptive control of feed force in tuning", *ASME J. Dyn. Syst. Meas., Control*, Vol.108, No.3, pp.215-222, 1986.

[190] M.A. Elbestwi et al., "Application of some parameter adaptive control algorithm in machining", *ASME J. Dyn. Syst. Meas., Control*, Vol.112, No.4, pp.611-617, 1990.

[191] M.A. Elbestwi et al., "Parameter adaptive control in peripheral milling", *Int. J. Mach. Tools and Manuf.,*, Vol.27, No.3, pp.399-414, 1987.

[192] S.J. Rober et al., "A digital robust controller for cutting force control in the end of milling", *ASME J. Dyn. Syst. Meas., Control*, Vol.119, No.2, pp.146-152, 1997.

[193] W.C. Pitstra et al., "Controller designs for constant cutting force tuning machine control", *ISA Transactions*, Vol.39, pp.191-203, 2000.

[194] S.J. Huang et al., "The application of neural networks in self-tuning constant force control", *Int. J. Mach. Tools and Manuf.,*, Vol.36, No.1, pp.17-31, 1996.

[195] M.K. Kim et al., "Applications of the fuzzy control strategy to adaptive force control of non-minimum phase and milling operations", *Int. J. Mach. Tools and Manuf.,*, Vol.34, No.5, pp.677-696, 1994.

[196] F.C. Chen, and H.K. Khalil, "Adaptive control of a class of nonlinear systems using neural networks", *IEEE Trans. Automat. Contr.*, Vol. 40, pp.791-801, 1995.

[197] F.L. Lewis et al., "Neural net robot controller with guaranteed tracking performance", *IEEE Trans. Neural Networks*, Vol. 6, pp. 703-715, 1995.

[198] A.S. Poznyak et al., "Stability analysis of dynamical systems using neural networks", *Expert Syst. Applicat.*, Vol.14, pp.227-236, 1998.

[199] R.M. Sanner et al., "Gaussian networks for direct adaptive control", *IEEE Trans. Neural Networks*, Vol.3, pp.837-863, 1992.

[200] S.S. Yu et al., "Adaptive control of nonlinear dynamic systems using θ-adaptive neural networks", *Automatica*, Vol.33, No.11, pp.1975-1995, 1997.

[201] M.M. Polycarpou et al., "Neural networks as on-line approximators of nonlinear systems", *Proc. of 31st IEEE Conf. Decision Contr.*, pp.7-12, 1992.

[202] J.T. Spooner and K.M. Passino, "Stable and adaptive control using fuzzy systems and neural networks", *IEEE Trans. on Fuzzy Systems*, Vol.4, pp. 339-359, 1996.

[203] S.S. Ge et al., "Adaptive neural network control of robot manipulators", *London: Word Scientific*, 1998.

[204] T.Zhang, S.S. Ge and C.C. Hang, "Design and performance analysis of a direct adaptive controller for nonlinear systems", *Automatica*, Vol.35, pp. 1809-1817, 1999.

[205] T.Zhang, S.S. Ge and C.C. Hang, "Stable adaptive control for a class of nonlinear systems using a modified Lyapunov function", *IEEE Trans. on Automatic Control*, Vol. 45, pp.129-132, 2000.

[206] M.M. Polycarpou et al., "Stable adaptive neural control scheme for nonlinear systems", *IEEE Trans. on Automatic Control*, Vol.41, pp.447-451, 1996.

[207] C. Kwan et al., "Robust neural network control of rigid-link electrically robots", *IEEE Trans. on Neural Networks*, Vol.9, pp.581-588, 1998.

[208] S.S. Ge, C. Wang, T.H. Lee, "Adaptive backstepping control of a class of chaotic systems", *Internation Journal of Bifurcation and Chaos*, Vol.10, No.5, pp.1149-1156, 2000.

[209] T. Zhang, S.S. Ge, C.C. Hang, "Adaptive neural network control for strict-feedback nonlinear systems using backstepping design", *Automatica*, Vol. 36, pp.1835-1846, 2000.

[210] Y. Zhang et al., "Stable neural controller design for unknown nonlinear systems using backstepping", *IEEE Trans. on Neural Networks*, Vol.11, No.6, pp.1347-1360, 2000.

[211] Y. Fang et al., "Synthesis of the sliding -mode neural network controller for unknown nonlinear discrete-time systems", *Internation Journal of Systems Science*, Vol.31, No.3, pp.401-408, 2000.

[212] M. Ertugrul and O. Kaynak, "Neuro sliding mode control of robotic manipulators", *Mechatronics*, Vol.10, pp.239-263, 2000.

[213] A.I. Bhatti et al., "A nonlinear sliding mode control design approach based on neural network modeling", *International Journal of Robust and Nonlinear control*, Vol.9, pp.397-423, 1999.

[214] Y. Li, K.C. Tan, and M. Gong, "Global structure evolution and local parameter learning for control system model reductions", In D. Dasgupta and Z. Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*, pp.345-360. Springer Verlag, 1997.

[215] Y. Li and A. Haeussler, "Artificial evolution of neural networks and its application to feedback control", *Artificial Intelligence in Engineering*, Vol.10, No.2, pp.143-152, 1996.

[216] M. Chowdhury and Y. Li, "Learning fuzzy control systems directly from the environment", *Int. J. Intelligent Systems*, Vol.13, No.10, October, 1998.

[217] W.K. Lennon and K,M. Passino, "Genetic adaptive identification and control", *Engineering Applications of Artificial Intelligence*, Vol.12, pp.185-200, 1999.

[218] J. R. Gremling and K.M. Passino, "Genetic adaptive state estimation", *Engineering Applications of Artificial Intelligence*, Vol.13, pp.611-623, 2000.

[219] A. H. Watson et al., "System Identification of Fluid Systems using Genetic Programming", *Proc. of the Second Online Workshop on Evolutionary Computation (WEC2)*, 45-48, 1996.

[220] B. Howley, "Genetic Programming of Near minimum Time Spacecraft Attitude Manoeuvres", *Proc. of the 1st Annual Conference on Genetic Programming*, July 28-31, pp. 98-106, 1996.

[221] D.C. Dracopoulos, "Evolutionary Control of a Satellite", *Proc. of the 2st Annual Conference on Genetic Programming*, July 13-16, pp. 77-81, 1997.

[222] S. Dominic, "Chemical Process Controller Design using Genetic Programming", *Proc. of the 3st Annual Conference on Genetic Programming*, July 13-16, pp. 77-81, 1998.

[223] K. Chellapilla, "Automatic generation of nonlinear optimal control laws for broom balancing using evolutionary programming", *Proc. of the IEEE World Congress on Computational Intelligence*, pp.195-200, 1998.

[224] F.Brauer et al., "The qualitative theory of ordinary differencial equations: an introduction", *Benjamin*, 1969.

[225] A.I. Bhatti et al., "A nonlinear sliding mode control design approach based on neural network modeling", *International Journal of Robust and Nonlinear Control*, Vol.9, pp.397-423.

[226] Y. Fang and T.W.S. Chow, "Synthesis of the sliding-mode neural network controller for unknown nonlinear discrete-time systems", *Internation Journal of Systems Science*, Vol.31, No.3, pp.401-408, 2000.

[227] X. Li et al., "Neural network based, discrete adaptive sliding mode control for idle speed regulation in IC engines", *Journal of Dynamic Systems, Measurement and Control*, Vol.122, pp.269-275, 2000.

[228] M. Ertugrul and O. Kaynak, "Neuro sliding mode control of robotic manipulators", *Mechatronics*, Vol.10, pp.239-263, 2000.

[229] S.G. Tzafestas et al., "Design and stability analysis of a new sliding-mode fuzzy logic controller of reduced complexity", *Machine Intelligence and Robotic Control*, Vol.1, No.1, pp.27-41, 1999.

[230] J-P. Su et al., "Adaptive fuzzy sliding mode control with GA-based reaching laws", *Fuzzy Sets and Systems*, Preprint, 2000.

[231] Q.P.Ha et al., "Fuzzy moving sliding mode control with application to robotic manipulators", *Automatica*, Vo.35, pp.607-616, 1999.

[232] H. Lee and M. Tomizuka, "Robust adaptive control using a universal approximator for SISO nonlinear systems", *IEEE Trans. on Fuzzy Systems*, Vol.8, No.1, 2000.

[233] H. Richter and K. Reinschke, "Optimization of local control of chaos by an evolutionary algorithm", *Physica D*, Vol.144, pp.309-334, 2000.

[234] A. Bastian, "Identifing fuzzy models utilizing genetic programming", *Fuzzy Sets and Systems*, Vol.113, pp.333-350, 2000.

[235] C.C. Lee, "Fuzzy logic in control system: fuzzy logic controller (part I and part II)", *IEEE Trans. on Systems, Man and Cybenetics*, Vol.20, pp.408-435, 1990.

[236] J.M. Mendel, "Fuzzy logic systems for engineering : a tutorial", *Proceedings of IEEE*, Vol.83, pp.345-377, 1995.

[237] F. Bouslama et al., "Fuzzy control rules and their natural control laws", *Fuzzy Sets and Systems*, Vol.48, pp.65-86, 1992.

[238] A.E. Hajjaji et al., "Explicit formulas for fuzzy controller", *Fuzzy Sets and Systems*, Vol.62, pp.135-141, 1994.

[239] R. Langari, "A nonlinear formulation of a class of fuzzy linguistic control algorithms", *Proc. of the 1992 American Control Conference*, Chicago, IL, 1992.

[240] F.L. Lewis and K. Liu, "Towards a paradigm for fuzzy logic control", *Automatica*, Vol.32, pp.167-181, 1996.

[241] H.A. Malki et al., "New design and stability analysis of fuzzy PD controllers", *IEEE Trans. on Fuzzy Systems*, Vol.2, pp.245-254, 1994.

[242] F. Matia et al., "Fuzzy controller: lifting the linear-nonlinear frontier", *Fuzzy Sets and Systems*, Vol.52, pp.113-129, 1992.

[243] Y.M. Pok et al., "An analysis of fuzzy control systems using vector space", *Proc. of the Second IEEE Int. Conf. on Fuzzy Systems*, California, pp.363-368, 1993.

[244] K.L. Tang et al., "Comparing fuzzy logic with classical controller designs", *IEEE Trans. on Systems, Man and Cybenetics*, Vol.17, pp.1085-1087, 1987.

[245] H. Ying et al., "Fuzzy control theory: a nonlinear case", *Automatica*, Vol.26, pp.513-520, 1992.

[246] H. Ying, "The simple fuzzy controllers using different inference methods are different nonlinear proportional-integral controllers with variabl gains", *Automatica*, Vol.29, pp.1579-1589, 1993.

[247] K. Tanaka et al., "Robust stabilization of a class of uncertain nonlinear systems via fuzzy control: Quadratic stabilizability H^{∞} control theory and linear matrix inequalities", *IEEE Trans. on Fuzzy Systems*, Vol.4, pp.1-13, 1996.

[248] H.O. Wang et al., "An approach to fuzzy control of nonlinear systems: stability and design issue", *IEEE Trans. on Fuzzy Systems*, Vol.4, pp.14-23, 1996.

[249] R.R. Yager and D.P. Filev, "Essentials of fuzzy modeling and control", *Wiley*, New Yeak, 1994.

[250] L. Reznik et al., "PID plus fuzzy controller structure as a design base for industrial applications", *Engineering Applications of Artificial Intelligence*, Vol.13, pp.419-430, 2000.

[251] Z.W. Woo et al., "A PID type fuzzy controller with self-tuning scaling factors", *Fuzzy Sets and Systems*, Vol.115, pp.321-326, 2000.

[252] J. Carvajal et al., "Fuzzy PID controller: design, performance evaluation, and stability analysis", *Information Science*, Vol.123, pp.249-270, 2000.

[253] T.-H. S. Li et al., "Design of a GA-based fuzzy PID controller for non-minimum phase systems", *Fuzzy Sets and Systems*, Vol.111, pp.183-197.

[254] H. Ying, "Theory and application of a novel fuzzy PID controller using a simplified Takagi-Sugeno rule scheme", *Information Science*, Vol.123, pp.281-293, 2000.

[255] S.S. Farinwata, "An approach to the analysis of robust stability of fuzzy control systems", *Fuzzy Logic*, John Wiley and Sons, Ltd, pp.165-201, 2000.

[256] R. Palm et al., "Improving the global performance of a fuzzy gain-scheduler by supervision", *Engineering Applications of Artificial Intelligence*, Vol.12, pp.297-307, 1999.

[257] D.K. Pirovolou et al., "Output tracking uing fuzzy neural networks", *Fuzzy Logic*, John Wiley and Sons, Ltd, pp.335-348, 2000.

[258] W.K. Lennon and K. Passino, "Intelligent control for brake systems", *IEEE Trans. on Control Systems Technology*, Vol.7, No.2, pp.188-202, 1999.

[259] S.-J. Kang et al., "Evolutionary design of fuzzy rule base for nonlinear system modeling and control", *IEEE Trans. on Fuzzy Systems*, Vol.8, No.1, pp.37-45, 2000.

[260] J.-H. Chou et al., "Application of the Taguchi-genetic method to design an optimal grey-fuzzy controller of a constant turning force system", *Journal of Materials Processing Technology*, Vol.105, pp.333-343, 2000.

[261] M.R. Emami et al., "Fuzzy-logic control of dynamic systems: from modeling to design", *Engineering Applications of Artificial Intelligence*, Vol.13, pp.47-69, 2000.

[262] Q.P. Ha et al., "Fuzzy moving sliding mode control with application to robot manipulators", *Automatica*, Vol.35, pp.607-616, 1999.

[263] G.G. Rigatos et al., "Mobile robot motion control in partial unkonwn environments using a sliding-mode fuzzy-logic controller", *Robotics and Autonomous Systems*, Vol.33, pp.1-11, 2000.

[264] C.M. Kwan and F.L. Lewis, "Robust backstepping control of induction motors using neural networks", *IEEE Trans. on Neural Networks*, Vol.11, No.5, pp.1178-1187, 2000.

[265] F. Gao et al., "A simple nonlinear controller with diagonal recurrent neural network", *Chemical Engineering Science*, Vol.55, pp.1283-1288, 2000.

[266] K. B. Kim et al., "Control of chaotic dynamical systems using radial basis function network approximators", *Information Science*, Vol.130, pp.165-183, 2000.

[267] D. Wang and P. Bao, "Enhancing the estimation of plant Jacobian for adaptive neural inverse control", *Neurocomputing*, Vol. 34, pp.99-115, 2000.

[268] M. Norgaard, O. Ravn, N. K. Poulsen, L. K. Hansen, "Neural Networks for Modeling and Control of Dynamic Systems", Springer-Verlag, London, 2000.

Appendix A

Publication List

- Y. Chen and S. Kawaji, "Evolving Neurofuzzy Systems for System Identification" *Proc. of Int. Symposium on Artificial Life and Robotics(AROB)*, January 15-17, pp.204-207, 2001, Tokyo, Japan.
- Y. Chen and S. Kawaji, "System Identification and Control using Probabilistic Incremental Program Evolution Algorithm", *Journal of Robotics and Machatronics*, Vol.12, No.6, 2000.
- Y. Chen and S. Kawaji, "Thrust Force Estimation in Drilling System by Using Neural Networks", *Proc. of The Society of Instrument and Control Engineers*, Kyushu Branch, pp.199-202, 2000.
- Y. Chen and S. Kawaji, "Evolutionary Design of Hierarchical T-S fuzzy Models by Modified PIPE and EP Algorithms", Submitted to *IEEE Trans. on Evolutionary Computation*, 2000.
- Y. Chen and S. Kawaji, "Evolving Wavelet Neural Networks for System Identification", *Proc. of The International Conference on Electrical Engineering (ICEE'2000)*, pp.279-282, July 24-28, 2000, Kitakyushu, Japan.
- S. Kawaji and Y. Chen, "Soft Computing Approaches to System Identification", *Japan-USA-Vietnam Workshop on Research and Education in Systems, Computation and Control Engineering (RESCCE'2000)*, pp. 170-179, June, 2000.
- S. Kawaji and Y. Chen, "Soft Computing Approach to Nonlinear System Identification", *Proc. of the IEEE Int. Conf. on Industrial Electronics, Control and Instrumentation (IECON-2000)*, October 22-28, pp. 1803-1808, 2000, Nagoya Congress Center, Nagoya, JAPAN.
- Y. Chen and S. Kawaji, "Evolving Artificial Neural Networks by hybrid approaches of PIPE and Random Search Algorithm", *Proc. of The Third Asian Control Conference (ASCC'2000)*, July 4-7, 2000, pp.2206-2211 Shanghai, China.

- Y. Chen and S. Kawaji, "Evolving Wavelet Neural Networks for System Identification", *Proc. of The 18th Annual Conference of the Robotics and Machtronics of Japan*, 1A1-29-039, pp. 1-6, 2000.
- Y. Chen and S. Kawaji, "Identification and Control of Nonlinear System using PIPE Algorithm", *Proc. of Int. Workshop on Soft Computing in Industry'99 (IWSCI'99)*, pp. 60-65, Muroan, Japan.
- Y. Chen and S. Kawaji, "Evolutionary Control of Discrete-Time Nonlinear System using PIPE Algorithm", *Proc. of IEEE Int. Conference on SMC'99*, pp. 1078-1083, Tokyo.
- Y. Chen and S. Kawaji, "A Hybridization of PIPE and Random Search Algorithm for Model Identification and Parameters Estimization", Sent to the *Japanese Journal of Systems, Control and Information*, 1999.
- Y. Chen, J. Dong and K. Shi, "Accelerated evolutionary programming based optimal design of an nonlinear fuzzy control system", *Chinese Journal of Shandong Institute of Building Materials*, No.1, pp .28-30, 1998.
- J. Dong, Y. Chen and K. Shi, "An improved genetic algorithm", *Chinese Journal of Qingdao Institute of Chemical Technology*, No.2, pp. 169-173, 1998.
- K. Shi and Y. Chen, "A method of improving the convergence speed of genetic algorithm", *Chinese Journal of Information and Control*, Vol. 27, No.4, pp.289-293, 1998.
- C. Li and Y. Chen, "An algorithm for blurred image caused by rotatory Motion", *Chinese Journal of Shandong Institute of Building Materials*, No.1, pp. 31-33, 1998.
- Y. Chen, J. Dong and K. Shi, "Approximation of arbitrary nonlinear function using symbolic express of genetic programming", *Chinese Journal of Shandong Institute of Building Materials*, No.2, pp.142-145, 1998.
- X. Cui, Y. Chen, K. Shi and B. Yang, "Application of artificial neural networks-genetic algorithms to prediction of cement strength", *Chinese Journal of Shandong Institute of Building Materials*, No.3, pp.275-277, 1998.
- K. Shi, Y. Chen and J. Dong, "Semi-self-organizing artificial neural networks", *Proc. of Chinese Conference on Intelligent Automation*, pp.194-199, 1998.
- X. Cui, Y. Chen, K. Shi, and B. Yang, "A training method of neural network for prediction and modeling— accelerated genetic algorithm", *Proc. of 4th Chinese Electron Association Society Conference*, pp. 579-582,1998.
- K. Shi, Y. Chen and J. Dong, "Classification of cancer cells using neural network in combination with expert experience", *Chinese Journal of Tsinghua science and technology*, No.4, pp. 853-855, 1997.

- Y. Chen, J. Dong and K. Shi, "A method of using genetic algorithm to extract fuzzy rules directly from numerical input-output data for approaching nonlinear functions", *Chinese Journal of Shandong Institute of Building Materials*, No.4, pp.36-40, 1996.
- R. Shong and Y. Chen, "Reform the teaching method and enhance the teaching quality of higher mathematics", *Chinese Journal of Shandong Polytechnic University*, No.1, pp. 86-88, 1993.
- Y. Chen, "Reinforce the education of the humanities and improve the adaptation of the engineering students", *Chinese Journal of Shandong Polytechnic University*, No. 2, pp.88-92, 1992.
- X. Zhang and Y. Chen, "Try to discuss the morbid state of mind of students and its treating method", *Chinese Journal of Shandong Polytechnic University*, No. 3, pp. 53-56, 1991.

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Professor Shigeyasu Kawaji, Chairman

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Professor Hiroshi Kashiwagi

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Professor Ryozo Nakamura

Approved for the University Committee on Graduate Studies.

Professor Masanao Ebata

This dissertation was submitted to the Graduate School of Science and Technology, and was accepted as partial fulfilment of the requirement for the degree of Doctor of Philosophy.

March 2001

Dean of the Graduate School
Professor Yorinbu Sonoda