Fuzzy Adaptive Recurrent Counterpropagation Neural Networks: A Neural Network Architecture for Qualitative Modeling and Real-Time Simulation of Dynamic Processes

by

YaDung Pan

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY
WITH A MAJOR IN COMPUTER ENGINEERING

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 9 4

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

With the emerging need for ever higher degrees of system autonomy, the design of intelligent control systems has become the most active research area not only within the control community, but also in modeling and simulation.

The goal of intelligent control systems is to address control problems that involve system uncertainty, fault diagnosis, and system reconfiguration, which require the capability of learning modified system behavior and adapting to it. Sometimes, intelligent control systems are also expected to have the capability to respond to commands issued at a high level of abstraction in terms of symbolic or linguistic variables. These goals cannot be achieved easily by using conventional control methods only. Therefore, intelligent control systems are integrating systems that not only contain conventional control for low-level component control, but also use expert systems for knowledge abstraction and fault diagnosis, fuzzy logic for qualitative reasoning and modeling, and artificial neural networks for system learning and adaptation.

In conventional control methodology, the system to be controlled is viewed as a plant consisting of a collection of interacting components. It can be conceptually thought of as a source of data measurable through the system outputs, and offers the capability to influence its behavior by exerting the system through the system inputs. It is therefore easy to distinguish the controller from the system. However in an intelligent control system due to the hierarchies of its functional structure, it is not an easy task to separate the controller from the system to be controlled. The various controllers are part of the overall system, and lower-level controllers may look like part of the plant to the higher-level controllers.

Due to the inherent complexity of intelligent control systems, the control community has not yet even agreed upon a clear definition of what "intelligent control" really means or entails. It is however useful to provide a few definitions of intelligent control as they can be found in the contemporary control literature. [59] defines an intelligent control system as "a control system with ultimate degree of autonomy in terms of self-learning, self-reconfigurability, reasoning, planning and decision making, and the ability to extract the most valuable information from unstructured and noisy data from any dynamically complex system and environment." [1] discusses intelligent control from the point of view of a definition of intelligence per se, and explores dimensions of intelligently acting or behaving systems.

1.1 System Modeling

Since an understanding of the underlying system characteristics is fundamental to all control actions, the ability to mathematically describe, i.e., model, these system characteristics is essential to control in general. Furthermore, a methodology capable of describing basic system characteristics or modes of system behavior in the context of incomplete system knowledge, accounting for potential changes in the very fabric of these system characteristics as time passes, is paramount to the successful design of an intelligent control system. Thus, an intelligent control architecture must contain, as a central part, a modeling engine capable of dealing with uncertainty, and of learning, on the fly, new modes of system behavior.

Due to the inherently hierarchical structure of all intelligent control systems, the plant characteristics cannot be described as a set of operational data alone, it must also include current set point information for the linearization module, scheduling information for the event control module, elements of nominal operational modes and significant variations thereof for the fault diagnosis module, and a symbolic processor and reasoning algorithms for the human interface module.

Based on the most appropriate knowledge representation scheme and the available form of information about the system, the plant characteristics can be described using either a rule-based or a model-based approach, and either quantitative or qualitative variables.

The rule-based approach is commonly used for the fault diagnosis module, since the information needed for fault diagnosis is of a global nature, and since a conclusion can often be reached without resorting to the detailed information at a microscopic level that a model-based approach would provide. Thus, the rule-based approach is for this purpose usually more practical and more economical. By using the "pattern $\Rightarrow$ action" rules generated by the expert employing heuristic problem-solving techniques, the fault diagnosis module can easily be constructed on the basis of available expert knowledge and take action when it is triggered by a discrepancy in the observed operational patterns.

Fuzzy logic control is a technique for implementing control strategies by using a rule-based formulation, but instead of dealing with crisp information as would usually be employed by expert-system controllers or programmable logic controllers, fuzzy logic control is able to use fuzzy, i.e., imprecise, information.

Model-based approaches are used to describe the system's causal and functional, temporal and spatial information, and are commonly applied to constructing the event-control and scheduling module. The distinction between rule-based and model-based approaches is somewhat artificial. Does not any codification of information about a system constitute a "model"? So, why then should a rule base not be called a model as well? Evidently, there is no easy answer to these questions. Traditionally, rule bases are static in nature, i.e., the conditions for firing a rule are instantaneous, and the effects of such firings are equally immediate. Such a "rule base" (a very narrow definition of the term) can evidently not cope with situations where the use of information collected at different points in time is essential. In such a case, a model-based approach is usually preferred, whereby the term "model" simply implies use of more thorough information about the plant to be controlled. Model-based and rule-based modules are often integrated into diagnostic units, whereby the model-based component is used primarily for fault detection and isolation, and the rule-based component is employed predominantly in the processes of fault characterization and repair.

Using quantitative variables and data, a system can be described by sets of mathematical (differential and algebraic) equations. This description has the advantage of being very compact, much more so than any behavioral description. Its accuracy and interpretation capability have made this representation attractive for use by classical control theory. This system representation is mostly used to describe the low-level components of the system, where precise information is required for control.

Inspired by the human capability for reasoning and predicting physical system behavior without the use of structural models, artificial neural networks have been introduced to identify system input/output models through behavioral learning. Neural networks provide an unstructured approach to recognizing system behavior when no or only very little structural information about the system to be modeled is available. Their capability of learning new modes of system behavior on the fly, thereby adapting themselves easily to an environment that changes with time, have secured them a prominent role in intelligent control architectures.

Qualitative modeling approaches have been inspired by the ways how humans reason about system behavior. Qualitative approaches are forgiving when dealing with knowledge about a system that is only available qualitatively, when confronted with unmodeled plant dynamics, or when exposed to unknown disturbances, i.e., whenever they have to cope with data uncertainty of one sort or other. Several qualitative modeling approaches have been proposed including naïve physics, common sense theory, qualitative reasoning, and qualitative simulation. Most of these approaches however have difficulties accommodating pieces of precise knowledge, and are therefore overgeneralizing, with the negative effect that their predictions are unnecessarily ambiguous.

Another emerging approach using fuzzy logic to describe system characteristics through qualitative information is based on human thinking and natural language processing. It provides a means to describe both the system behavior and control strategy with varying degrees of detail. It can accommodate exactly as much quantitative information as is available and/or as is deemed suitable for the task at hand. This approach therefore provides an ingenious compromise between the quantitative and qualitative approaches to modeling. It is this approach to modeling that will be studied and discussed in great detail in this dissertation.

1.2 Research Motivation and Organization

Expert system technology has been the most successful methodology in all of artificial intelligence. The use of expert systems is widespread across all branches of science and engineering, and contrary to many other artificial intelligence methods that work well for small textbook example but don't scale up to industry-size applications, expert systems have been successfully employed in the solution of very large and arbitrarily complex industrial applications.

Currently, expert systems use a rule-based formulation to represent knowledge. They have the advantage of an easy interpretation and implementation. The bottleneck in today's expert system technology is the process of knowledge acquisition. Rules are usually generated from human experts' experience, and are not being generated automatically from observations. Experts who are able to formulate their implicit knowledge in the form of explicit rules are hard to come by, and make the on-line design of expert systems or at least their on-line adaptation (incorporation of new pieces of knowledge) not easily feasible. To this end, it would be very useful to have a technology available that can synthesize expert systems directly from observations. Another shortcoming of contemporary expert system technology is their computational dependence on the size of the rule base. With an increasing volume of available knowledge, today's expert systems have a tendency to soon become sluggish. These two shortcomings are to be addressed in this dissertation. A methodology will be shown that allows to design expert systems from observations alone, and a parallel implementation scheme will be developed whose execution speed is independent of the size of the rule base.

The objective of our research is to develop an efficient autonomous system modeling methodology combining the advantages of the quantitative approach (neural networks) with those of the qualitative approach (fuzzy logic). A novel neural network architecture will be demonstrated that avoids the most prevalent pitfall of previous neural network architectures, namely their extremely slow behavior learning capabilities, as documented e.g. by the popular feedforward neural networks trained through backpropagation. The new architecture has been coined Fuzzy Adaptive Recurrent Counterpropagation Neural Network (FARCNN). FARCNNs can be directly synthesized from a set of training data, making behavioral learning extremely fast. It takes a single pass through the available training data set to learn every piece of information that is contained in the data. FARCNNs can be applied directly and effectively to model dynamic and static system behavior based on the observed input/output data without need to know anything about the internal system structure. Due to their capability of automatic model construction and information extraction, the FARCNN architecture also provides a new approach to constructing expert systems for real time applications.

The FARCNN architecture adopts the technique of Fuzzy Inductive Reasoning (FIR), an advanced behavioral modeling methodology based on the human ability to find analogies among similar processes, to predict coarse system dynamic behavior. A neural network architecture is presented that allows to rapidly implement a large variety of different static and dynamic functional relationships with minimal need of learning. The basic counterpropagation neural network (CNN) architecture provides a parallel implementation mechanism for binary truth-table lookup. As a typical application of the technology, an arbitrary combinatorial digital circuit can be realized quickly and efficiently. A single pass through the Kohonen and Grossberg layers suffices to produce the desired outputs to an arbitrary set of inputs. The gate delays will thereby be minimized.

A generalized counterpropagation neural network (GCNN) generalizes the CNN architecture to deal with multi-valued logic. This enhanced architecture can for example be used for efficient implementation of finite state machines (FSMs). Each pass through the network corresponds to one clock of the FSM. This allows for real-time implementation of Petri net simulators.

Another application of this technology is the real-time implementation of forward chaining expert systems. The Kohonen layer can implement the conditions for rules to be fired, while the Grossberg layer implements the consequences of such firings.

A recurrent counterpropagation neural network (RCNN) is also presented. Its enhancement allows to incorporate memory into the network. It enables the transition from dealing with purely static relationships only to being able to handle fully dynamic functional relationships. It enables us to efficiently implement electronic circuits containing flip flops, it allows us to realize second generation time-dependent expert systems for real-time applications, and it provides us with means to implement crisp inductive reasoning models.

The processing of continuously changing information causes considerable difficulties to the inherently digital network architecture. However, a solution to this dilemma was proposed involving newly designed fuzzy A/D and fuzzy D/A converters and a standardized backpropagation neural network (BNN). The problem of propagating fuzzy membership values across the network is addressed, and a network architecture for accomplishing the fuzzy signal propagation is presented.

In Chapter 2 of this dissertation, the basic concepts behind neural network and fuzzy system technologies are reviewed, and current state-of-the-art implementations of both methodologies are discussed together with their most successful applications.

In Chapter 3, the methodology of fuzzy inductive reasoning is presented.

In Chapter 4, it will be shown how fuzzy inductive reasoners can be implemented using a basic counterpropagation neural network, and the design strategy of the FARCNN architecture is introduced.

In Chapter 5, several examples are shown that illustrate the application of the FARCNN architecture to practical problems, and in some cases compare the results obtained using the FARCNN technology to previously published inductive models for the same applications using different inductive modeling approaches. The applications discussed in Chapter 5 include a static continuous nonlinear equation, a discrete sequential equation, and a dynamic continuous nonlinear hydraulic motor system.

In Chapter 6, conclusions are presented, and open research questions are being discussed.

CHAPTER 2

Review of Neural Networks and Fuzzy Logic

Both the technologies of artificial neural networks and fuzzy logic are inspired by the human ability to learn by observation and infer knowledge about unknown systems. Extensive research and application results have been published demonstrating the potential promises of these two technologies for a variety of areas. These are also the two most important technologies making intelligent control systems feasible. Some references even equate the term "intelligent control" with making use of neural networks and/or fuzzy logic within the overall control architecture. With the recent appearance and commercialization of neural chips and fuzzy chips, it has now also become possible to use neural control and fuzzy control in real-time applications. As these two emerging technologies have attracted a lot of attention by many researchers leading to hundreds if not thousands of new articles in journals and conference proceedings every year, an exhaustive survey of either technology has become quite impractical. It will not be attempted.

In this chapter, a brief overview of the basic concepts pertaining to both technologies is given, some relevant state-of-the-art research results are being examined, yet only those subjects with direct relevance to our own research efforts are presented in more detail.

2.1 Neural Networks

Since artificial neural networks (perceptrons) were first mentioned in the literature by McCulloch and Pitts [43], the theory of neural networks has been advanced a lot. Especially during the most recent decade, neural networks have been among the most active research areas worldwide, leading to hundreds if not thousands of publications in every year. Impressive results have also been reported relating to the applicability of neural networks for solving complex problems of industry-size proportions. In our view, the most promising results are those that show the universal applicability of neural network technology. Stinchcombe and White have shown that a two-layer neural network with sufficiently many neurons in its hidden layer can approximate *any* continuous function [61]. Similar results were reported in the same year by Funahashi [16].

As neural networks have the ability to learn and map the behavior of complex systems without being provided with any knowledge about their internal structure, neural networks have predominantly been applied to both static and dynamic pattern recognition problems in signal processing, image processing, system identification, and control.

Also due to the inherent parallelism in neural networks, they have the potential to process information in very efficient ways when appropriate hardware is available. One example of exploiting the parallelism of neural networks for engineering problems is the application of parallel sorting. The computational complexity of straightforward sequential sorting algorithms grows quadratically with the number of elements to be sorted. The theoretical lower limit of the computational complexity of any sequential algorithm for sorting n elements is $n \cdot \log n$. With the advent of neural network implementations of sorting algorithms, it has now become feasible to make sorting (an important component of many engineering problems) even faster, and implement the sorting function in hardware as a neural chip.

In the following sections, we are going to look at neural networks in more detail from these two points of view: pattern recognition and parallelization. Finally, a much more detailed review of one particular neural network architecture, the counterpropagation network, will be presented. The reason for this selection is simple: all these components are geared toward use in our own neural network architecture, the Fuzzy Adaptive Recurrent Counterpropagation Neural Network (FARCNN), that will be discussed in detail in the subsequent chapters of this dissertation.

2.1.1 Neural Networks in System Control and Identification

The capability of neural networks to identify unknown systems from their input/output behavior alone has secured them a prominent role in the design of intelligent control architectures.

Most of mathematical system control theory is based on complete knowledge about the plant to be controlled. A model of the plant is described as a set of linear or nonlinear differential equations, and the controller or control loop design is based on this model. However, realistically complex problems always contain time-variant, unknown parameters, or even unknown nonlinear terms associated with the system, the so-called unmodeled dynamics of the system. Neural networks, with their inherent ability to identify system behavior on the fly and to adapt themselves easily to changing situations, are earmarked for use in control applications where the influence of unmodeled dynamics is significant. The more nonlinear a plant and the less is known about the plant to be controlled, the better will neural network controllers perform in comparison with the more traditional control architectures. In neural control, neural networks are either used to identify the plant to be controlled, or to identify the controller itself. Often neural controllers come in pairs of neural networks, one identifying the plant, and the other identifying the controller.

Two different classes of neural networks have traditionally been employed in system identification and controller design: multilayer feedforward neural networks [74] and recurrent networks [25]. From a system dynamics perspective, the multilayer feedforward network is a static nonlinear input-output mapping system, i.e., a nonlinear multivariable function generator, whereas the recurrent network is a dynamic nonlinear feedback system [49]. Each of these two classes has proven its utility by means of many realistically-sized applications, and extensions to the basic architectures have been reported that improve their efficiency. Feedforward networks are advantageous because of their simpler structure, and because they are somewhat easier to use. Recurrent networks have the advantage of being able to learn more complex behavior with a smaller number of neurons; they can, due to their inherently dynamic nature, mimic the behavior of dynamic systems more easily; and they can learn new behavioral patterns much faster than their static (feedforward) cousins.

Subsequently, we shall focus our attention on the multilayer feedforward networks, since they are at the heart of our own FARCNN architecture.

Multilayer networks were first introduced by Werbos [74]. They make use of the generalized delta rule with back-propagation for training the network by changing its connection weights, biases, and the thresholds of the output activation functions. Werbos' original work went practically unnoticed for twelve years, after which this network architecture was finally popularized in a frequently cited publication by Rumelhart, Hinton, and Williams [55]. Since then, there have been written many papers that either make use or extend the basic architecture of back-propagation trained multilayer feedforward networks.

Multilayer networks exhibit some problematic shortcomings that need to be addressed: the slow convergence (learning) rate, the local minima problem, and the network configuration problem. A number of publications have dealt with each of these three problems: [68], [54], and [33] suggest variations of the standard back-propagation training algorithm for improved learning speed, [24], and [42] propose strategies for optimally choosing initial weights. [44] and [23] use genetic algorithms and simulated annealing to avoid getting stuck in local minima. [48] presents a scheme to iteratively increase and decrease the number of neurons during the learning process. [5] uses Voronoi diagrams to determine the number of layers, the number of neurons in each layer, and the connection weights needed to classify patterns in a multidimensional feature space. [4] discusses lower and upper bounds on the required size of the training data set for training a feedforward neural network given the desired approximation accuracy that the network is supposed to achieve.

Further, [50] proposes the "functional link network," an architecture that uses an enhanced input space to create complex nonlinear mappings from the input to the output layer. With these high-order terms in the extended input layer, the hidden layers can be eliminated, and thereby, the learning process is considerably simplified.

A number of interesting publications show the applicability of neural networks to control. [49] demonstrates how the multilayer and recurrent neural networks can be used for system identification as well as controller design in the adaptive control of unknown nonlinear dynamic systems. [34] compares the neural network approach to two more traditional adaptive controller design approaches: the self-tuning regulator and the Lyapunov-based model reference adaptive controller. [52] uses the functional link network to identify the system behavior, then uses the required control objectives and the emulated system to train a controller that is emulated by another neural networks.

2.1.2 Using Neural Networks for Parallel Sorting

Sorting plays an important role in many engineering processes. Conventional sorting algorithms are based on the sequential comparison of elements [3]. Due to the inherent parallelism of neural networks, several authors have studied the possibility of implementing sorting algorithms in a neural network for real-time applications. Two different approaches have been reported.

The more direct approach uses a general-purpose neural network trained for this application using specially chosen initial conditions. [22] uses a Hopfield network for this purpose, whereas [70] makes use of a probabilistic neural network respectively.

The other approach uses a dedicated network. [57] and [78] employ custom-designed neural networks especially constructed to solve the sorting problem. This approach has the advantage that it makes better use of the massive parallelism realizable in neural networks. [57] uses three different types of neurons: linear neurons, quasi-linear neurons, and threshold-logic neurons distributed over four layers of a feedforward network structure. Its overall processing time is that of four neurons placed in series, independent of the number of elements to be sorted. Only the number of neurons that are needed in each layer grows with the number of elements to be sorted, not the processing speed. [78] uses a simplified parallel sorting algorithm.

It is implemented by a network employing binary neurons with AND/OR synaptic connections. The overall processing time is two clock cycles. The structure has been implemented for sorting positive integers only, but the approach can be extended to sorting real-valued numbers by making use of real-number comparators.

Both approaches are quite general. They lend themselves to use in industry-size applications. The dedicated approach employing a specialized network has the advantage of a constant processing speed regardless of the number of data points to be sorted, whereas the direct approach has the advantage of being easily implementable in a general-purpose neural chip. Either approach will work fine when used in the context of the FARCNN neural architecture.

2.1.3 Counterpropagation Networks

By combining a competitive network with Grossberg's outstar structure, [19] introduces a two-layer network called counterpropagation network, which can be trained quite rapidly in comparison with other networks. This network can be used to learn any continuous function that maps the input u into the output z, $z = \Phi(u)$, or, if the inverse of Φ exists, then this network can also learn the inverse mapping, $u = \Phi^{-1}(z)$.

In the following chapter, we are going to show how a much simplified binary counterpropagation neural network can be constructed instantaneously without any need for training. This network can be applied as an efficient approach to implementing a finite state machine. However, for now, let us review the counterpropagation architecture as proposed in [19].

The competitive network consists of a layer of instar processing elements as shown in figure 2.1. Each instar element is governed by the equation:

$$\dot{x} = -a \cdot x + b \cdot net \tag{2.1}$$

where x is the output, net is the dot product of the input vector $\mathbf{U}$ and the weight vector $\mathbf{W}$, i.e, $net = \mathbf{U} \cdot \mathbf{W}$, and a, b are positive constants. The weight learning rule is as follows:

$$\dot{\mathbf{W}} = (-c\mathbf{W} + cU) \cdot \mathrm{sgn}(net) \tag{2.2}$$

where sgn is the sign function, and c is a positive constant.

Based on this learning rule, the weight vector will align with the input vector, and a single instar element can reach the largest output after the alignment has taken place. The interconnection among the instars in the competition layer is an on-center off-surround connection as shown in figure 2.2. The i^{th} output is determined by the following equation [15]:

$$\dot{y}_i = a\ y_i + (b - y_i)[f(y_i) + net_i] - y_i \left(\sum_{k \neq i} f(y_k) + \sum_{k \neq i} net_k \right) \tag{2.3}$$

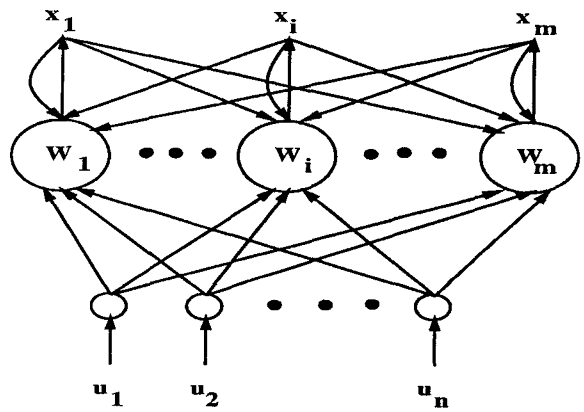

Figure 2.1: Competitive network

where $f(y)$ is a quadratic function, $f(y) = y^2$. This equation emulates a winner-takes-it-all function, i.e., it enhances the largest input and suppresses all the others. The overall function of the competitive network is to recognize an input pattern through a winner-takes-it-all competition, thus, the competitive network is acting like a select function. Consequently, counterpropagation networks can be (and have been) successfully applied to pattern classification problems.

Usually, we choose as many hidden units as there are classes to be identified. Through the training process, the weights of the competitive layer settle near the centroid of each cluster. Each class is represented by a cluster of training data. The winner-takes-it-all competition selects the closest class for new testing data.

The outstar layer consists of a similar layer of instar processing elements that are connected to the outputs of the competition network. As the output value from the competition network is 1 for the recognized input pattern, and 0 for all other

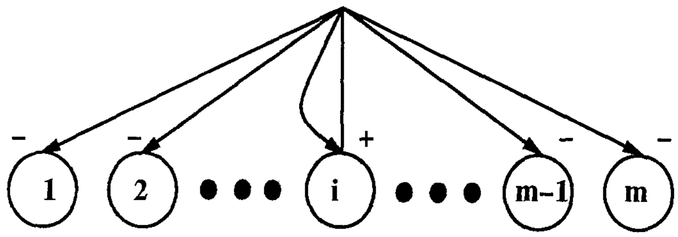

Figure 2.2: An on-center off-surround interconnection

patterns, the weight learning rule is only applied to the corresponding output pattern z. There is no competition function in the outstar layer. The overall function of the outstar layer is to identify the corresponding output for the selected class from the competitive layer.

Interestingly enough, the counterpropagation network operates like an implementation of the nearest-neighbor algorithm [11]. The correct identification of the nearest neighbor carries an error probability bounded by twice the Bayes probability.

If interpolation between neighboring classes is desired, more than one unit of the competitive layer can be used to share the winning class. In this way, an average score for the best matching can be applied instead of the single closest class. This represents an implementation of the k-Nearest-Neighbors algorithm [13] using counterpropagation networks.

One of the drawbacks of working with nearest neighbors is that finding them is computationally expensive both in terms of memory allocation and computation time needed for comparing the new element with every one of the training samples. In the following chapters, we are going to show how the fuzzy reasoning technique can be applied to reduce this burden to some degree.

Several other research efforts are related to the task at hand. [80] shows how the performance of the nearest neighbor algorithm can be improved by using a two-layer perceptron network. [60] demonstrates how appropriate initial connection weights can be produced using the nearest neighbor concept.

2.2 Fuzzy Logic

Since the introduction of fuzzy set theory by Zadeh [81], this theory has been used in a variety of applications. The most fruitful application areas of this theory are fuzzy logic control and fuzzy system estimation [51]. Fuzzy set theory represents an extension to ordinary (crisp) set theory. As in crisp set theory, each element in the set belongs to a class. All classes together form the referential set or the universe of discourse. However, fuzzy set theory adds to each element x in the set E an additional piece of information, the so-called membership value of x, which denotes the degree of confidence in deciding that the element belongs to the class (or subset) A. The fuzzy membership value is a real-valued number in the range 0.0 to 1.0.

$$\forall x \in E : \mu_A(x) \in [0, 1] \tag{2.4}$$

E is the universe of discourse. The value x of E belongs to A with a level of confidence $\mu_A(x)$. All the ordinary set-theoretic operators can be extended to fuzzy sets by adding rules related to the propagation of membership values [28], [17].

Engineering applications deal predominantly with real-valued phenomena. Applying crisp set theory to real-valued data inevitably means to throw away a lot of valuable information. The use of fuzzy set theory overcomes this problem. Real-valued data are converted to fuzzy data using a fuzzification stage. The fuzzifier maps a real-valued number into one or several fuzzy numbers consisting of a class value and a membership value. The fuzzy membership functions describe the process of mapping quantitative (real-valued) data points into fuzzy data points. One fuzzy membership function is associated with each class value.

In most applications, the shape of the membership functions is convex and normal. Typically, membership functions are either triangular, or Gaussian (bell-shaped), or trapezoidal. Figure 2.3 shows a few typical examples of membership functions.

Where the traingular membership function can be characterized as: $\mu_{A_i}(x) = 1 - \frac{|x-c|}{a}$, and the Gaussian membership function can be formulated as: $\mu_{A_i}(x) = 1/(1 + [(x - c/a)^2]^b)$, the parameter b adjusts the slope at membership value of 0.5.

It is possible that a single quantitative data point can be mapped into multiple fuzzy numbers. For example, an outside temperature of 87° F maybe be mapped into the fuzzy numbers $< hot, 0.75 >$ and $< moderate, 0.25 >$ meaning that 75% of

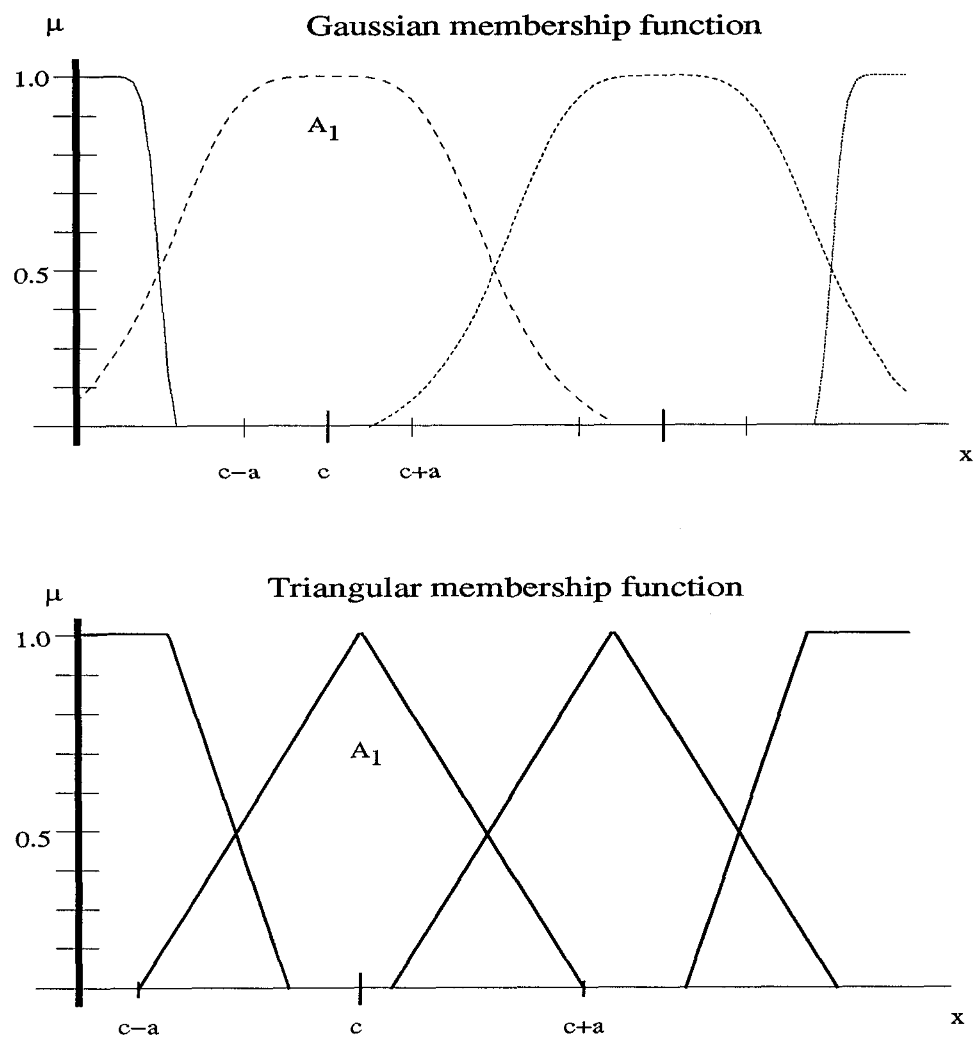

Figure 2.3: Fuzzy membership functions

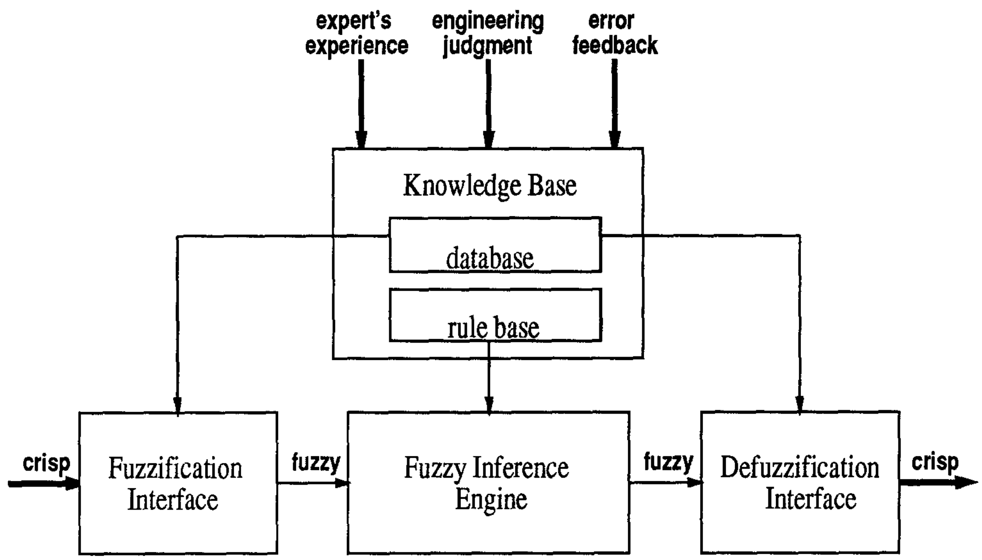

Figure 2.4: Fuzzy reasoning system

the experts on temperature values would argue that 87° F is *hot*, whereas 25% would consider this to be a *moderate* temperature.

The basic configuration of fuzzy reasoning systems for engineering applications contains four components and is shown in Figure 2.4.

The fuzzification interface converts a crisp (quantitative) number into one or several fuzzy numbers consisting of a (usually linguistic) class value and a fuzzy membership value. The knowledge base contains a set of fuzzy rules, usually expressed in linguistic form, and a database that defines the linguistic values. The inference engine is similar to the inference engine used in rule-based expert system. It performs the inferencing operations on the rules. The only dissimilarity is that fuzzy inference engines can fire more than one rule at a time, and they use either the product or the minimum of the individual membership values on the premise part as the firing strength for each rule. The defuzzification interface transforms the result of fuzzy inferencing back into a crisp (quantitative) value. The most commonly used strategies for defuzzification are the centroid of area (COA) rule, and the mean of maxima (MOM) rule. [38], [39], and [76] provide extensive surveys of the derivation of these four fuzzy operation components, the different design strategies, and the different defuzzification methods.

The rule base consists of n fuzzy rules written in the following form:

$$R_k: \quad \text{IF } x_1 \text{ is } A_1^k \text{ and } x_2 \text{ is } A_2^k \text{ and } \cdots$$

$$\text{THEN } z \text{ is } B^k$$

where $k = 1, 2, \cdots n$, is the rule number, x_i denotes the i^{th} input variable to the fuzzy system, z is the output variable, and A_i^k and B^k are the linguistic values of input x_i and output z, respectively. The linguistic values of A_i^k and B^k are further characterized by the fuzzy membership values $\mu_{A_i^k}(x_i)$ and $\mu_{B^k}(z)$.

It is interesting to notice that fuzzy logic can be view as an alternative realization approach to qualitative modeling, since both deal with information in qualitative terms and infer qualitative information. However, fuzzy logic uses the membership values to preserve some of the quantitative information, whereas qualitative modeling approaches usually give this information away.

In our FARCNN architecture, we will use a fuzzy A/D converter as the fuzzification interface, a fuzzy D/A converter as the defuzzification interface, and some sort of counterpropagation network instead of a rule base as the fuzzy inferencing engine.

2.2.1 Fuzzy Logic Control

A significant number of successful applications of fuzzy logic for control system design have been reported in the literature. Those applications include: various control tasks in robotics [67], vehicle speed, tracking, and lateral control [26], [21], automobile transmission control, and motor control [56], to mention just a few.

With the availability af a fuzzy chip [79], fuzzy control has become an attractive low-cost alternative to classical control. Especially in Japan, fuzzy controllers are being integrated into many household appliances, such as cameras (automatic focus control), washing machines, air conditioners, and refrigerators (temperature control).

Fuzzy logic controllers use fuzzy rules that are initially derived from expert experience or engineering judgment, then manually fine tune these fuzzy rules. The design and tuning of fuzzy controllers can be accomplished without any knowledge of the dynamics of the system under control. Since the control logic is specified in the form of coarse rules, it can be modified easily by adding more rules to the knowledge base, or by changing the fuzzy membership values associated with any of the assertions. Results have shown that fuzzy logic controllers can provide a smooth and highly effective control response. In some cases, fuzzy controllers have been shown to be even more accurate than optimally tuned time-invariant PID controllers [12].

However, fuzzy logic controllers suffer from three major shortcomings. First, although the fuzzy controller provides effective and conservative control, the control solution may be far from the optimal control solution. Some authors have addressed this problem. [58] describes the design of an expert fuzzy optimal control system, whereas [20] derives the fuzzy rules from mathematical knowledge of the time-optimal control law with some heuristics.

Second, no analytical approach for the derivation of fuzzy rules is currently known. Thus, the design of a fuzzy controller is usually relying on heuristic knowledge, and it may take a lot of experimentation before the controller is properly tuned. Also, multivariable fuzzy controllers are so difficult to derive that, to this date, the fuzzy control literature has mostly restricted its attention to SISO control problems. Several approaches have been proposed integrating fuzzy learning mechanisms into the fuzzy controller design. Some are based on the feedback system performance, such as [66], [62], and [53]. Others are based on comparing the feedback performance with a fuzzy reference model [37]. [71] and [47] present automatic approaches to generate fuzzy rules from learning.

Third, no analytical analysis technique is currently known that would allow to analyze the stability properties of a fuzzy controller. Consequently, although fuzzy controllers have been demonstrated to provide satisfactory performance, no analytical proof can be given that the fuzzy control system will remain stable under all excitations. This has led to a reluctance of accepting this new technology for critical applications, where a failure of the control system to remain stable might lead to a major catastrophe. No remedy for this shortcoming is currently in sight yet.

2.2.2 Fuzzy System Estimation

Due to the successful application of fuzzy logic to the design of control strategies for many real-world applications and the requirement of a fuzzy reference model for fuzzy model-reference adaptive controllers, a number of researchers have focused their attention on the problem of employing fuzzy logic for identifying the structure of a plant or estimating its parameters. Two different approaches have been proposed for formulating fuzzy rules. One approach uses fuzzy sets in the consequence [69], whereas the other uses parameterized variables in the consequence [64]. In the latter approach, the output of the consequence is a crisp instead of a fuzzy number. More recently, [76] proposed a generalized structure approach called "flexible structure fuzzy logic controller" parameterizing the basic operations used in fuzzy rule processing. It has been shown that the two previously mentioned approaches can be interpreted as special cases of this more general class of models.

Each of these approaches uses a set of measured data as training data, and an adaptive learning algorithm for adjusting the rule base, either by modifying the structure of the rules or by varying the shape of the membership functions. Since neural networks have shown their capability to learn behavior from observations, some references employ neural networks for training fuzzy systems. [27] and [65] use feedforward multilayer networks to update the shape of the membership functions in the premise part, and the parameters in the consequence part. Other references employ fuzzy self-learning algorithms. [18] and [40] use fuzzy learning algorithm to modify the rule base.

Evidently, any general fuzzy learning algorithm can just as easily be applied to learning system behavior as control performance. Thus the approaches advocated in [71] and [47] will also work for this modified task.

All these approaches show the feasibility of using fuzzy rules to identify a plant without extensive knowledge of the system dynamics. They offer a promising alternative to the more traditional methods of system identification.

CHAPTER 3

Fuzzy Inductive Reasoning

In system identification, the process of constructing an input-output model can be decoupled into two separate steps. In the first step, the structure is identified that best characterizes the observed input-output patterns. In the second step, the parameters associated with the selected structure are identified, such that the observed input-output patterns are optimally reproduced by the synthesized system.

Some algorithms don't deal with the problem of structure identification at all, but leave it up to the user to provide the structure. Their task is limited to finding the most appropriate set of parameters for any selected structure. Other algorithms lump the two stages into one. Which algorithm works best in any given situation depends on the problem, i.e., the observed input-output patterns. No single system identification algorithm has been found yet that works uniformly well for all problems.

In this chapter, the fuzzy inductive reasoning methodology is presented as a tool for structure identification. A so-called "optimal mask" is introduced to describe the optimal structure of the system to be identified. However, as the fuzzy inductive reasoning methodology is a qualitative modeling approach, the optimal mask does not describe the complete structure. It only selects a subset of variables from all observed potential input variables that are best suited to be used in reproducing the observed input-output patterns in as unambiguous a fashion as possible. In the next chapter, a methodology based on fuzzy neural networks will be presented for parameter identification.

The inductive reasoning methodology had originally been developed by Klir [29]. It was used in his *General System Problem Solving* (GSPS) methodology as a tool to describe conceptual modes of system behavior. SAPS-II is an implementation of a large subset of the GSPS methodology developed at the University of Arizona [6]. Fuzzy measures were introduced into the GSPS methodology in 1989 [30], and the kernel functions of fuzzy inductive reasoning were incorporated in SAPS-II one year later [41]. SAPS-II is currently available as either a CTRL-C library or a Matlab toolbox.

3.1 The Representation of System Dynamics as a Finite State Machine

In order to describe the dynamics of a system without using any mathematical equations, a time history matrix can be generated by measuring and recording its inputs and outputs at a given set of time points, usually at equidistantly spaced sampling points. The time history is called the *system behavior trajectory* matrix. Each column of this matrix represents one of the recorded variables, and usually, the output variables are concatenated from the right to the input variables. Each row represents a set of measurements of all recorded variables at a given point in time.

The system behavior trajectory matrix represents the characteristics of the system, irrespective of whether it is a static or a dynamic system. However, in the case of characterizing dynamic systems, the sampling interval must be chosen sufficiently small to avoid aliasing effects.

As explained in [73], dynamic systems can always be approximated by difference equations of the form :

$$\mathbf{y}(t) = \mathrm{f}(\mathbf{u}(t), \mathbf{u}(t-\Delta t), \mathbf{y}(t-\Delta t), \mathbf{u}(t-2\Delta t), \ldots) \tag{3.1}$$

That is, the output vector, $\mathbf{y}$, at time t, is a function of the input vector, $\mathbf{u}$, at the same time, the input and output vectors at one sampling interval back, etc.

Now that the system dynamics of function $\mathbf{f}$ are represented in a tabular rather than an analytic form, the structure identification problem is transformed to one of selecting a set of candidate input variables from all the available variables in equation (3.1) that best characterize each of the output variables $y_i(t)$. A separate structure will be identified for each of the output variables, i.e., the original multi-input/multi-output (MIMO) structure identification problem for a p output system is decomposed into p separate structure identification problems for p different multi-input/single-output (MISO) systems. The goal is to find minimal sets of variables that characterize the observed input-output patterns, represented through the system behavior trajectory matrix, in an optimal fashion. A performance index must be defined to evaluate the aptitude of a given set of variables to capture the observed input-output patterns, and a search strategy must be defined for selecting the candidate sets of variables to be tested.

Each entry is stored in the behavior trajectory matrix as a quantitative, i.e., real-valued, number. The direct use of this information invariably leads to an infinite number of combinations of input-output patterns. In order to reduce the variability between input-output patterns, a qualitative form of the behavior trajectory matrix can be generated. In the inductive reasoning approach, so-called "level" or "class" values are being used to define ranges for both input and output variables. Each level is identified by a positive integer. Different variables may use the same class value to denote different ranges. This means that the landmarks, i.e., the borders between neighboring ranges, are determined separately for each variable. An example of discretizing two variables, u_1 and u_2, into three levels each with separate ranges (landmarks) for the two variables is given below:

$$
\begin{array}{cccc}
\textit{level} & \longleftrightarrow & u_1 & u_2 \\
1 & \longleftrightarrow & [0, 0.3] & [0, 0.5] \\
2 & \longleftrightarrow & (0.3, 0.9] & (0.5, 1.0] \\
3 & \longleftrightarrow & (0.9, 1.0] & (1.0, 1.5]
\end{array}
$$

By mapping the continuous trajectories separately into discrete so-called "episodes," the continuous behavior trajectory matrix can be recoded into a qualitative form, called the *discrete episodical behavior* matrix. The episodical behavior of a system is the qualitative counterpart of the quantitative trajectory behavior.

By defining a combination of legal levels of all variables in the set as a "state" of the system, all possible states can now be enumerated, and the system description is thus transformed into a Finite State Machine. The discrete episodical behavior matrix is functioning as a transition table from the input state to the output state. For a static system, the input state consists of one row only, but for a dynamic system, the input state usually involves several rows in accordance with equation (3.1).

Qualitative modeling and simulation derived from the episodical behavior of a system can be interpreted as a generalization of the naïve physics approach [35] and [45]. Traditionally, naïve physics uses only three qualitative values $\{-, 0, +\}$ to represent the two legal levels *negative* and *positive*, and the landmark *zero* in between these two levels. Inductive reasoning, on the other hand, always works with intervals that are appropriately chosen for the modeling task. Naïve physics usually employs first and higher-order qualitative derivatives from measured variables to constrain the feasible states, whereas inductive reasoning uses only the observed variables themselves. Finally, naïve physics commonly induces all episodical behaviors that are compatible with the imposed constraints, whereas inductive reasoning usually only reports the most likely next behavior based on the episodical behavior matrix.

The choice of legal levels influences strongly the representation accuracy of system dynamics in a qualitative form. It is similar to the choice of landmarks in some naïve physics dialects, such as QSIM, and to tuning the premise conditions in rule-based expert systems. Usually, this information is extracted from expert knowledge, and is represented in the form of a crisp set, as in the example shown in this section. In the next section, we are going to introduce the concept of fuzzy recoding and shall show how the fuzzy logic approach can be applied to smooth out the abrupt transitions from one level to the next, thereby leading to an enhanced representation accuracy.

3.2 Finding the Optimal Masks

In inductive reasoning, a mask is a matrix that denotes the dynamic relationship between variables. Once the episodical behavior matrix has been constructed as described in the last section, the purpose of qualitative modeling is to discover the most appropriate finite automata relations among the recoded variables in the episodical behavior matrix. Once such a relationship is found for each output variable, the behavior of the system can be forecast by iterating through the episodical behavior matrix. This corresponds to finding the next state in a finite state machine from the state transition matrix. The more deterministic the state transition matrix, the better the certainty that the future behavior will be predicted correctly.

Assume an episodical behavior matrix for a two input, three output system to be specified as follows:

$$
\begin{array}{l}
time \\ 0.0 \\ \delta t \\ 2 \cdot \delta t \\ 3 \cdot \delta t \\ \vdots \\ (n_{rec} - 1) \cdot \delta t
\end{array}
\begin{array}{c}
\begin{array}{ccccc} u_1 & u_2 & y_1 & y_2 & y_3 \end{array} \\
\begin{pmatrix}
\dots & \dots & \dots & \dots & \dots \\
\dots & \dots & \dots & \dots & \dots \\
\dots & \dots & \dots & \dots & \dots \\
\dots & \dots & \dots & \dots & \dots \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\dots & \dots & \dots & \dots & \dots
\end{pmatrix}
\end{array}
\tag{3.2}
$$

A possible qualitative input-output relation function for output y_1 may present itself in the form:

$$y_1(t) = \tilde{\mathrm{f}}(y_3(t - 2\delta t), u_2(t - \delta t), y_1(t - \delta t), u_1(t)) \tag{3.3}$$

where $\tilde{\mathrm{f}}$ is the qualitative counterpart of function f in equation (3.1), hopefully representing a fairly deterministic relationship. Equation (3.3) can be rewritten in the form of the following "mask":

$$
\begin{array}{l}
{}_t\backslash^x \\ t - 2\delta t \\ t - \delta t \\ t
\end{array}
\begin{array}{c}
\begin{array}{ccccc} u_1 & u_2 & y_1 & y_2 & y_3 \end{array} \\
\begin{pmatrix}
0 & 0 & 0 & 0 & -1 \\
0 & -2 & -3 & 0 & 0 \\
-4 & 0 & +1 & 0 & 0
\end{pmatrix}
\end{array}
\tag{3.4}
$$

A mask is a matrix with as many columns as there are variables in the system, and with as many rows as are needed to cover the dynamics of the finite state machine, the so-called depth of the mask. The negative mask elements denote inputs of this qualitative functional relationship. In the above example, there are four mask inputs. The numbering sequence is immaterial; mask inputs are usually enumerated from left to right and form top to bottom. The single positive mask value denotes the mask output.

The name "mask" is derived from the masks used in grading multiple choice tests. Let the mask matrix be represented by a piece of cardboard. Negative elements are denoted by round holes in the cardboard, whereas the single positive element is denoted by a square hole. It is now possible to shift the mask over the episodical behavior matrix and read out input-output states.

How is the optimal mask determined? It is necessary to define a syntax denoting the ensemble of all feasible masks from which the optimal mask will be selected. This is accomplished by means of the so-called "mask candidate matrix." A mask candidate matrix for the previous example might be specified as following:

$$\begin{array}{c} {}_t\backslash^x \\ t-2\delta t \\ t-\delta t \\ t \end{array} \begin{array}{c} \begin{array}{ccccc} u_1 & u_2 & y_1 & y_2 & y_3 \end{array} \\ \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & +1 & 0 & 0 \end{pmatrix} \end{array} \tag{3.5}$$

In the mask candidate matrix, -1 elements indicate *potential* inputs, whereas the only $+1$ element still indicates the true mask output. 0 elements denote forbidden connections. From the mask candidate matrix, the optimal mask is often determined in a process of exhaustive search, but more effective suboptimal search strategies have also been devised [10].

A general rule for constructing the mask candidate matrix is to cover with the mask the largest time constant of the system that we wish to capture in our model. The mask depth should be chosen as twice the ratio between the largest and the smallest time constants to be captured. A meaningful design procedure could be the following:

- Plot the Bode diagram of the system to be modeled.
- Measure the band width, ω_{3dB}, and the drop-off frequency, ω_0.
- Approximate the largest time constant, t_s, and the shortest time constant, t_r, as

$$t_s \approx \frac{2\pi}{\omega_0} \qquad t_r \approx \frac{2\pi}{\omega_{3\text{dB}}} \tag{3.6}$$

- Choose the mask such that it covers the largest time constant, t_s, whereas the sampling rate, δt, should be no larger than half the shortest time constant, t_r. Thus, the depth of the mask can be computed as:

$$\Delta t \geq t_s \quad ; \quad \delta t \leq \frac{t_r}{2} \tag{3.7}$$

$$depth = \text{round}(\frac{\Delta t}{\delta t}) + 1 \tag{3.8}$$

where Δt is the time interval covered by the mask.

In order to determine the optimal mask from the episodical behavior, constraints limiting the search space can be specified through the mask candidate matrix by blocking out forbidden connections by means of 0 elements, and by specifying the maximum tolerated mask complexity, i.e., the largest number of non-zero elements that the optimal mask may contain.

The exhaustive search process starts by evaluating first all legal masks of complexity two, i.e., all masks containing a single input; it then proceeds by evaluating all legal masks of complexity three, i.e., all masks with two inputs and finds the best of those; it then continues in the same manner until the maximum complexity has been reached. In all practical examples, the quality of the masks will first grow with increasing complexity, eventually reach a maximum, and then decay rapidly. A good value for the maximum complexity is usually five or six.

Each of the possible masks is compared to all others with respect to its potential merit. The optimality of the mask is evaluated with respect to the maximization of its forecasting power.

The Shannon entropy measure is used to determine the uncertainty associated with forecasting a particular output state given any legal input state. The Shannon entropy relative to one input is calculated from the equation:

$$H_i = \sum_{\forall o} \mathrm{p}(o|i) \cdot \log_2 \mathrm{p}(o|i) \tag{3.9}$$

where $p(o|i)$ is the conditional probability of a certain output state o to occur, given that the input state i has already occurred. The term "probability" is meant in a statistical rather than in a true probabilistic sense. It denotes the quotient of the observed frequency of a particular state divided by the highest possible frequency of that state.

The overall entropy of the mask is then calculated as the sum:

$$H_m = -\sum_{\forall i} p(i) \cdot H_i \tag{3.10}$$

where $p(i)$ is the probability of that input to occur. The highest possible entropy $H_{\max}$ is obtained when all probabilities are equal, and a zero entropy is encountered for relationships that are totally deterministic.

A normalized overall entropy reduction H_r is defined as:

$$H_r = 1.0 - \frac{H_m}{H_{\max}} \tag{3.11}$$

H_r is obviously a real-valued number in the range between 0.0 and 1.0, where higher values indicate an improved forecasting power. The optimal mask among a set of mask candidates is defined as the one with the highest entropy reduction.

In the computation of the input/output matrix, a confidence value can be assigned to each row, and this value indicates how much confidence can be expressed in the individual rows of the input/output matrix.

The *basic behavior* of the input/output model can now be computed. It is defined as an ordered set of all observed distinct states together with a measure of confidence of each state. Rather than counting the observation frequencies (as would be done in the case of a probabilistic measure), the individual confidences of each observed state are accumulated. If a state has been observed more than once, more and more confidence can be expressed in it. Thus, the individual confidences of each observation of a given state are simply accumulated. A normalized confidence of each input-output state can then be calculated by dividing the accumulated confidence in that input-output state by the sum of confidences for all input-output states sharing the same input state.

Application of the Shannon entropy to a confidence measure is a somewhat questionable undertaking on theoretical grounds since the Shannon entropy was derived in the context of probabilistic measures only. For this reason, some scientists prefer to replace the Shannon entropy by other types of performance indices that were derived in the context of the particular measure chosen [30] and [63].

There remains one problem. The size of the input/output matrix grows with increasing complexity of the mask, and consequently, the number of legal states of the model grows quickly. Since the total number of observed states remains constant, the frequency of observation of each state shrinks rapidly, and so does the predictiveness of the model. The entropy reduction measure does not account for this problem. With increasing complexity, H_r simply keeps growing. Very soon, a situation is encountered where every state that has ever been observed has been observed precisely once. This obviously leads to a totally deterministic state transition matrix, and H_r assumes a value of 1.0. Yet the predictiveness of the model will be dismal, since in all likelihood already the next predicted state has never before been observed, and that means the end of forecasting. Therefore, this consideration must be included in the overall quality measure.

From a statistical point of view, every state should be observed at least five times [36]. Therefore, an *observation ratio*, O_r, is introduced as an additional contributor to the overall quality measure [41]:

$$O_r = \frac{5 \cdot n_{5\times} + 4 \cdot n_{4\times} + 3 \cdot n_{3\times} + 2 \cdot n_{2\times} + n_{1\times}}{5 \cdot n_{\text{leg}}} \tag{3.12}$$

where:

n_{leg} = number of legal input states;

$n_{1\times}$ = number of input states observed only once;

$n_{2\times}$ = number of input states observed twice;

$n_{3\times}$ = number of input states observed thrice;

$n_{4\times}$ = number of input states observed four times;

$n_{5\times}$ = number of input states observed five times or more.

If every legal input state has been observed at least five times, O_r is equal to 1.0. If no input state has been observed at all (no data are available), O_r is equal to 0.0. Thus, O_r can be used as a quality measure.

With the observation ratio, O_r, the overall *quality of a mask*, Q_m, is then defined as the product of its uncertainty reduction measure, H_r, and its observation ratio,

O_r:

$$Q_m = H_r \cdot O_r \tag{3.13}$$

The optimal mask is the mask with the largest Q_m value.

3.3 Fuzzy Recoding

With the same arguments that are used to defend replacing crisp sets by fuzzy sets in rule-based expert systems and in programmable logic control systems, the advantages of using fuzzy sets can be preserved also in qualitative modeling and simulation. To this end, it is necessary to revise the manner in which the discretization (or recoding) is done.

Two main advantages of using fuzzy sets as level intervals for episodical behavior recoding are that: (i) the fuzzy membership value provides a smooth transition between neighboring levels, and (ii) trajectory behavior can be regenerated from fuzzy episodical behavior using the membership information.

The overall process of this variant of qualitative modeling and simulation that makes use of *fuzzy recoding* for the conversion of quantitative to qualitative signals, i.e., the transition from trajectory to episodical behavior, and utilizes *fuzzy signal regeneration* for the conversion of qualitative to quantitative signals, i.e., the reverse operation, is called *fuzzy inductive reasoning* [41].

In fuzzy inductive reasoning, the definition of level or class values is the same as in crisp inductive reasoning. However, the fuzzy episodical behavior consists now of three matrices rather than only one. Each class value is associated with two other pieces of information: a *fuzzy membership value*, and a *side value*.

A usually bell-shaped curve is employed to define the fuzzy membership function. This function has a maximum value of 1.0 in the middle between the two landmarks that mark the borders of the interval characterizing a given level, and assumes a minimum value of 0.5 at the landmarks themselves. Such a bell-shaped membership function can be calculated as follows:

$$Memb_i = \exp(-\tau_i \cdot (x - \mu_i)^2) \tag{3.14}$$

where x is the quantitative variable to be recoded, μ_i is the algebraic mean between two neighboring landmarks, and τ_i is determined such that the membership function, $Memb_i$, degrades to a value of 0.5 at the landmarks. In some cases, triangular rather than bell-shaped membership functions are being used.

The side function indicates whether the quantitative value is to the left or to the right of the maximum of the fuzzy membership function. The side function assumes a value of 0 at the point where the membership function reaches its maximum, it assumes a value of -1 to the left of that point, and a value of $+1$ to the right of that point.

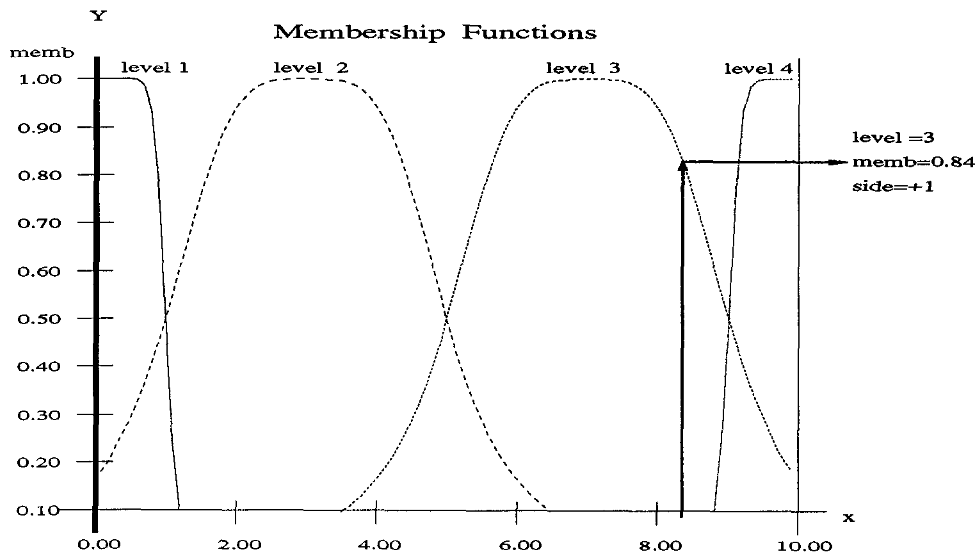

Figure 3.1: Fuzzy recoding

Evidently, the qualitative triple, consisting of the class value, the fuzzy membership value, and the side value, contains the same information as the original quantitative variable. The quantitative value can be regenerated accurately, i.e., without any loss of information, from the qualitative triple. An example of fuzzy recoding is depicted in figure 3.1.

The episodical behavior matrix introduced in the last section is now replaced by three identically sized matrices, one containing the class values, the second storing the fuzzy membership values, and the third maintaining the side values. The process of determining the optimal mask remains the same, except that the measure of confidence of a row of the input/output matrix is now defined as the minimum membership value of all variables associated with that row, rather than as an observation frequency of that particular state.

All modules needed for fuzzy inductive reasoning have been implemented in SAPS-II [41]. SAPS-II is available as either a Matlab toolbox or as a CTRL-C library. The syntax of the SAPS-II function calls and the proper use of these functions are presented in [7].

3.3.1 Determination of Landmarks

One of the more difficult tasks in both crisp and fuzzy recoding is selecting the landmarks for each interval level. No analytical method is known for optimally selecting the landmark values. Yet, the success of the qualitative simulation depends critically on a good selection of the landmarks. In fuzzy logic control, we have the same problem. There, the landmarks are chosen either by experience or by making use of a learning algorithm that modifies the shape of the membership functions while optimizing a performance index.

In this section, an efficient and effective approach for selecting the landmarks is presented that is based on statistical considerations. The applications presented in chapter 5 will demonstrate this approach in more detail, and will elaborate on some trade-offs.

Since, for any class analysis, each legal discrete state should be recorded at least five times [36], a relation exists between the total number of legal levels and the number of recorded data:

$$n_{\text{rec}} \geq 5 \cdot n_{\text{leg}} = 5 \cdot \prod_{\forall i} k_i \tag{3.15}$$

where n_{rec} denotes the total number of recordings, i.e., the total number of observed states; n_{leg} denotes the total number of distinct legal states; i is an index that loops over all variables in the state; and k_i denotes the number of levels for the i^{th} variable. In real applications, the number of variables is usually fixed, and the number of recordings is frequently predetermined. In such a case, the optimum number of levels, n_{lev}, of all variables can be found from the following equation:

$$n_{\text{lev}} = \text{round}(\sqrt[n_{\text{var}}]{\frac{n_{\text{rec}}}{5}}) \tag{3.16}$$

assuming that all variables are classified into the same number of levels. For reasons of symmetry, an odd number of levels is often preferred over an even number of levels. In most practical applications, it is found that either three or five levels for each variable is sufficient.

Once the number of levels for each variable has been selected, the landmarks must be chosen that separate neighboring regions from each other. There are several ways to find a meaningful set of landmarks. The most effective way is based on the idea that the expressiveness of the model will be maximized if each level is observed equally often. In order to distribute the observed trajectory values of each variable equally among the various levels, they are sorted into ascending order, the sorted vector is then split into n_{lev} segments of equal length, and the landmarks are chosen as the arithmetic mean values of the extreme values of neighboring segments.

This approach provides an effective way to select initial landmarks without requiring extensive knowledge of the system to be modeled, and some learning algorithm can then be applied to adjust this set of landmarks based on the recorded data. We shall demonstrate in the following chapters how such a learning algorithm can be devised, and when the learning algorithm is necessary.

CHAPTER 4

Fuzzy Adaptive Recurrent Counterpropagation Neural Networks

As fuzzy set theory and fuzzy logic have the advantage of being able to integrate qualitative information into a quantitative information system, they can be employed to process linguistic information stemming from human knowledge describing, in qualitative terms, aspects of the dynamics of a system without possessing precise quantitative knowledge about those aspects in the form of mathematical equations; yet they can incorporate the so obtained information in an otherwise quantitative modeling and simulation environment. The interfaces between the quantitative and the qualitative subsystems are the fuzzifiers and defuzzifiers, respectively.

Fuzzy logic has been one of the most active research fields lately both in the contexts of controller design and system identification. It is believed that fuzzy logic, due to its capability of mimicking the imprecise yet highly adaptive human reasoning processes, will assume the role of a corner stone technology amongst the intelligent system design methodologies of the 21^{st} century.

Most fuzzy systems operate by applying knowledge inferencing techniques to a knowledge base composed of fuzzy rules. The core of the knowledge base is normally constructed using knowledge acquisition from human experts. However, the knowledge base needs to be fine tuned to the task at hand. To this end, it is common practice to adjust the shape of the fuzzy membership functions by means of some adaptation mechanism. However, the inherent nonlinearity of fuzzy logic makes the learning mechanism difficult to develop.

Although classical optimization techniques may accomplish the desired fuzzy system adaptation in some cases, a more promising approach employs neural networks as the optimization techniques for adjusting the membership functions. The outputs of the neural network are the optimized parameters of the fuzzy system. Learning is then accomplished by adjusting the weights and nonlinear activation functions of the neural network, rather than by modifying the fuzzy system parameters directly. This approach has been briefly introduced in section 2.2.2 of this dissertation.

In this chapter, an alternative approach for constructing fuzzy systems shall be introduced, a technique that relies entirely on input-output training data, avoiding the need to employ knowledge obtained from human experts. Several eminent researchers in artificial intelligence, such as Marvin Minsky, have identified the process of *knowledge acquisition* as the most serious bottleneck in the whole of artificial intelligence. Extracting knowledge from human experts is a tedious and highly heuristic enterprise. No formal techniques are known that could accomplish this knowledge extraction in a systematic fashion. Thus, a methodology capable of extracting the required knowledge directly and in a fully automated fashion from measured data streams should be highly welcome.

In our methodology, we use generalized counterpropagation neural networks as the fuzzy knowledge lookup mechanism for generating fuzzy output. This approach differentiates itself from other fuzzy system techniques in several aspects. First, it fixes the membership functions from the very beginning instead of fine tuning them either through the use of a learning algorithm or by trial and error. Second, it uses the counterpropagation network as a one-pass regression procedure to select fuzzy knowledge embedded in the network instead of employing normal fuzzy inferencing techniques applied to a fuzzy rule base. By combining this technique with the fuzzy inductive reasoning methodology introduced in the previous chapter, we can construct a *Fuzzy Adaptive Recurrent Counterpropagation Neural Network* from input-output training data alone. This approach provides a highly efficient and effective way of integrating a fuzzy logic system into the qualitative modeling paradigm.

A comparable approach called *fuzzy basis function methodology* proposed in [72] also fixes the fuzzy membership functions from the beginning, then uses an orthogonal least-square learning algorithm as the selection function to generate crisp output. The authors also prove that their approach is capable of uniformly approximating any real continuous function.

4.1 Implementing Finite State Machines Using Generalized Counterpropagation Neural Networks

We have already explained in section 2.1.2 how general counterpropagation neural networks operate. In this section, we are going to introduce a much simplified Counterpropagation Neural Network (CNN) that had first been proposed in [7]. We shall then show how the basic CNN can be generalized for rapid implementation of finite state machines. This so enlarged CNN will be called Generalized Counterpropagation Neural Network (GCNN).

The basic properties of the CNN are best introduced by means of a simple example: the infamous XOR problem. It is desired to design a neural network that can reproduce the behavior of an XOR gate:

u_1	u_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Table 4.1: Truth table of XOR gate.

The counterpropagation neural network for this problem is shown in figure 4.1.

The first layer, the so-called *Kohonen layer*, consists of four simple perceptrons with threshold values of 1.0. Its weight matrix, $\mathbf{W^1}$, is the transpose of the matrix that is composed of a horizontal concatenation of all legally possible input patterns

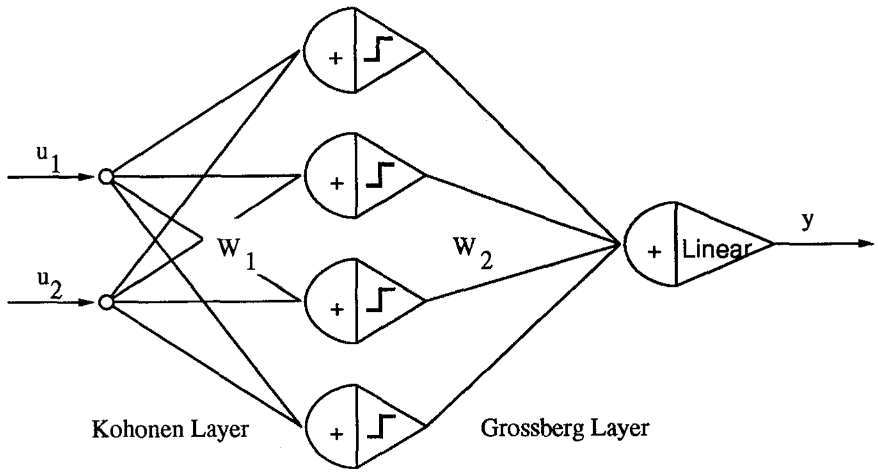

Figure 4.1: Counterpropagation network for XOR functions

represented as column vectors. It is similar to the truth table of the input patterns in table 4.1:

$$\mathbf{W}^1 = \begin{pmatrix} -1 & -1 \\ -1 & +1 \\ +1 & -1 \\ +1 & +1 \end{pmatrix} \tag{4.1}$$

except that the representation of *false* is now -1 rather than 0, while the representation of *true* is still $+1$. In the sequel, superscripts shall always point to the layer of the neural network, whereas subscripts shall denote elements of vectors or matrices.

If a particular input pattern, e.g. $\mathbf{u}^1 = (-1 \quad +1)^T$, is shown to the neural network, the matrix multiplication inside the Kohonen layer produces the vector $\mathbf{x}^1 = \mathbf{W}^1 \cdot \mathbf{u}^1 = (0 \quad +2 \quad -2 \quad 0)^T$, and the output activation functions of the four neurons produce the vector $\mathbf{y^1} = \mathcal{F}(\mathbf{W^1} \cdot \mathbf{u^1}) = (0 \;\; 1 \;\; 0 \;\; 0)^{\mathrm{T}}$ accordingly, where $\mathcal{F}$ is the output activation function of the perceptron, a step function with a threshold value of 1.0. Thus, the Kohonen layer acts as a *powerset generator*. It contains as many perceptrons as there exist different legal input patterns. If the CNN has k different binary inputs, a complete Kohonen layer will consist of 2^k perceptrons, the thresholds of which can be set to $d_i^1 = k - 1$. There is no competitive learning as in the case of Hecht-Nielson's CNNs. Both the weight matrix, $\mathbf{W^1}$, and the threshold vector of the perceptrons, $\mathbf{d^1}$, can be fixed from the onset, and the perceptrons do not need to cooperate with each other at all.

The second layer, or *Grossberg layer*, consists of as many linear neurons as there are output variables; in the case of the XOR example, a single linear neuron. The weight matrix, $\mathbf{W^2}$ consists of a horizontal concatenation of the output patterns. In the XOR example, it is similar to the transpose of the output patterns in the truth table, thus:

$$\mathbf{W^2} = \begin{pmatrix} -1 & +1 & +1 & -1 \end{pmatrix} \tag{4.2}$$

The Grossberg layer acts as a *selector*. The product $\mathbf{W^2} \cdot \mathbf{y^1}$ picks out the output pattern that corresponds to the input pattern that was shown to the CNN. As in the case of the Kohonen layer, no learning is necessary. The weight matrix, $\mathbf{W^2}$, can be predetermined just like the weight matrix $\mathbf{W^1}$.

In the remainder of this chapter, it is this type of neural network that will be referred to as Counterpropagation Neural Network (CNN). A CNN is a *binary neural network* in that all its inputs and outputs are binary.

The output layer of the CNN would not necessarily have to be binary. The $\mathbf{W}^2$ matrix elements could assume any values. In such a case, the network shall be called *Generalized Counterpropagation Neural Network* (GCNN).

A Finite State Machine (FSM) is similar to a logic truth table, except that its variables may employ multi-valued logic. Often, the number of output variables equals the number of input variables, and the semantic meaning of the FSM is a transition from the input state to the output state, i.e., the current values of the set of input variables represent the current state of the system, whereas the values of the set of output variables represent the next state of the system. This is where the name "finite state machine" comes from. However, the concept of a FSM can be generalized by defining it to mean an arbitrary truth table with i input variables and o output variables, each of which can assume a finite set of values that can be represented either by integers, such as '1,' '2,' and '3,' or by symbols, such as 'small,' 'medium,' and 'large.'

The purpose of the CNN is that of a table-lookup function. Whenever one of the input patterns is shown to it, it reacts by presenting the corresponding output pattern at its output.

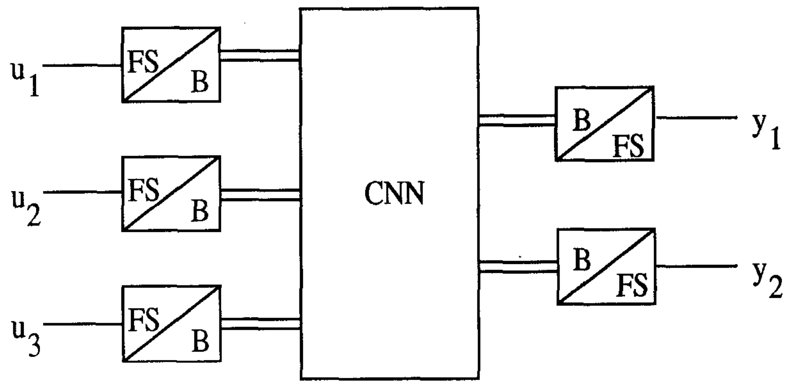

Figure 4.2: Counterpropagation network for Finite State Machine

The realization of this finite state machine using a CNN is shown in figure 4.2. It consists of a set of Finite State to Binary (FS/B) converters, followed by a regular CNN, followed by a set of Binary to Finite State (B/FS) converters. The example shown in figure 4.2 contains three input signals, u_1 to u_3, and two output signals, y_1 and y_2. Each of the five signals is a multi-valued logic signal. The FS/B converter converts one multi-valued logic signal into multiple binary signals. For example, if u_1 has eight levels, it can be converted into three separate binary signals, u_{1a} to u_{1c}, etc. Each variable can be converted separately, thereby providing means to parallelize also this operation.

The B/FS converters can themselves be realized as GCNNs. Since cascaded CNNs can always be amalgamated into one, the B/FS converters could be combined with the CNN to their left forming a single GCNN.

If a system involves a feedback and this feedback provides the necessary memory to capture the system dynamics through a Finite State Machine, the neural network

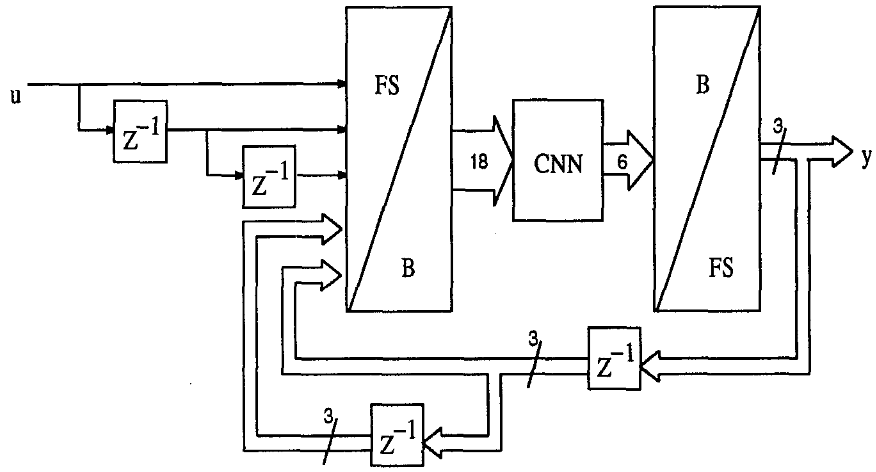

Figure 4.3: Recursive counterpropagation network

has become recurrent. Such a CNN is called Recurrent Counterpropagation Neural Network (RCNN). An illustrative example is presented in figure 4.3. The RCNN of figure 4.3 represents a dynamic system with a single input and three outputs. The input and the three outputs are multi-valued logic signals with four levels each. Thus, two-bit FS/B converters suffice to map these multi-valued logic signals into binary signals. It is assumed that a history of two sampling intervals of input and outputs suffices to capture the dynamics of the system. The boxes denoted as z^{-1} in figure 4.3 represent delays.

4.2 Implementing RCNNs Using Fuzzy Recoding

Continuous-time systems can be approximated by difference equations just as well as multi-valued logic systems can. Consequently, a multi-variable real-valued function generator, such as a backpropagation-trained feedforward network (BNN) together with appropriate feedback loops containing delays, can be used to approximate a continuous-time system with arbitrarily many inputs, states, and outputs.

However, since BNNs are so slow in their training, it would be useful if the previously presented GCNN architecture could be generalized to process real-valued variables as well as multi-valued logic variables. This would enable us to synthesize multi-variable continuous function generators directly rather than having to train them first.

One obvious solution is to preprocess continuous input and output variables using standard Analog to Digital (A/D) converters. An example of a continuous-time system with three inputs and two outputs is shown in figure 4.4. It is assumed that two sets of past values suffice to adequately characterize the plant dynamics. It is furthermore assumed that 12-bit A/D converters are used, converting the $3+3+3+2+2=13$ analog inputs to $12 \cdot 13 = 156$ digital inputs, and the two analog outputs to $12 \cdot 2 = 24$ digital outputs. A standard CNN can now be devised that relates the 156 binary inputs to the 24 binary outputs.

After identification of the Counterpropagation Neural Network, the outputs of the CNN are converted back to continuous signals by Digital to Analog (D/A) converters. Continuous inputs signals newly arriving at the neural network are first converted to binary input signals using A/D converters. They are then processed by the CNN. The resulting binary outputs are converted back to continuous output signals using

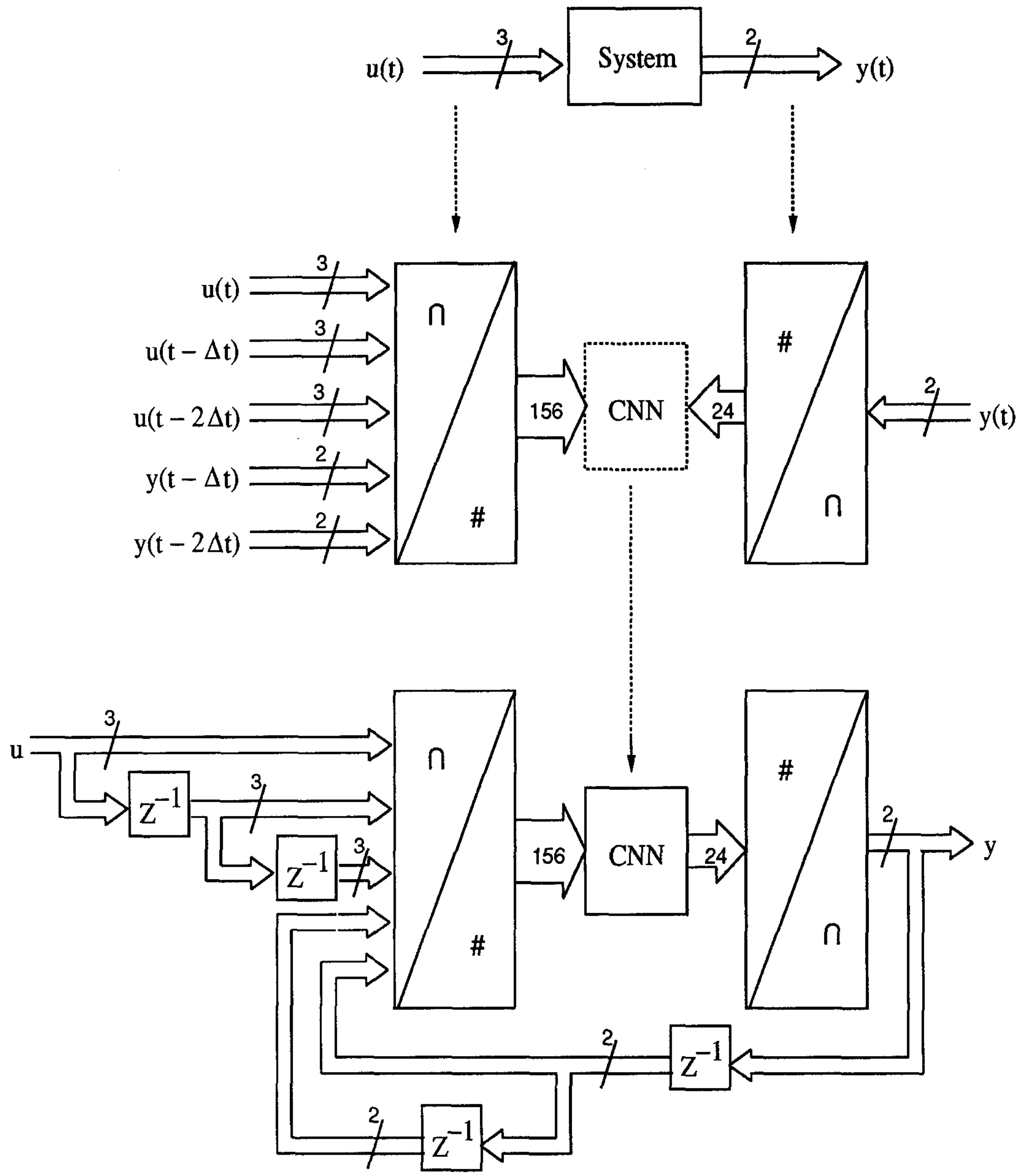

Figure 4.4: Recursive counterpropagation network for continuous signals

D/A converters. The so generated continuous output signals are then delayed and fed back to the input layer.

There are two problems with this approach. Since the discrete representation of the system is only an approximation, it is not evident that the input/output relationship is still fully deterministic, i.e., it is conceivable that, for the same set of input values, different output values can be observed. This should not be a big problem. If the representation is chosen carefully, the input/output relationship should be fairly deterministic, or at least, if different digital output values result, they should correspond to very similar analog output values. Consequently, any of them may be chosen during forecasting, and it is usually quite acceptable to always select the most frequently occurring input/output combination. Thus, the CNN can still be made fully deterministic.

The more serious problem with this approach is the sheer size of the CNN. Since the CNN has 156 binary inputs, there exist conceptually 2^{156} legal combinations or input states. Thus, a complete CNN should have a hidden layer of length 2^{156}, which is quite unreasonable. Although not all of these combinations may occur in practice, the required number of training data records for the CNN would still be unmanageably large.

An alternative solution for this problem is to use Fuzzy Recoding as introduced in section 3.3, which converts quantitative signals into *qualitative triples* consisting of a *class value* of the enumerated type, a *side value* of the binary type, and a real-valued *fuzzy membership function.*

The class values are of the enumerated type, and the side value is even binary. Thus, it is easy to generate an RCNN that translates the class and side values of observed inputs into the class and side value of an observed output. Since the discretization is now much more coarse than before, the input/output relationship will be even more deterministic, and the number of discrete inputs to the CNN is drastically reduced. Experience has shown that one usually gets away with three to eight classes [8]. Thus, fuzzy two-bit converters will often suffice. Together with the binary side value, an analog signal is thereby discretized into three binary signals and a real-valued fuzzy membership function. In the presence of abundant training data, fuzzy three-bit converters may yield slightly more accurate results. Fuzzy three-bit converters map analog signals into four binary signals and a real-valued fuzzy membership function.

Fuzzifiers replace the A/D converters. In the example of figure 4.4, the fuzzifiers now convert 13 analog inputs into 39 binary inputs plus 13 fuzzy membership values. *Defuzzifiers* replace the former D/A converters. In the same example, they now convert six binary outputs and two fuzzy membership values back into two analog outputs. The approach is illustrated in figure 4.5. The CNN will only predict the class and side values, which is a rather crude approximation. The purpose of the box marked with '?' is to provide a means for smoothly interpolating between neighboring discrete predictions.

The outlined approach can be viewed in the context of fuzzy measures. Most fuzzy inferencing and defuzzification techniques, such as the *Mean of Maxima* (MoM) technique and the *Center of Area* (CoA) technique, use the information stored in the tails of the overlapping fuzzy membership functions of the inputs to generate multiple discrete output values, and smoothly interpolate between them using the relative importance of the input classes, i.e., the membership value of the output is inferred by looking at the input space alone [14]. The advantage of this approach is that no training data need to be stored away. If training data are used at all, they only serve to help with the shaping of the fuzzy membership functions.

It was shown in [46] that –at least in a data-rich environment– it may be more beneficial to store away the available training data set, to look at the input space for computing a distance function between the new vector of inputs and the vectors of inputs in the training data set, use this information to determine the relative importance of the inputs stored in the training data set, pick out the five nearest neighbors, and finally look at the output space to smoothly interpolate between the output values of these five nearest neighbors from the training data set. This is the approach we are going to implement in the '?'-box.

It was also demonstrated in [46] that the membership value of an output can be written as a linear combination (weighted sum) of the membership values of the same

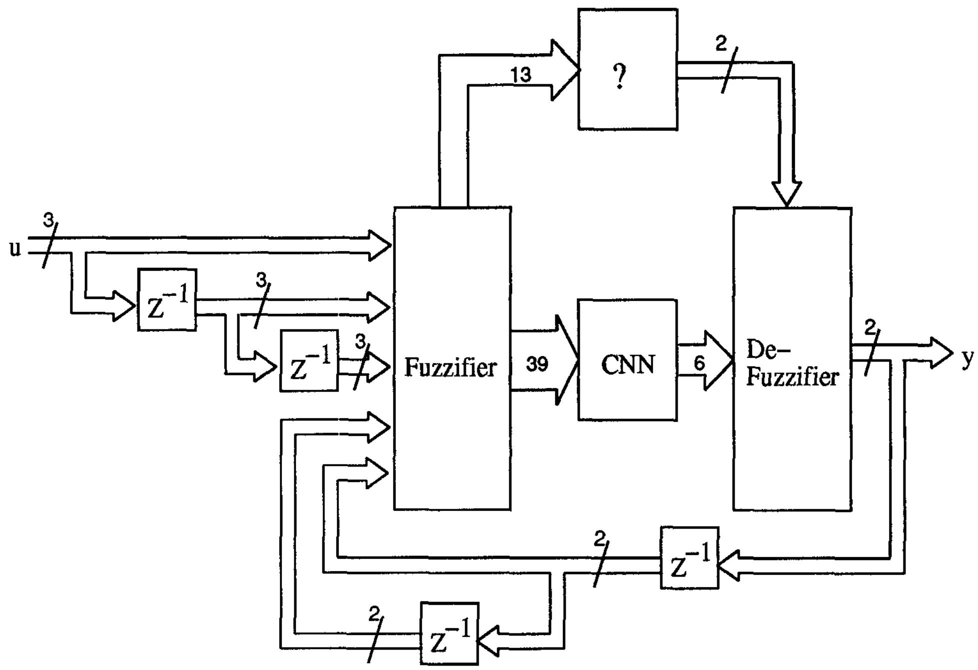

Figure 4.5: Fuzzy inferencing for counterpropagation network

output from previous occurrences of the same input/output pattern. The weights themselves are deterministic functions of the membership and side values of the inputs. This fuzzy inferencing technique has been coined the *Five-Nearest-Neighbor* (5NN) rule. The functions are independent of the shape of the fuzzy membership function and of the landmark values that separate neighboring classes.

We can employ a feedforward multilayer neural network using a back-propagation training algorithm to implement the 5NN rule. This training process is very slow. However, this Backpropagation Neural Network (BNN) will be shown to be *application independent*, and therefore, it can be trained off–line once-and-for-all.

There is still a problem with this approach. Although the number of inputs of the CNN has been reduced from formerly 156 to 39, there are nevertheless 2^{39} legal states left in the system, still a frighteningly large number. The *Fuzzy Inductive Reasoning* (FIR) approach [7, 8] has demonstrated that not all of the possible inputs are needed to predict the output correctly. A selector function picks a subset of the available inputs. This subset is problem-dependent. The FIR methodology uses the *optimal mask* as the selector function. A different optimal mask (selector function) is needed for each of the output variables. Each optimal mask requires, on the average, somewhere between three to five inputs. Figure 4.6 illustrates the proposed architecture for the same example that was already shown in figure 4.5.

In this example, it is assumed that the optimal mask to compute y_1 has four inputs, whereas the optimal mask to compute y_2 has only three inputs. Four classes (fuzzy two-bit converters) are used for all analog variables. The top CNN that is used to compute y_1 has now 12 binary inputs corresponding to $2^{12} = 4096$ states, while the bottom CNN, used to compute y_2, has nine binary inputs corresponding to $2^9 = 512$ different states. These numbers are now quite reasonable. The CNNs predict the class and side values of the output from the class and side values of the inputs. The BNN predicts the membership value of the output from the membership and side values of the inputs. Notice that the two BNNs are identical (they use the same weights) although the number of input variables is different. The BNN was trained such that unused membership inputs can be fixed at 0.0.

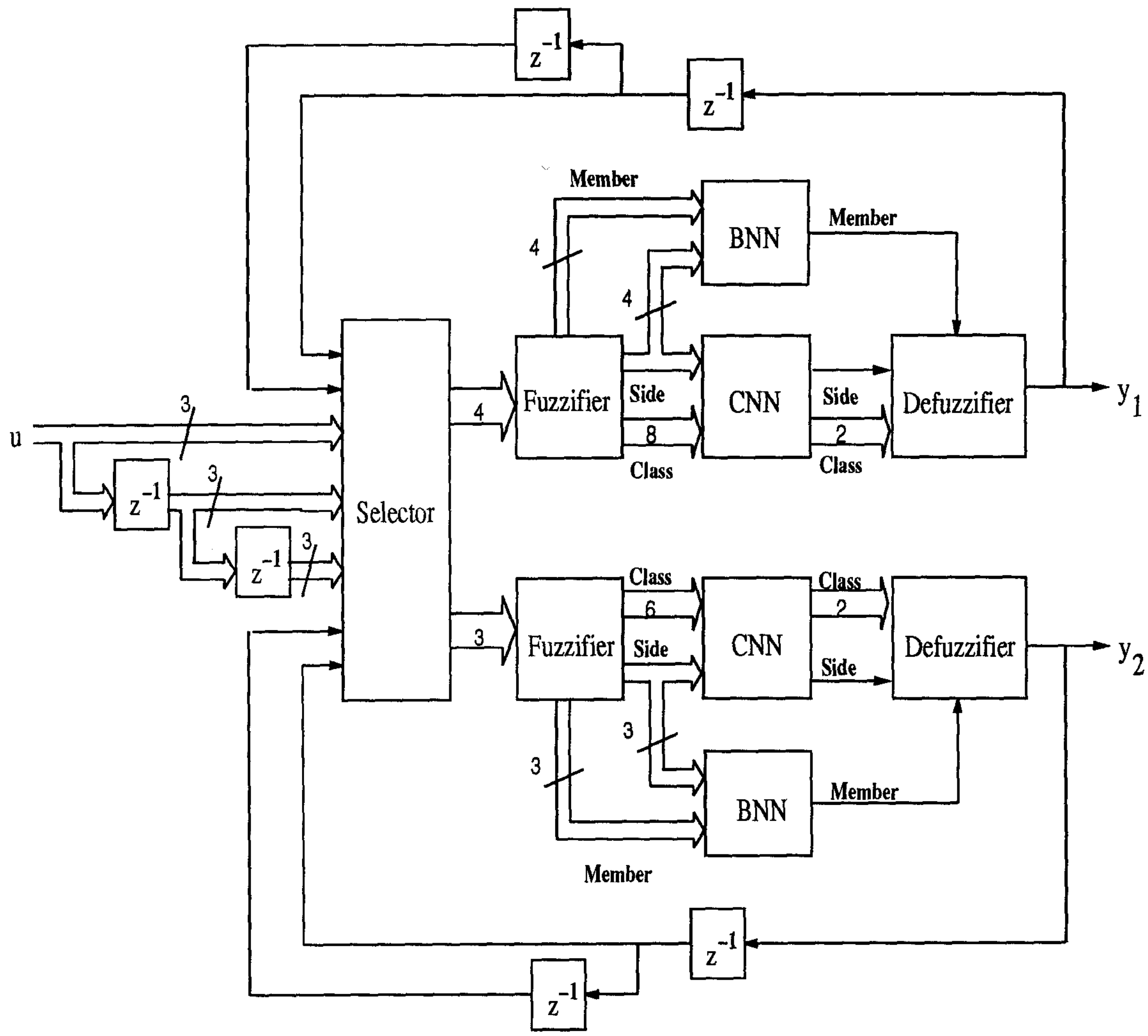

Figure 4.6: Basic FRCNN architecture

The network architecture depicted in figure 4.6 is called *Fuzzy Recurrent Counterpropagation Neural Network*, or FRCNN.

4.3 Implementation of the FRCNN Architecture

In this section, a practical implementation of the FRCNN architecture shall be described as well as some details of the design of the BNN implementing the 5NN algorithm.

As we can always separate a multi-output system into a number of single-output systems in the implementation of the FRCNN architecture, it suffices to consider the FRCNN implementation of multi-input/single-output systems.

Let us assume that we are dealing with a system with m inputs and a single output. A new input vector, $\mathbf{r}$, of length m arrives. The five nearest neighbors are those input-output training pairs, $\mathbf{t}_j = (\mathbf{r}_j^{\mathrm{T}}, y_j)^{\mathrm{T}}$, with minimal distances of their respective input spaces:

$$|\mathbf{r} - \mathbf{r}_j| \stackrel{!}{=} \min_{\forall j} \tag{4.3}$$

The i^{th} new input variable has a quantitative value of r_i. The recoded r_i variable has a class value of c_i, a side value of s_i, and a membership value of M_i. The j^{th} nearest old neighbor of the i^{th} new input variable has a quantitative value of r_{ij}. The recoded r_{ij} variable has a class value of c_{ij}, a side value of s_{ij}, and a membership value of M_{ij}.

For each input, i, so-called *position functions*, p_{ij} and p_i, can be defined as follows [46]:

$$p_{ij} = c_{ij} + s_{ij} \cdot (1.0 - M_{ij}) \tag{4.4}$$

$$p_i = c_i + s_i \cdot (1.0 - M_i) \tag{4.5}$$

The position functions represent normalized versions of defuzzified signals. They are normalized since the lowest class ($c_i = 0$) corresponds to a position variable range from 0.0 to 0.5, the next higher class ($c_i = 1$) corresponds to the range [0.5, 1.5], and so on. Each class, except for the lowest and highest, is exactly equally wide. The slope of all membership functions is identical relative to the normalized position variables.

All p_{ij} elements can be combined to form the vector $\mathbf{p}_j$, and all p_i elements together make up the vector $\mathbf{p}$:

$$\mathbf{p}_j = (p_{ij} \quad ; \; i = 1, 2 \cdots, m)^T \tag{4.6}$$

$$\mathbf{p} = (p_i \quad ; \; i = 1, 2, \cdots, m)^T \tag{4.7}$$

where m is the number of input elements, $\mathbf{p}_j$ are the position vectors of the five nearest neighbors from the training data set, and $\mathbf{p}$ is the position vector for new testing data.

Distance functions, d_j, can now be computed as follows:

$$d_j = |\mathbf{p} - \mathbf{p}_j| \tag{4.8}$$

The largest of these distance function is called $d_{\mathbf{max}}$.

Absolute weights are then computed using the equation:

$$w_{\mathbf{abs}_j} = \frac{d_{\mathbf{max}} - d_j}{d_{\mathbf{max}}} \tag{4.9}$$

The sum of all absolute weights is:

$$s_w = \sum_{\forall j} w_{\mathbf{abs}_j} \tag{4.10}$$

From this, it is possible to compute relative weights as follows:

$$w_{\mathbf{rel}_j} = \frac{w_{\mathbf{abs}_j}}{s_w} \tag{4.11}$$

The relative weights are numbers in the range $[0.0, 1.0]$. Their sum adds up to 1.0.

Since equation (4.4) depends not only on the membership values of the previously observed neighbor inputs, but also on their class and side values, all these quantities would have to be passed from the CNN to the BNN to enable the BNN to implement equation (4.4). In order to avoid this, the old membership values are modified as follows:

$$\bar{M}_{ij} = s_i \cdot ((c_i - c_{ij}) + (s_i - s_{ij}) + s_{ij} \cdot M_{ij} \tag{4.12}$$

where c_{ij} stands for the current class value of the qualitative triple $< c_{ij}, s_{ij}, M_{ij} >$, which is used in equation (4.4), and c_i stands for the class into which the qualitative triple is to be mapped. The corrected qualitative triple has a class value of c_i, a side value of s_i, and an extended membership value of $\bar{M}_{ij}$. Equation (4.12) is not valid for $s_i = 0$, but this doesn't matter, since, while individual data points may accidentally assume a value of $M_{ij} = 1$ together with $s_{ij} = 0$, the "boxes" inside the CNN into which the five neighbors must be mapped never assume a value of $s_i = 0$.

This modification allows to rewrite equation (4.4) as:

$$p_{ij} = c_i + s_i \cdot (1.0 - \bar{M}_{ij}) \tag{4.13}$$

thus, only the modified membership values are still needed.

In [46], it was proposed to simply search the input space for the five nearest neighbors, irrespective of whether they belong to the same class or not. However, this solution puts a heavy load on the sorting algorithm. The *five nearest neighbors* (5NN) algorithm requires exhaustive search and comparison, an expensive proposition in the context of a real-time modeling and simulation environment.

The computational effort can be reduced by searching for the five nearest neighbors only within the (class/side) combination of the testing data point. These five neighbors are not necessarily the true nearest neighbors, since it is perfectly possible that the testing data point is close to the border of the (class/side) combination to which it belongs, and the true five nearest neighbors all belong to a different (class/side) combination. The modified *fuzzy 5NN method* represents a compromise between our desire for maximizing the approximation accuracy of the method and that of keeping the computational effort low.

The fuzzy 5NN method needs one additional rule. If the particular (class/side) combination of the input vector of the current testing data record cannot be found at least five times in the training data set, the training data must be searched for input patterns that vary only in their side values, and these records are then used as neighbors. If there are even not enough of those patterns available in the training data set, neighboring cells of the input space are searched. The fuzzy 5NN method still requires searching, but at least not over the entire training data set. It is thus computationally much cheaper that the true 5NN method.

A yet cheaper alternative may be to use a predefined set of five neighbors (5N) for each (class/side) combination, rather than searching the input space for the nearest five neighbors (5NN). This 5N algorithm can be implemented efficiently in a GCNN, since the Grossberg layer of the GCNN can generate the required $\bar{M}_{ij}$ values instantaneously.

Using the 5N algorithm instead of the fuzzy 5NN algorithm will lead to a further degradation in performance, as will be discussed in chapter 5. However, this does not have to be so. The neural searching function, described briefly in section 2.1.3, allows us to come up with a mixture between the 5N and the 5NN algorithm. In the mixed algorithm, the GCNN finds *all* neighbors within the (class/side) combination, and if there are less than five of those, adds all neighbors within the neighboring classes as well. A neural searching network is then placed in between the CNN and the BNN to determine the five nearest neighbors among those. These will certainly be the same five nearest neighbors that the fuzzy 5NN algorithm would come up with. Chapter 6 will explore this idea in a little more detail.

We still need to prove that the BNN is indeed application independent. In equations (4.13), (4.8), (4.9), (4.10), and (4.11), only current and past values of the fuzzy membership, class, and side functions of the *input* variables are being used. The *output* variable does not appear anywhere in these equations. A BNN can be used to emulate these equations. Evidently, this BNN must be application independent, since it does not contain any information about the input/output relationship of the system to be modeled.

Such a BNN linked with other FRCNN architecture components is shown in figure 4.7. It is a general-purpose BNN that can be used for *any* system with up to five input variables and up to five data records per input variable in the training data set.

The BNN has been designed as a group of cascaded multi-layer feedforward neural networks. Each subnetwork consists of an input layer, two hidden layers, and an output layer trained by means of backpropagation, and implements one of the fuzzy inferencing equations. The overall BNN contains 32 layers with altogether 958 neurons.

The BNN was trained once using random input by comparing the BNN output with the output obtained from fuzzy inferencing. The training process converged very slowly. However, it was a one-time effort. The same BNN without a single modification has been used thereafter in *all* applications discussed in chapter 5.

According to the 5NN (or 5N) algorithm, the membership function of the output can now be computed as follows:

$$M^{o} = \sum_{\forall j} w_{\mathrm{rel}_j} \cdot M_j^{o} \tag{4.14}$$

where w_{rel_j} are the outputs of the BNN, and M_j^{o} are the –hitherto unused– membership functions of the outputs of the five neighbors found in the fuzzy inferencing process.

Equation 4.14 can be realized by a neural network with five inputs and a single linear neuron for the output. The network will be called *Linear Neural Network* (LNN). The membership values of the output variable from the training data set, M_j, are fed into this LNN as inputs, whereas the relative weights are multiplied into the LNN as weights.

4.4 Learning Mechanisms

Many technical applications are slowly time-varying, i.e., their parameters are not truly constant. This has to do with various factors. The most typical factor that leads to slowly time-varying behavior is temperature dependence of the behavior of some system components.

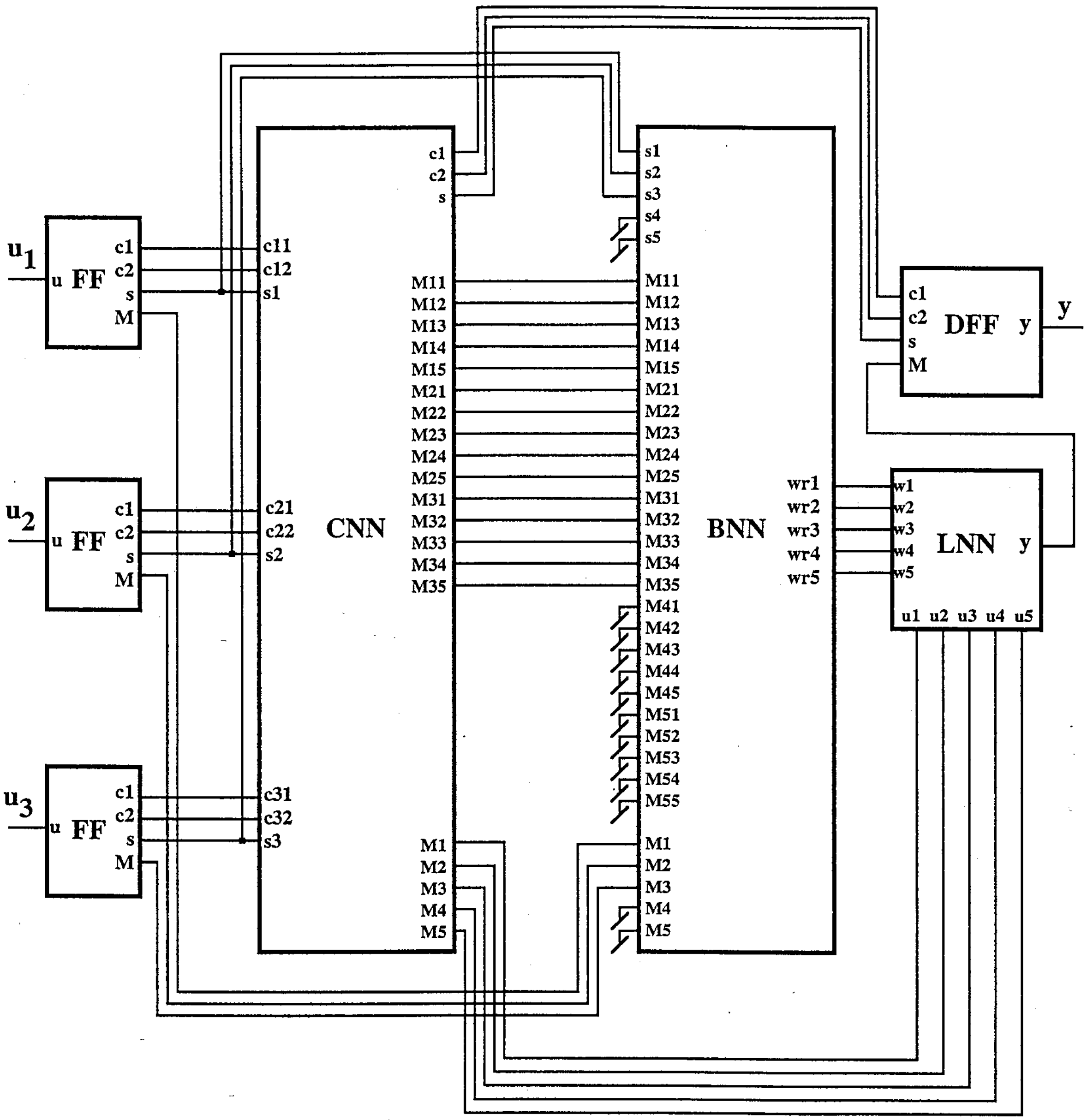

Figure 4.7: Implementational issues of FRCNN architecture

A good model should be able to adapt itself to changes in the system as time passes. Neural networks use on-line self-learning to update their own behavior based on observed modifications in the system's behavior. This is an adaptation mechanism. Adaptation mechanisms have also been reported for fuzzy systems. Often, this is accomplished by adjusting the shape of the membership functions on-line. Thus, the fuzzy inferencing scheme is slightly modified, whereas the defuzzification stage remains the same.

This effect can easily be implemented in the FRCNN architecture. Once a new pattern has been processed, a search determines whether this pattern has ever before been observed. If this is not the case, an additional perceptron is added to the Kohonen layer of the CNN, and the new input/output pattern is added as an additional row of its W^1 matrix and as an additional column of its W^2 matrix.

This mechanism can be used both for adaptation and for on-line learning. In principle, it is possible to start out with an empty CNN. Of course, the predictions made by this empty CNN will be dismal in the beginning, but each newly observed pattern will be stored in the CNN at once, and already after a short while, the CNN will start to become effective.

If a pattern has been observed before, the Kohonen layer will not be enlarged. However, it is possible that this pattern has been observed less than five times before. In this case, only the Grossberg layer is enhanced by adding the newly observed membership information as additional "old" observation to it, possibly replacing a previous entry with M_{ij} lying outside the range [0.5, 1.0], i.e., an entry that had been borrowed from a neighboring class and mapped into the class of interest because of a lack of a sufficiently large number of observations of that class.

Finally, if a pattern has indeed been observed at least five times before, it is possible to replace, in the Grossberg layer, the oldest observation of the same pattern by the newest one in a round-robin fashion. This is then truly a mechanism for taking care of slowly-varying system behavior.

A FRCNN that implements these adaptation mechanisms will henceforth be called *Fuzzy Adaptive Recurrent Counterpropagation Neural Networks*, or FARCNN.

In section 3.3.1, we introduced algorithms to decide, from the total number of recorded data, i.e., from the size of the training data set, how many classes should be used in the recoding of individual variables. We also introduced an algorithm for the determination of the landmarks for each of these classes.

However, there are situations where the number of classes needs to be settled in advance to ensure a sufficiently high degree of *expressiveness* of the qualitative information, yet there are not enough training data available to support this decision. The price to be paid for such a choice is a reduction in *predictiveness* of the qualitative model. However, albeit the fact that a reduction in predictiveness may indeed be unavoidable, it is essential to minimize the negative effects of having to work with too many classes.

This is a similar situation to that of a fuzzy system with an incomplete rule base, i.e., a system, the fuzzy region of which is not exhaustive in the antecedent parts.

In the sequel, we shall discuss some simple learning strategies that can be applied to adjust the landmarks, originally determined using the approach suggested in section 3.3.1. We shall also discuss a slightly modified fuzzy inferencing scheme, in which the absolute weights, w_{abs_j}, are computed using a different formula, in order to improve the correctness of the prediction for testing data that have been previously observed and are stored in the training data set.

Let us begin by looking at the problem where insufficient training data are available to provide at least five neighbors for each (class/side) pair, or even worse, where insufficient training data are available to assign five neighbors to every class. In fuzzy systems, this kind of problem is called the "incompleteness of rule base" problem. Usually, a "completeness factor" of 0.5 is required for fuzzy logic control systems, in order to guarantee that there is at least one rule whose antecedent significantly matches every possible input, and that the dominant rule is associated with a membership value greater than 0.5 [38].

The property of completeness needs to be addressed even before designing the fuzzification interface, such that the process of the fuzzy partitioning of the input space should preserve the completeness property as well as possible. If there are still unavoidable cases of not satisfying the completeness property for some of the classes, additional rules are added, derived from general human engineering knowledge, that should overcome these deficiencies.

[62] elaborates on two methodologies that are commonly employed in image segmentation and that can be applied to the process of region completion. The *region growing approach* fills up empty regions by taking the average of nonempty neighbors. This method iterates over all empty regions with nonempty neighbors until no empty regions remain. The *weighted averaging approach* takes all nonempty regions and uses a distance function to empty regions as a similarity relation to fill up the empty regions. This method can be applied in a single pass.

In the FRCNN approach, the landmarks for the output classes are first computed using the algorithm explained in chapter 3, placing an equal number of previous observations into each of the classes. To this end, the previously observed output trajectory values are sorted into ascending order, the sorted vector is then split into several segments of equal length, and the landmarks are chosen as the arithmetic mean values of the extreme values of neighboring segments. However, when using equation (4.9) with either the 5NN or the 5N algorithm, the predicted output class level might not be in the target class. We can use the training data to adjust the landmarks in order to minimize the total number of misplaced output class values. This can be accomplished using the optimization algorithm described below.

Assume that the output class c_i is constrained by the two landmarks l_{i-1} and l_i. The breadth of class c_i is thus $(l_i - l_{i-1})$. We call the center point of this class $\hat{y}_i$, and the distance from either of the landmarks to the center point δ_i. These two quantities can be computed as

$$\hat{y}_i = \frac{(l_i + l_{i-1})}{2} \tag{4.15}$$

$$\delta_i = \frac{(l_i - l_{i-1})}{2} \tag{4.16}$$

In order to determine whether the predicted output y is in the correct class, a position function η_i can be defined as follows:

$$\eta_i = \delta_i - \mathrm{abs}(\hat{y}_i - y) \tag{4.17}$$

If y is in the correct class c_i, then $\eta_i \geq 0$, otherwise $\eta_i < 0$.

We notice that the step function of η_i has the following property:

$$\mathrm{step}(\eta_i) = \begin{cases} 1 & \text{if } y \text{ is in the correct class} \\ 0 & \text{otherwise} \end{cases} \tag{4.18}$$

If either the 5NN or the 5N algorithm is applied, the total number of misplaced classes is

$$\mathcal{E}_i = 5 - \sum_{j=1}^{5} \mathrm{step}(\eta_{ij}) \tag{4.19}$$

The step(η) function can be approximated by the sigmoid function

$$\mathrm{step}(\eta) \approx \mathrm{sigmoid}(\eta) = \frac{1}{1 + e^{-\eta}} \tag{4.20}$$

We can adjust the distance from either of the landmarks to the center of the class by $\Delta\delta_i$:

$$\delta_i(k+1) = \delta_i(k) + \Delta\delta_i(k) \tag{4.21}$$

Using a gradient algorithm, we can make

$$\Delta\delta_i \propto -\frac{\partial\mathcal{E}_i}{\partial\delta_i} \tag{4.22}$$

Dropping the index i, we can thus write:

$$\begin{aligned}
\Delta\delta &= \rho\frac{\partial\mathcal{E}}{\partial\delta} \\
&= \rho\frac{\partial\mathcal{E}}{\partial\eta}\frac{\partial\eta}{\partial\delta} \\
&= \rho\sum_{j=1}^{5}\frac{e^{-\eta_j}}{(1+e^{-\eta_j})^2}\frac{\partial\eta_j}{\partial\delta} \\
&= \rho\sum_{j=1}^{5}\frac{e^{-\eta_j}}{(1+e^{-\eta_j})^2}
\end{aligned} \tag{4.23}$$

Consequently, the learning strategy for δ_i is

$$\delta_i(k+1) = \delta_i(k) + \rho\sum_{j=1}^{5}\frac{e^{-\eta_{ij}}}{(1+e^{-\eta_{ij}})^2} \tag{4.24}$$

This learning rule applies to a single target output. If there are n training data points available, the learning rule should be applied to all training data before the landmarks are adjusted. The learning rule can then be written as

$$\delta_i(k+1) = \delta_i(k) + \rho\sum_{p=1}^{n}\frac{\sum_{j=1}^{5}\frac{e^{-\eta_{ijp}}}{(1+e^{-\eta_{ijp}})^2}}{|n|} \tag{4.25}$$

Notice that the landmark learning strategy only applies to situations where insufficient data points are available to support the desired number of classes. If the universe of discourse is properly covered by the training data, i.e., at least five neighbors are observed in every discourse, or, using the FRCNN terminology, at least five training data points are observed in every cell of the (class/side) combinations, then this learning strategy will not be needed. In chapter 5, we shall discuss one highly data-deficient example to which the landmark learning algorithm will be applied.

A modification of equation (4.9) will prove beneficial in the situation where testing data points may coincide exactly with a training data point. The equation computing the absolute weights can then be modified to

$$w_{\mathrm{abs}_j} = \begin{cases} 10000 & \text{if } d_j \leq \epsilon \\ \frac{(d_{max}^2 - d_j^2)}{(d_{max} \cdot d_j)} & \text{otherwise} \end{cases} \tag{4.26}$$

where ϵ is a small number. The modified weight equation has the advantage that previously observed testing data will lead to an *exact* prediction. In chapter 5, we shall discuss by means of different examples what effect the modified weight equation 4.26 has on the overall quality of the prediction.

CHAPTER 5

Experimental Simulations

The main objective of the FARCNN methodology is to model an unknown system structure and its functional behavior from a set of recorded data, called the *training data set.* In order to evaluate the effectiveness of this methodology in various situations, several examples have been applied for the purpose of performing an experimental analysis on them. These examples include two continuous static nonlinear equations, a discrete dynamic equation, and a true engineering application of a hydraulic motor control system.

In all of these examples, the simulation results obtained using the FARCNN architecture are analyzed and compared to results obtained using other approaches that have been advocated in the literature. Some of the conclusions and open questions are postponed until the next chapter.

5.1 Static Continuous Nonlinear Function I

The simple function

$$f(x) = \sin(2\pi x) \tag{5.1}$$

is chosen as a first example of a static continuous nonlinear equation to be modeled using the FARCNN approach. This example has been used frequently in the literature. In order to be able to compare our results with those given in [62], we selected the same testing method that Sudkamp and Hammell used in their paper.

The training data set is comprised of k input-output pairs of $(x,\ f(x))$. Input values are generated as random numbers uniformly distributed in the range $[-1,\ 1]$. After the training data set has been generated, the output values $o(x)$ produced by the FARCNN are compared with the output $f(x)$ obtained from the target function. The testing data are evenly spaced over the input domain, i.e., testing input samples are obtained at intervals of 0.05 in the range $[-1,\ 1]$. The modeling error is defined as

$$e(x) = |f(x) - o(x)| \tag{5.2}$$

Two sets of training data were chosen, one with $k = 100$, and the other with $k = 200$. Both the input and the output space were independently partitioned once into 5 fuzzy regions and once into 15 fuzzy regions using the method advocated in chapter 3. The two different weight equations (4.9) and (4.26) introduced in chapter 4 were applied for comparison. The results obtained are listed in table 5.1, where the *FARCNN e1 error* indicates use of equation (4.9), whereas the *FARCNN e2 error* indicates using the alternate equation (4.26).

The results show that the FARCNN approach performs considerable better for the situation with more training data, yet it performs better for a smaller number of

		FARCNN e1 error		FARCNN e2 error		[62] error	
regions	examples	ave.	max.	ave.	max.	ave.	max.
5	100	0.0702	0.2624	0.0452	0.2452	0.544	1.000
	200	0.0363	0.1411	0.0245	0.1225	0.544	1.000
15	100	0.1110	0.5797	0.0454	0.5797	0.090	0.216
	200	0.0489	0.1856	0.0245	0.1830	0.071	0.179

Table 5.1: Performance comparison for example 1

classes. This has to do with the fact that the *fuzzy 5NN method* was used for fuzzy inferencing, and with 15 classes, there were not enough data points left in each class for identifying the five nearest neighbors. The problem has become *data-deficient.*

A comparison of the performance obtained with 5 regions and 15 regions respectively is presented in figure 5.1. In both cases, 200 training data were used, and the weight equation chosen was equation (4.9).

The FARCNN method performed considerably better than the alternate solution proposed in the literature [62] for 5 regions, whereas the performance was comparable for 15 regions. The much better performance shown for 5 regions is due to the use of the 5NN fuzzy inferencing method [46].

5.2 Static Continuous Nonlinear Function II

As a second example of a static continuous nonlinear function, we chose:

$$y = (1.0 + u_1^{0.5} + u_2^{-1} + u_3^{-1.5})^2 \tag{5.3}$$

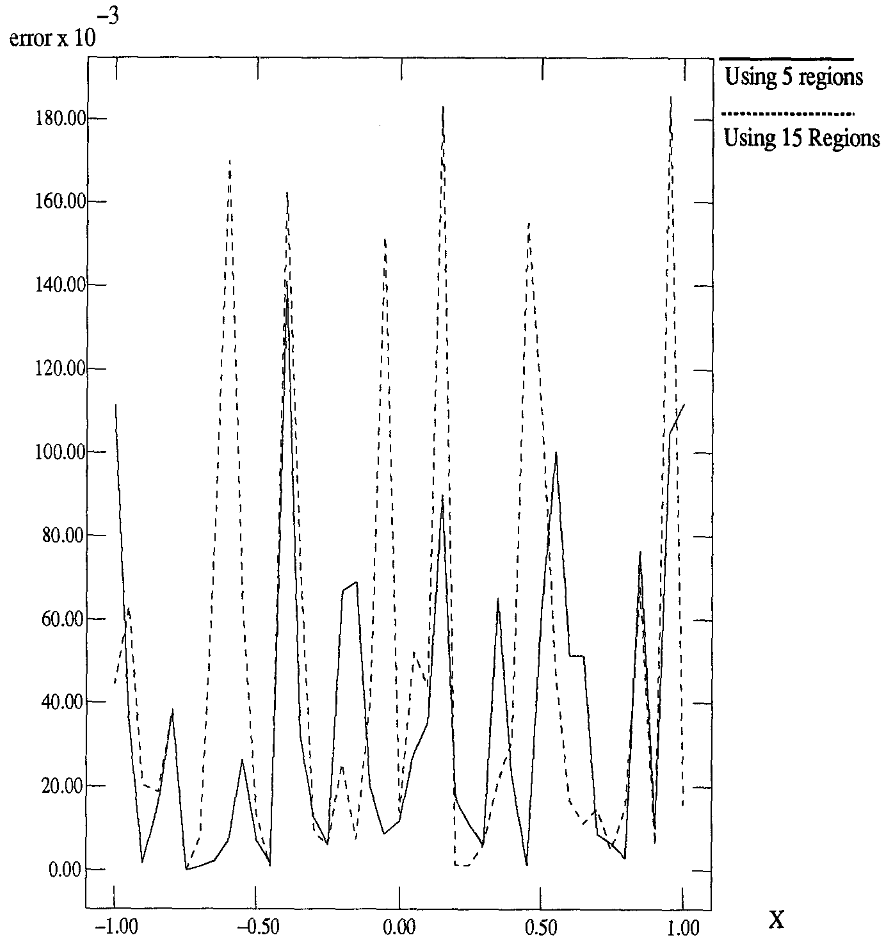

Figure 5.1: Error comparison for example 1 with 5 and 15 regions, 200 samples are used

Contrary to the previous example, this is a multi-input function, where u_1, u_2, and u_3 are three analog inputs, and y is the analog output. This function has been proposed as a benchmark example by many researchers proposing new system identification methodologies using either neural networks or fuzzy logic systems [27, 31, 63, 65].

In order to be compatible with [27], the most advanced among the previous papers making use of this benchmark problem, only 216 training data points were used, which makes this application highly data-deficient. The 216 training data are uniformly distributed over the input range $[1.0, 6.0] \times [1.0, 6.0] \times [1.0, 6.0]$. 125 testing data were selected uniformly distributed over the range $[1.5, 5.5] \times [1.5, 5.5] \times [1.5, 5.5]$. The testing data set is completely distinct from the training data set.

The same performance index, the Average Percentage Error (APE), that had been proposed in [27]:

$$APE = \frac{1}{k} \cdot \left(\sum_{\ell=1}^{k} \frac{|y(\ell) - \hat{y}(\ell)|}{|y(\ell)|}\right) \cdot 100\% \qquad (5.4)$$

is used in this example. k denotes the number of testing data pairs, $y(\ell)$ is the ℓ^{th} true output value, and $\hat{y}(\ell)$ is the ℓ^{th} output value as computed by the FARCNN.

In the FARCNN approach, two-bit fuzzy A/D converters are used as the fuzzifiers for the inputs, along with a one-bit side value for each input. Hence there are $2^9 = 512$ perceptrons in the hidden layer of the CNN, and consequently, the training data set should ideally consist of $5 \cdot 512 = 2560$ records. In the given case, there are not even 10% of these data available. Thus, many of the patterns will in fact be observed only once or twice during the training period, and some won't be observed at all.

We shall use the fuzzy 5NN method for fuzzy inferencing. However, since the example is highly data-deficient, the landmark learning approach proposed in chapter 4 should be applied.

We first selected the output landmarks from the sorting approach, as we usually do. However, due to the insufficient training data available, the FARCNN approach leads to a value of $APE = 10.75\%$ when applied to the testing data. We then applied the learning algorithm to modify the landmarks for the output variable, whereas the landmarks for the input variables were kept the same.

The learning algorithm converges quickly. After only three iterations, the APE value could be reduced by roughly 20% when using the training data as testing data. The progress of the iteration is shown in figure 5.2. Notice that the APE values shown in figure 5.2 are different from those mentioned in the text, since the numbers shown in the graph are those obtained when using the training data themselves as testing data, whereas the numbers given in the text are those obtained for the true testing data.

The refined landmark values for the output variable were then applied for use with the true testing data, leading to a value of $APE = 6.187\%$, an improvement of roughly 40% over the previously obtained APE value.

The best value reported so far in the literature is $APE = 1.066\%$ [27], a value obtained after 199.5 epochs of the training cycle of the adaptive network advocated by Jang. The results reported by other researchers using the same benchmark problem

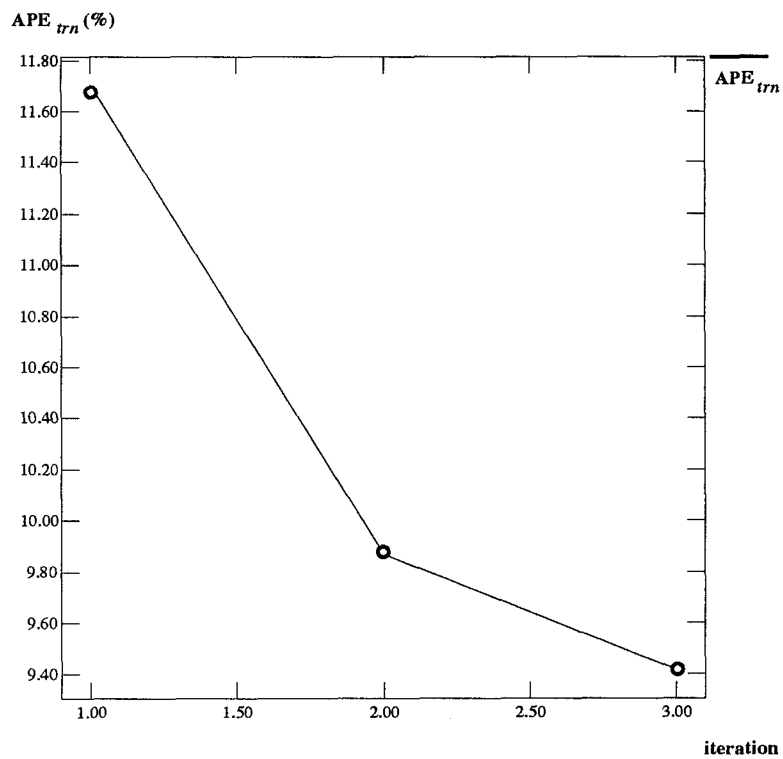

Figure 5.2: Learning error curve for example 2

Model	APE_{trn} (%)	APE_{chk} (%)
FARCNN e1	9.42	6.18
FARCNN e2	0.0006	7.03
ANFIS [27]	0.043	1.066
GMDH [31]	4.7	5.7
Linear [63]	12.7	11.1
Fuzzy 1 [63]	1.5	2.1
Fuzzy 2 [63]	0.59	3.34

Table 5.2: Performance comparison for example 2

are tabulated in table 5.2. The column APE_{trn} reports the errors found when the training data are used as testing data as well, whereas the column APE_{chk} reports the errors found when the true testing data are used. The *FARCNN e1* row refers to the use of equation (4.9) for the weight equation, whereas the *FARCNN e2* row refers to the use of equation (4.26).

Notice the incredibly small error obtained for the *FARCNN e2* method when applied to the training data used as testing data. We would expect any decent method to react in this way. Since we have seen the same inputs before, there is absolutely no need to guess what the output might be. We can know the answer. The results simply indicate that the *FARCNN e2* method exploits all the information available, whereas the other methods don't.

The table shows that the FARCNN approach doesn't fare well in comparison with most of the other methods. Yet, we are proud of the results obtained. While the other techniques took hours if not days of CPU time for training the system, our

FARCNN architecture can be set up instantaneously. This is a big advantage of the approach.

Furthermore, the example is highly data-deficient. The FARCNN approach isn't very well suited to deal with the problem of data deficiency. In order to compare the performance of the FARCNN method for differently sized training data sets, more uniformly distributed random numbers were generated to be used as additional training data using the same range as before. The testing data were kept unchanged. In this study, no landmark learning was applied, i.e., the output class landmarks were calculated directly using the sorting approach. The results of the study are shown in figure 5.3. It can be seen that the error decays exponentially with increasing number of training data. The APE value is 2.99% when 2560 training data records are used.

5.3 Discrete Dynamic Function

As a third example, let us consider the following two-input/single-output bi-linear dynamic function [40, 75]:

$$y(t) = 0.8\ y(t-1)u_1(t) + 0.5\ u_1(t-1)y(t-2) + u_2(t-4) \tag{5.5}$$

where the inputs $u_1(t)$ and $u_2(t)$ are chosen as uncorrelated random sequences uniformly distributed over the range [0.1, 0.9]. Two sets of 400 data set are generated from equation (5.5). One set is used as the training data set, whereas the other set is used for testing. A performance index $J(k)$ is used as a measure of similarity between

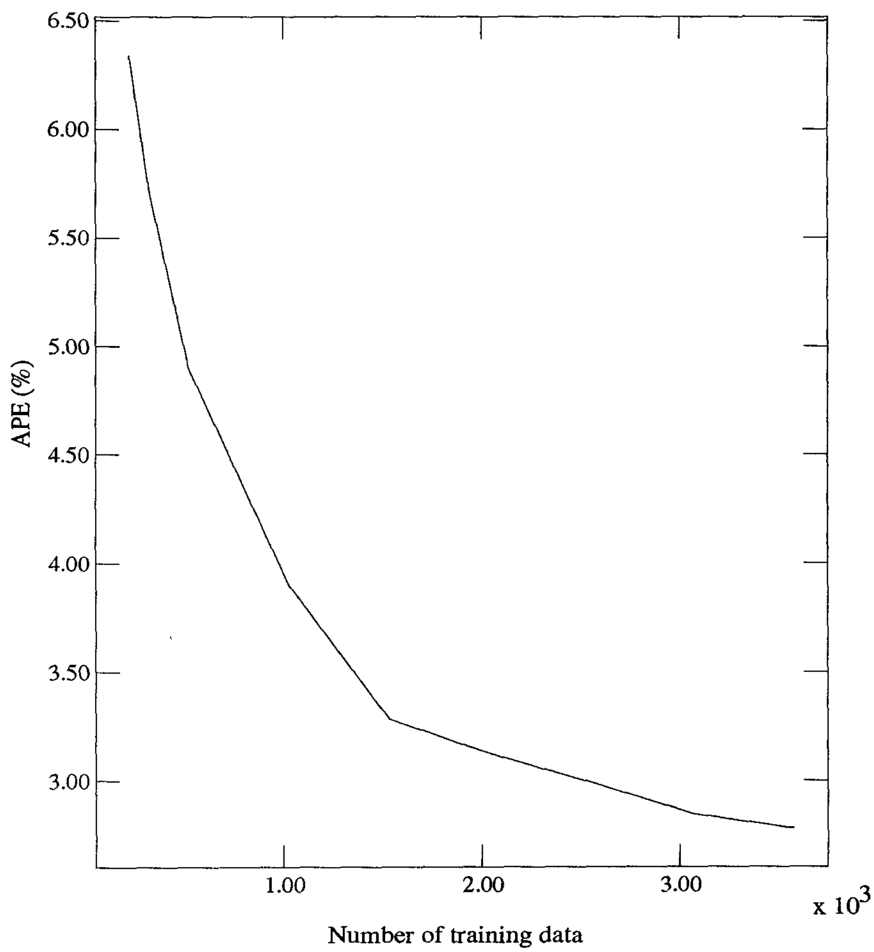

Figure 5.3: FARCNN results for example 2 with differently sized training data sets

the real and the predicted output:

$$J(k) = \frac{1}{k} \cdot \sum_{\ell=1}^{k} (y(\ell) - \hat{y}(\ell))^2 \tag{5.6}$$

Due to the recurrent characteristics of this dynamic function, an optimal mask is computed using the Fuzzy Inductive Reasoning approach introduced in chapter 3 to determine the most deterministic input-output relationship. Starting with a mask depth of three, the optimal mask returned is:

$$\begin{array}{c} {}_t\backslash^x \\ t-2 \\ t-1 \\ t \end{array} \begin{array}{c} \begin{array}{ccc} u_1 & u_2 & y \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ -2 & -3 & +1 \end{pmatrix} \end{array} \tag{5.7}$$

which suggests the following candidate for a qualitative model to evaluate y:

$$y(t) = \tilde{\mathrm{f}}(y(t-1), u_1(t), u_2(t)) \tag{5.8}$$

Using a mask candidate matrix of depth four leads to the same relationship. However, mask candidate matrices of depths five, six, and seven all suggest the qualitative model:

$$y(t) = \tilde{\mathrm{f}}(u_2(t-4), y(t-1), u_1(t)) \tag{5.9}$$

The results obtained using the model of equation (5.9) show a better performance, and this is, of course, what should be expected for this system.

The earlier references, [75] and [40], used the same data set both for training and testing. However, we decided to use a different data set for testing. The main

Model	J(400)
FARCNN e1 with relation (5.8)	0.2086
FARCNN e2 with relation (5.8)	0.1711
FARCNN e1 with relation (5.9)	0.0555
FARCNN e2 with relation (5.9)	0.0562
Fuzzy model 1 [75]	0.0230
Fuzzy model 2 [40]	0.0186

Table 5.3: Performance comparison for example 3

reason for this decision is that, when the modified weight equation (4.26) is used, we will always get a perfect match when the same data are used for training and testing, which is not of much interest. The results tabulated in table 5.3 compare our FARCNN results with the best results obtained in [75] and [40] after long iterations.

In this example, two-bit fuzzy converters were used as fuzzifiers, so the required length of the training data set should be $5 \cdot 2^9 = 2560$. Due to the complexity of the optimal mask, the function to be learned has three inputs, although the system only has two. However, the three mask inputs are correlated. This reduces the need for data somewhat. Thus, although this example is almost equally much data-deficient as the previous one, it can be noticed that the results obtained with the instantaneously synthesized FARCNN are quite comparable to those obtained after long training using the other two approaches.

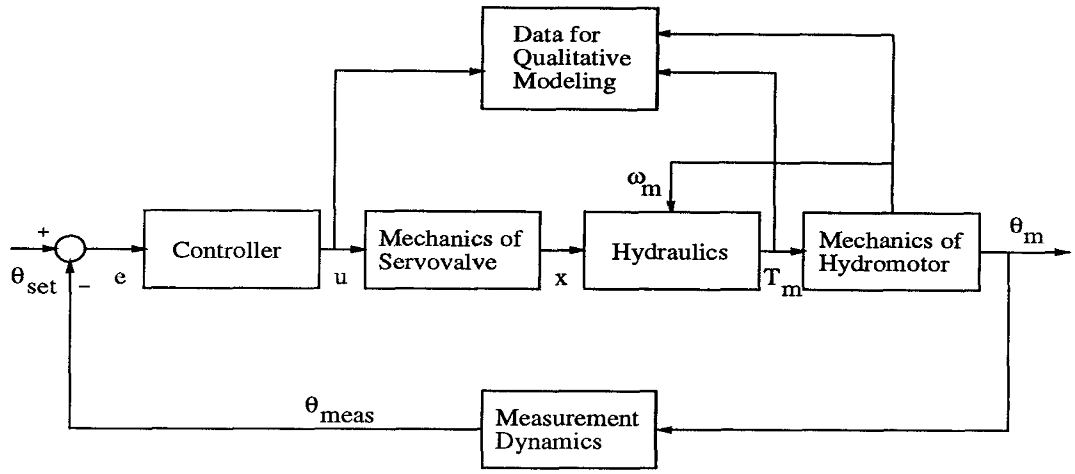

Figure 5.4: Position control diagram of a hydraulic motor

5.4 Hydraulic Motor

In this section, a complete hydraulic motor control system [8] is chosen as an example of a nonlinear dynamic system. The objective is to use a FARCNN to describe the hydraulic part of the system, and compare the performance of the FARCNN approximation with that of the quantitative model, after which it was shaped, in the closed loop control environment.

The quantitative model of this system will be used as a gauge to check the accuracy of a mixed quantitative/qualitative model in which the mechanics of the servo-valve and the hydraulics are replaced by a qualitative model (a FARCNN), whereas the controller, the mechanics of the hydraulic motor, and the measurement dynamics are kept as differential equation models. The position control diagram of the the hydraulic motor is shown in figure 5.4.

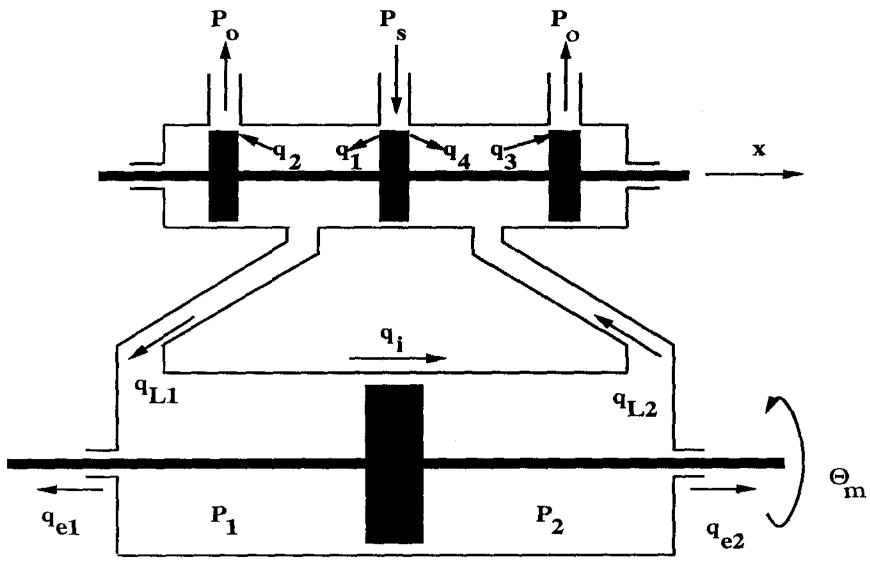

Figure 5.5: Hydraulic motor with four-way servo valve

The hydraulic motor consists of hydraulic and mechanical parts. It is controlled by a hydraulic four-way servo valve, as shown in figure 5.5.

The equations describing the servo valve are the following [8]:

$$\begin{aligned} q_1 &= k(x_0 + x)\sqrt{P_S - p_1} \\ q_2 &= k(x_0 - x)\sqrt{p_1 - P_0} \\ q_3 &= k(x_0 + x)\sqrt{p_2 - P_0} \\ q_4 &= k(x_0 - x)\sqrt{P_S - p_2} \end{aligned} \tag{5.10}$$

$$\begin{aligned} \dot{p}_1 &= c_1(q_{L1} - q_i - q_{e1} - q_{ind}) \\ \dot{p}_2 &= c_1(q_{ind} + q_i - q_{e2} - q_{L2}) \end{aligned} \tag{5.11}$$

where the parameter values are $P_S = 0.137 \times 10^8$ N m^{-2}, $P_0 = 1.0132 \times 10^5$ N m^{-2}, $x_0 = 0.05$ m, $k = 0.248 \times 10^{-6}$ kg$^{-1/2}$ m$^{5/2}$, and $c_1 = 5.857 \times 10^{13}$ kg m^{-4} sec^{-2}.

The internal leakage flow, q_i, and the external leakage flows, q_{e1} and q_{e2}, can be computed as:

$$
\begin{aligned}
q_i &= c_i \cdot p_L = c_i(p_1 - p_2) \\
q_{e1} &= c_e \cdot p_1 \\
q_{e2} &= c_e \cdot p_2
\end{aligned}
\tag{5.12}
$$

where $c_i = 0.737 \times 10^{-13}$ kg^{-1} m^4 sec, and $c_e = 0.737 \times 10^{-12}$ kg^{-1} m^4 sec.

The induced flow, q_{ind}, is proportional to the angular velocity of the hydraulic motor, ω_m:

$$q_{ind} = \psi \cdot \omega_m \tag{5.13}$$

with $\psi = 0.575 \times 10^{-5}$ m^3, and the torque produced by the hydraulic motor is proportional to the load pressure, p_L:

$$T_m = \psi \cdot p_L = \psi(p_1 - p_2) \tag{5.14}$$

The mechanical side of the motor has an inertia, J_m, of 0.08 kg m^2, and a viscous friction, ρ, of 1.5 kg m^2 sec^{-1}. The controller is a P-controller with limiter, $-1 \leq u = 1 \cdot e \leq 1$, and the measurement dynamics contain one fast electrical time constant, $\dot{\theta}_{meas} = 1000\theta_m - 1000\theta_{meas}$. Thus, the overall system is of seventh order and highly non-linear.

For the design of the FARCNN, we shall make use only of the knowledge that the torque, T_m, produced by the hydraulic motor somehow depends on the control signal, u, of the servo valve and the angular velocity, ω_m, of the hydraulic motor.

By realizing a qualitative model, we in fact replace a differential equation model by a difference equation model. This model contains implicitly a sample-and-hold circuit. Thus, the Shannon sampling theorem must be satisfied. Looking at the eigenvalues of the system, we notice that we need a sampling rate of $\Delta t = 0.0025$sec in order to guarantee the stability of the closed loop system.

When the FIR methodology is applied to find the most appropriate input-output relation between T_m, u, and ω_m, the following qualitative model is obtained:

$$T_m(t) = \tilde{\mathrm{f}}(u(t), \omega_m(t - 0.05), T_m(t - 0.05)) \tag{5.15}$$

which suggests a much larger sampling rate. Thus, both the angular velocity of the motor, ω_m, and the motor torque, T_m, need to be delayed by $20 \cdot \Delta t$.

The qualitative model operates on three analog inputs, the control input, u, at the current time, the angular motor velocity, ω_m, delayed by $20 \cdot \Delta t$, and the motor torque, T_m, also delayed by $20 \cdot \Delta t$. The FARCNN structure for this hydraulic motor is shown in figure 5.6.

The data used for constructing the qualitative model are records of the three variables u, ω_m, and T_m, sampled once every 0.0025 sec. To this end, Binary Random Noise (BRN) was applied to the quantitative model in closed loop: $\theta_{\text{set}} = BRN$. The

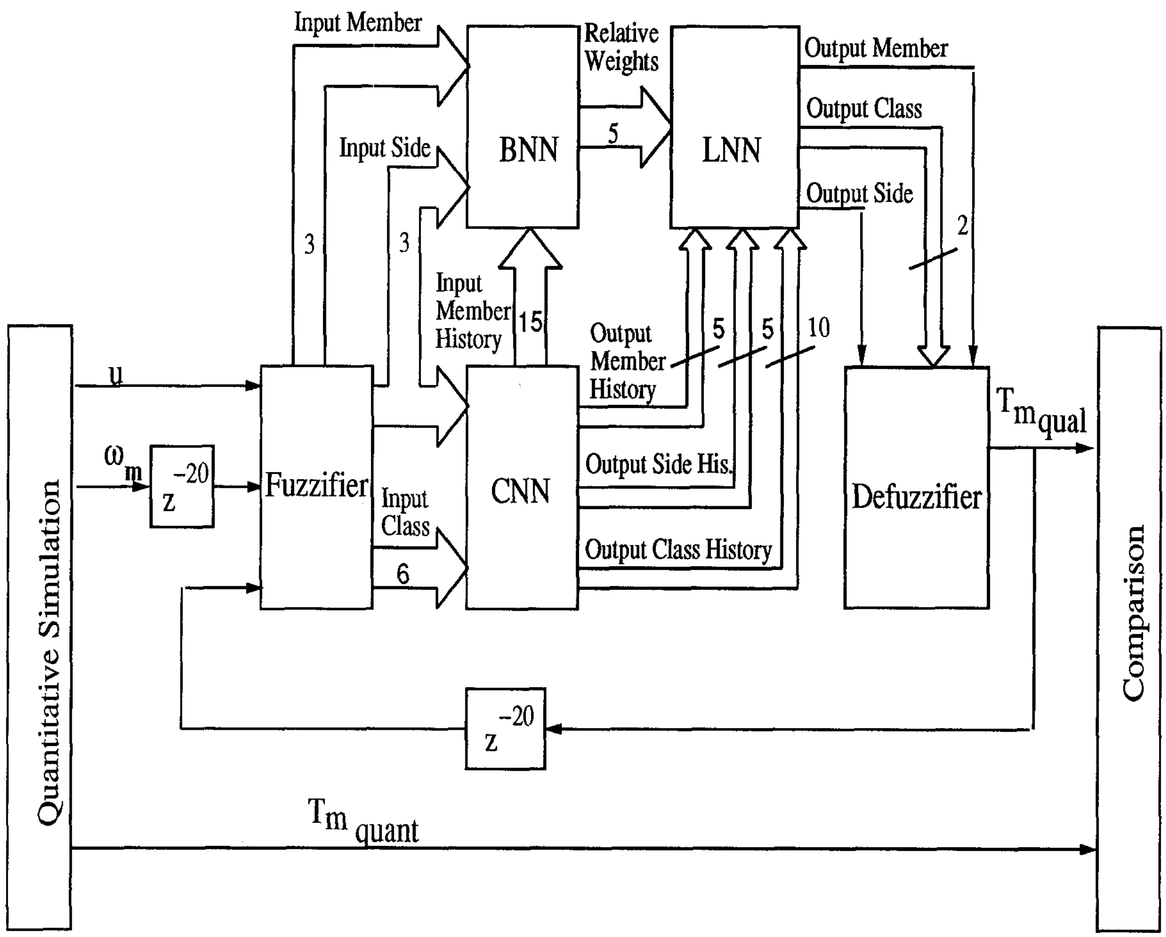

Figure 5.6: FARCNN structure for hydraulic motor

clock for the noise generator was 0.1 sec, i.e., the value of θ_{set} was randomly assigned a new value of either = 1.0 rad or -1.0 rad once every 0.1 sec.

The quantitative model was simulated across 2.5 sec of simulated time corresponding to 1001 data records. The first 2.25 sec, or 901 data records, were used as training data, and the final 100 data records were used for model validation. Figure 5.7 shows a comparison of the last 100 values of the torque, T_m, stored during the quantitative simulation (solid line) with the values, $\hat{T}_m$, that were predicted by the FARCNN when used in open loop (dashed line). In this experiment, u and ω_m were driven by the data stored during the quantitative simulation, and only T_m was fed back into the FARCNN.

During the next experiment, the control loop was closed around the FARCNN. Here, u and ω_m were generated on-line by the quantitative model, and T_m was fed back from the FARCNN to the quantitative model, thereby influencing future values of u and ω_m. Figure 5.8 shows the results of this experiment. The solid line shows the results from the purely quantitative simulation, whereas the dashed line shows the results from the mixed quantitative and qualitative simulation run.

As was to be expected, the results are slightly better for the open-loop case, but even the closed-loop results are excellent. The results are almost identical to those shown in [8]. This is not further surprising, since FARCNNs are simply a means for fast implementation of the FIR qualitative modeling methodology for real-time applications. In this example, three-bit fuzzy converters were used as fuzzifiers.

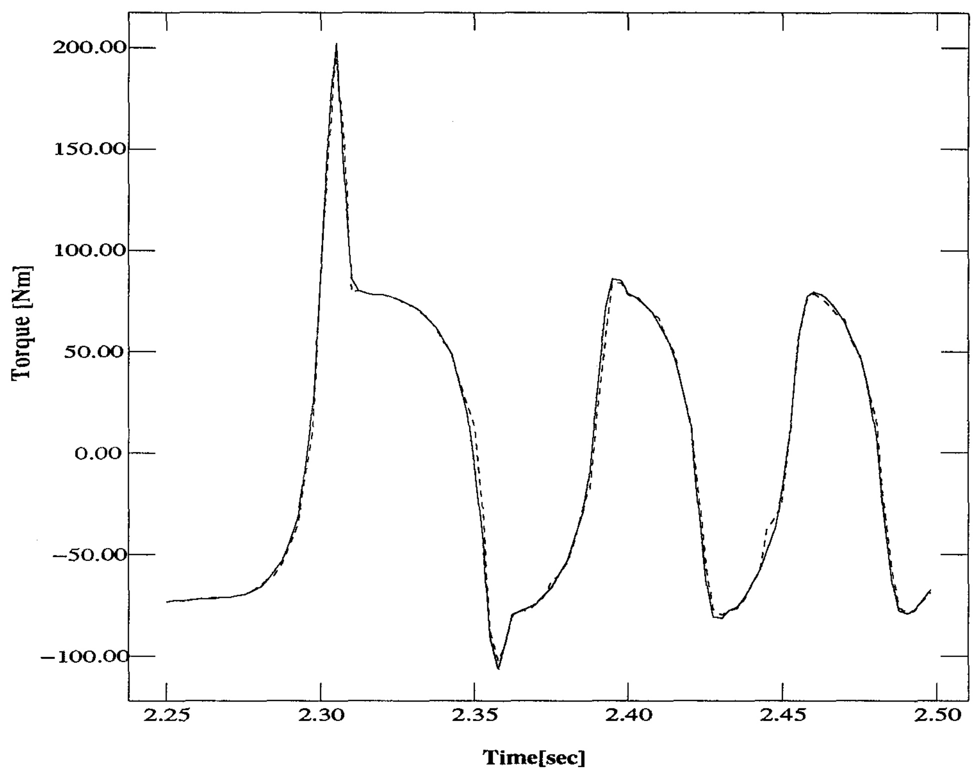

Figure 5.7: Simulation and forecast behavior of hydraulic motor in open loop

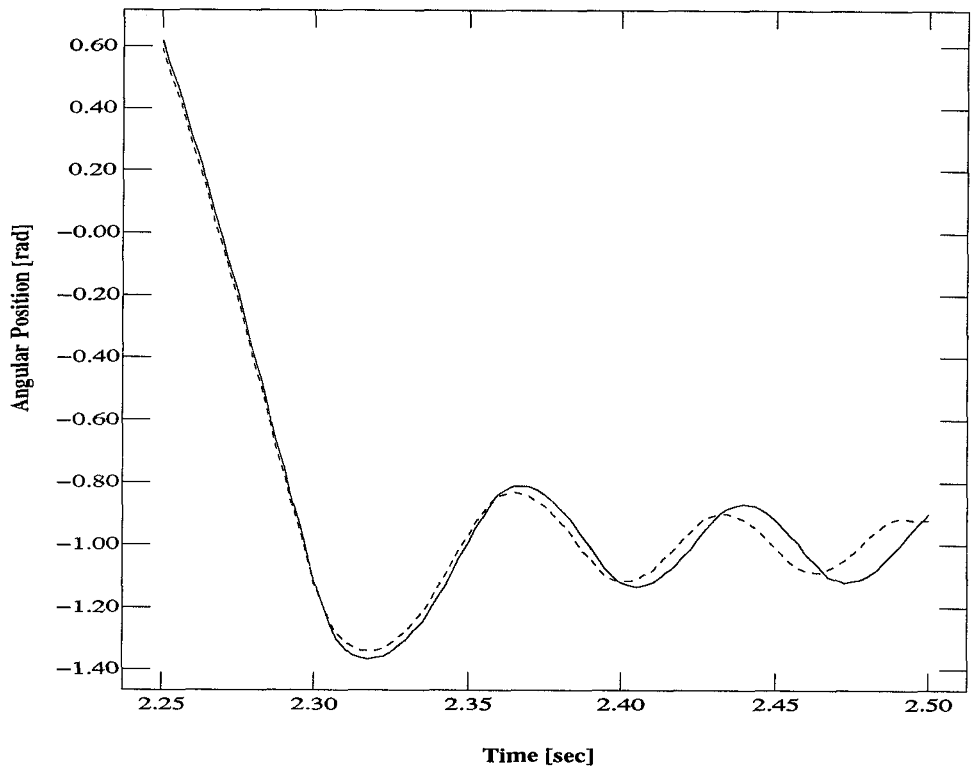

Figure 5.8: Simulation and forecast behavior of hydraulic motor in closed loop

5.5 Comparison of Different Fuzzy Reasoning Techniques

It is interesting to notice that when we combine equations (4.8), (4.9), (4.10) and (4.13), the fuzzy output from FIR will be:

$$O_{qt} = \frac{\sum_{j=1}^{5} w_{abs_j} \cdot y_{qt_j}}{\sum_{j=1}^{5} w_{abs_j}} \tag{5.16}$$

where the subscript qt is used to represent the qualitative triple. This is exactly the same value that we would obtain by using the Weighted Average Method, one of the widely used fuzzy reasoning techniques. The main difference is that we always use exactly five "rules," and that the so-called *premise value*, i.e., the *absolute weight*, w_{abs}, using FIR terminology, is calculated by using the fuzzy 5NN (or 5N) algorithm instead of the Mean of Maxima (MoM) or Center of Area (CoA) techniques. In [46], it was demonstrated that the 5NN method performs much better than either MoM or CoA in data-rich environments. If we replace the y_{qt} with fuzzy singletons, i.e., crisp values, equation (5.16) will become the same as the third type of fuzzy reasoning mentioned in [39], operating in the same way as the standard 5NN technique.

The second nonlinear function in this chapter can be used as an example to demonstrate the performance difference in fuzzy reasoning when either the 5NN, the fuzzy 5NN, the 5N, or the nearest neighbor (1NN) techniques are used. Comparative results are shown in figure 5.9 plotted across the available number of training data.

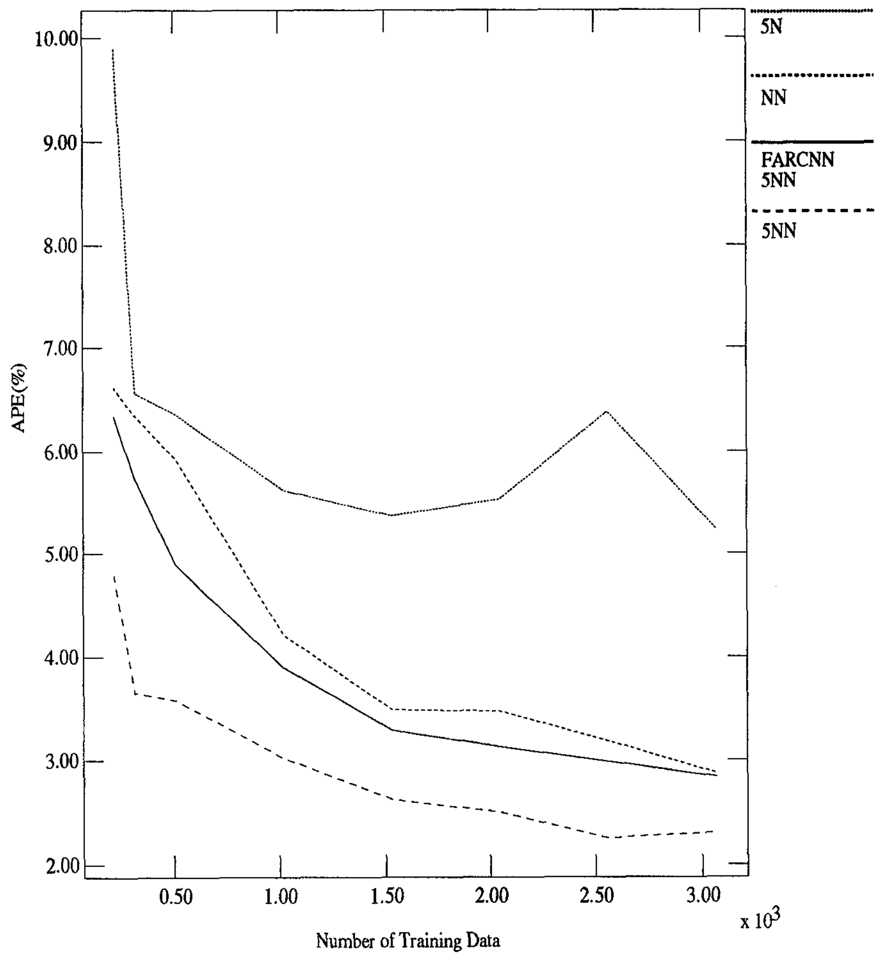

Figure 5.9: Comparative results for example 2 with different fuzzy reasoning techniques

The results show that the true 5NN technique exhibits a better performance than the fuzzy 5NN technique. This is not further surprising, since the true 5NN algorithm will always work with the truly nearest five neighbors, whereas the fuzzy 5NN technique gives a preference to neighbors in the same class, even if they are farther away than the truly nearest five neighbors. What is gained by the fuzzy 5NN algorithm is a much reduced search space, which makes the fuzzy 5NN algorithm much more economical, an important consideration especially in real-time applications.

It can also be noticed that the true 1NN technique always performs more poorly than the fuzzy 5NN algorithm. Since the true 1NN algorithm requires a search through the entire training data set, it is not an attractive algorithm. For this reason, it was never used in this dissertation, except here in order to demonstrate its poor performance.

It is interesting to notice that the fuzzy 5N technique is consistently the worst technique of all, and that it becomes even less attractive when a large amount of training data are available. For few training data, it suffers from the usual problems of data deprivation, just like all the other techniques. Its performance improves up to a point where we encounter five neighbors in every class (not in every class/side combination), and then it stagnates. The availability of more data doesn't seem to do much good.

To analyze what happens in this case, let us look at the simple linear function, $f(x) = 0.1\ x$, and assume that there are exactly five training data available in

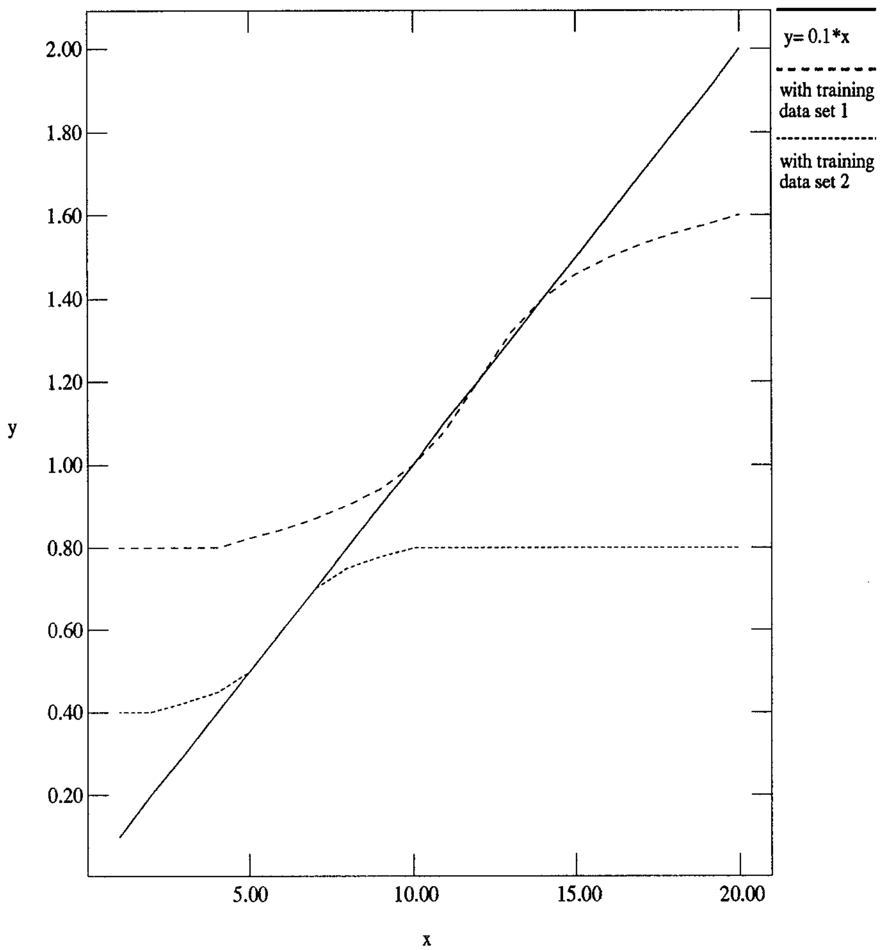

Figure 5.10: Approximation of $y = 0.1x$ by FARCNN using 5N algorithm

the range $x \in [1,\ 20]$. We shall make two tests, one with the training data being uniformly distributed in the range [1, 20], the second with the training data uniformly distributed in the range [1, 10]. The BNN presented in the last chapter and the 5N algorithm are applied. Their are twenty testing patterns evenly spaced over the range [1, 20]. The results are shown in figure 5.10.

Around the center of gravity of the five training data points, there is an area where the FARCNN approximates the correct function adequately. However, this area is fairly small. Towards the border of the region, the approximation degenerates rapidly.

This explains the difference in performance between the true 5NN and the fuzzy 5NN algorithms. The true 5NN algorithms will always look for the true five nearest neighbors, which statistically seen will usually be grouped *around* the testing data point, i.e., the center of gravity of the five nearest neighbors will always be in the vicinity of the testing point. However, this assumption does not hold true for the fuzzy 5NN. If a testing point is located in the vicinity of the border between two classes, the five so-called nearest neighbors are no longer grouped symmetrically around the testing point, and therefore, the approximation error will be larger.

This also explains the stagnation of the 5N algorithm. If more data points are provided to the fuzzy 5NN algorithm, it will look for the five nearest neighbors within the class/side combination. If there are many candidates available, it is more likely that these points will be grouped around the testing point. This is true in a larger area within the class/side combination. Consequently, the inaccurate areas around the borders between classes shrink in size with an increasing number of data points. However, this is not the case when the 5N algorithm is used. The additional data points are simply ignored.

The reason why the 5N algorithm is still attractive is purely economical. The 5N algorithm does not require any sorting at all, and therefore, it is very attractive for use in real-time applications.

If there are about five training data available for each class, the fuzzy 5N algorithm and the fuzzy 5NN algorithm perform similarly, because both will use the same five training data points.

This argument can be turned around. If we wish to make the fuzzy 5N algorithm perform about as well as the fuzzy 5NN algorithm irrespective of the amount of training data available, we need to make sure that there are always about five training data points in every class, i.e., the number of classes must be adjusted to reflect the amount of available training data.

Unfortunately, with the number of classes growing, the size of the Kohonen layer inside the CNN will grow rapidly, and the CNN will soon become unmanageably large. A possible solution to this problem will be outlined in chapter 6.

CHAPTER 6

Conclusion and Future Research

In this dissertation, a new Artificial Neural Network (ANN) architecture called *Fuzzy Adaptive Recurrent Counterpropagation Neural Network* (FARCNN) has been presented. FARCNNs can be directly synthesized from a set of training data, making system behavioral learning extremely fast. FARCNNs can be applied directly and effectively to model both static and dynamic system behavior based on observed input/output behavioral patterns alone, without need of knowing anything about the internal mechanisms of how the system to be modeled operates, i.e., without knowing anything about the internal structure of the system under study.

The FARCNN architecture is derived from the methodology of Fuzzy Inductive Reasoning (FIR), which is an advanced behavioral modeling technique based on the human ability to find analogies among similar processes, to predict coarse system dynamic behavior. FARCNNs also make the simulation of mixed quantitative and qualitative models much more feasible, even under real-time constraints. Previous implementations of mixed quantitative and qualitative simulation models using FIR [8] were executing so slowly that they were of not much practical value.

The concept of Fuzzy Logic has been applied to FIR for its *Fuzzy recoding* stage to convert crisp values to fuzzy values. A newly proposed fuzzy A/D converter was used for this purpose. Instead of using the regular fuzzy rules as the knowledge base, a Generalized Counterpropagation Neural Network (GCNN) is applied for efficient implementation of Finite State Machines (FSM). The GCNN is used for storing knowledge obtained from training data. A recurrent counterpropagation neural network (RCNN) has also been presented. This enhancement of the methodology allows to handle fully dynamic system by adding memory to the network. Matched with the form of the knowledge base used in the FARCNN architecture, the five nearest neighbors (5NN) method has been adapted for use as the fuzzy inferencing stage. It was also shown that a standard Backpropagation Neural Network (BNN) can be applied to implement the 5NN algorithm in a highly efficient fashion. This BNN can be applied to arbitrary applications without need for retraining. As the final stage of the overall architecture, a fuzzy D/A converter has been applied to convert fuzzy values back into crisp values.

In simulation experiments, we have shown that FARCNNs can be applied directly and easily to different types of systems, including static continuous nonlinear systems, discrete sequential systems, and as parts of large dynamic continuous nonlinear control systems, embedding the FARCNN into much larger industry-sized quantitative models, even permitting a feedback structure to be placed around the FARCNN. Comparisons were made to analyze the performance of the FARCNN architecture relative to that of other techniques previously advocated in the open literature to demonstrate the strengths and weaknesses of the new methodology.

Since the FARCNN architecture is simply a real-time implementation scheme for the FIR methodology, it inherits most of the advantages of the FIR approach to qualitative modeling. In particular, in inherits the robustness properties of fuzzy logic, i.e., the ability to perform adequately well under conditions of incomplete knowledge. It also inherits, through the use of the 5NN algorithm, the superb fuzzy inferencing properties of the FIR methodology, which, as has been shown in [46], are far superior to those exhibited by other fuzzy inferencing techniques that are commonly employed by fuzzy logic systems, such as the Mean of Maxima (MoM) or the Center of Area (CoA) techniques, at least when used in a data-rich environment.

In [9], it was also shown that the 5 Neighbors (5N) technique, instead of the 5 Nearest Neighbors (5NN) technique, can also be employed in the FARCNN architecture. The 5N algorithm does not perform as well as the 5NN algorithm with respect to the accuracy of the prediction, but it is much more economical, since no sorting is necessary in this case.

More importantly, by a similar approach as the one used in [72], it can be shown that FARCNNs can approximate arbitrary real continuous functions. Similar to [32], it can also be shown that linear combinations of the fuzzy rules employed inside the FARCNN can approximate continuous functions to arbitrary accuracy.

Quite a bit of effort was spent in this dissertation on designing techniques that would keep the size of the CNN inside the FARCNN reasonably small. In combination with the optimal mask analysis of the FIR methodology, FARCNNs have been shown to be able to approximate the behavior of arbitrary dynamic systems without growing unacceptably large.

6.1 Future Research

Although the usefulness of the FIR methodology and its FARCNN implementation has been demonstrated by a good number of examples already, the methodology still suffers from a number of shortcomings. In particular, it is characterized by a high degree of heuristicism. Also, as has been shown in chapter 5, FARCNNs are quite vulnerable to data deprivation. A few of the unsolved problems shall be mentioned in this section to stimulate future students to continue with this line of research.

1. As we have shown in chapter 5, FARCNNs and other implementations of the FIR methodology require a large amount of rich training data in order to approximate functions with a high degree of accuracy. It was shown that the approximation error of the FARCNN decreases exponentially with an increasing number of training data. We can say that the FARCNN performs "exponentially well" with increasing training data. It has been shown in [62] that, by applying the fuzzy region completion technique, it is possible to obtain a good approximation accuracy with a much smaller number of training data, i.e., that synthetic training data generated using the fuzzy region completion technique can, under some circumstances, replace real training data with very good success. However, [62] shows these results only for static continuous functions. It remains to be analyzed how the use of synthetic training data generated

by the fuzzy region completion technique affects the forecasting power in qualitative dynamic system simulation.

2. The FIR methodology, and with it the FARCNN architecture, is based on the 5NN algorithm for fuzzy inferencing. We have shown in chapter 5 that this algorithm performs much better than other known algorithms for fuzzy inferencing at least in a data-rich environment. In [11], it was shown that the error bound for the 1NN algorithm is twice the Bayes probability, and [4] proves that the relation function for k nearest neighbors is as follows:

$$\begin{aligned} k &\sim \frac{n}{\epsilon^2} \ln(\epsilon^{-1}) \\ M &= o(\frac{n}{\epsilon^2} \ln(\epsilon^{-1})) \end{aligned} \tag{6.1}$$

which means that the k nearest neighbors yield an ϵ accurate classifier when M training data records are used. Following the reasoning outlined in these two references, it should be perfectly feasible to provide a formal analysis of the approximation accuracy of the 5NN algorithm when used in a FARCNN or another FIR implementation scheme linking the approximation accuracy to the size of the training data set.

3. In chapter 4, a learning mechanism was outlined that would enable FARCNNs to adjust themselves to a slowly changing system. Although off-line learning schemes (such as the landmark learning strategy) have indeed been implemented, they are comparatively harmless because they are not time-critical. However, to this date, no on-line adaptation mechanisms have actually been implemented within our FARCNN

architecture, and this is for a good reason. Whereas on-line learning can be easily implemented in a simulation environment, such as the one used in this research effort, it is not so clear how such a mechanism would have to be implemented on a chip. What does it mean "to simply add another neuron to the Kohonen layer"? How is a round-robin strategy implemented in hardware? Until now, although all the results shown in this thesis were obtained by means of simulation only, we concentrated our attention on features that can easily be implemented in hardware. For example, it would be a very easy task to design a chip implementing the BNN proposed in this thesis. More research is needed before the same can be said for the on-line learning feature. The problem is that, whereas most on-line learning algorithms proposed in the literature for other types of neural networks or fuzzy systems are parametric, ours is non-parametric.

4. It was shown in chapter 5 that the 5N algorithm only operates efficiently in the case where the number of legal states (combinations of classes of all variables) equals one fifth of the number of training data records. In the case where a high degree of approximation accuracy is required, we need many training data records, and therefore, many classes. This will soon make the CNN inside the FARCNN unmanageably large. Let us assume we have 30,000 training data records. This means that our system should have 6000 legal states or 12,000 combinations of classes and side values. Thus, the Kohonen layer will require 12,000 neurons. This may be quite unacceptable.

We could use the 5NN algorithm with only 1000 legal states, in which case there would be 30 observations in each state. We would then have to find the 5 Nearest Neighbors out of the 30. Alternatively, we can implement a 30N algorithm, also reducing the number of neurons in the Kohonen layer to 2000, but now, the Grossberg layer would have to generate 30 M_{ij} values rather than only 5 of them. Thus, the Grossberg layer would require 25 additional neurons per input variable.

We could then feed the membership values of the testing data record and the old membership values generated by the Grossberg layer of the CNN into a *Sorting Neural Network*, as outlined in chapter 2 that would find the 5 Nearest Neighbors (5NN) out of the 30 Neighbors (30N). These could then be fed to the BNN as in the past. The enhanced FARCNN architecture is shown in figure 6.1.

In this way, the job of searching the training data for appropriate information would be split between the CNN and the SNN, thereby keeping each of the two networks reasonably compact. However, it remains to be researched, how this should be done in practice .

5. One of the most interesting properties of the FIR methodology has not been preserved in the FARCNN architecture. The FIR methodology not only provides a forecast, but with it, also provides an estimate of the quality of that forecast. We feel strongly that this is an essential function that ought to be provided by *any* qualitative simulation tool. More research is needed to come up with a scheme that would enable us to duplicate this FIR functionality also within the FARCNN architecture.

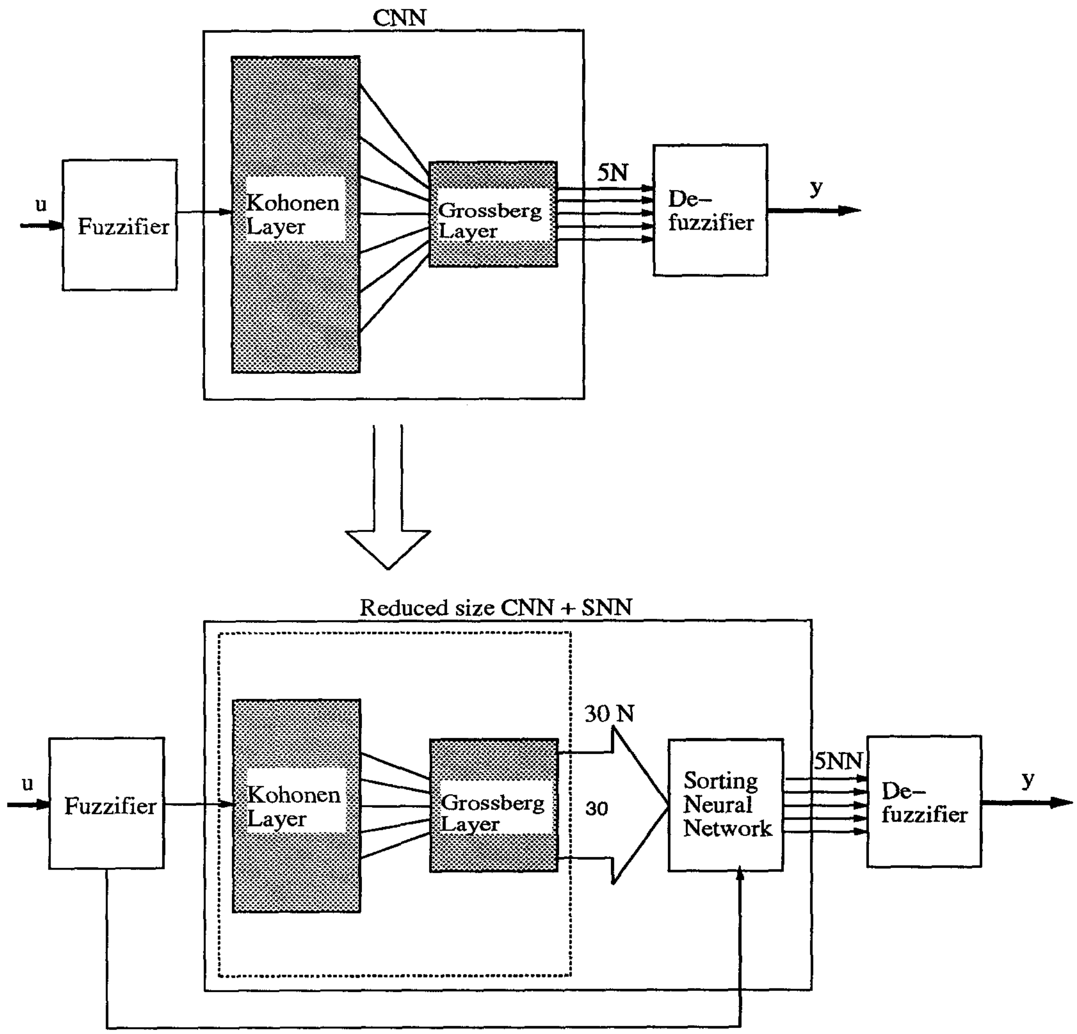

Figure 6.1: Enhanced FARCNN architecture to cascade a reduced size CNN and SNN

6. Right now, the FARCNN architecture stores the knowledge of the training data in a generalized counterpropagation neural network, instead of a set of fuzzy rules. At least conceptually, each training data is treated as one fuzzy rule. However, these fuzzy rules contain a lot of redundancy. How can a condensed fuzzy rule base be generated from the CNN for use by other reasoners or for use in a modified neural network that is able to store the available training data information in a more compact fashion? This is another highly interesting research topic that ought to be looked at. [71] provides some ideas that could be explored in the context of this research.

7. No serious discussion of fuzzy system technology should end without at least mentioning the truly hard problems. How can we prove that a FARCNN used within a feedback loop will keep this loop BIBO stable under all excitations? Several researchers have addressed related questions in the past without too much of a success. Maybe the reason for this failure is that the question has been posed incorrectly. Maybe a better approach would be to ponder about the possibility of designing a stabilizing circuit that can be placed within any feedback loop that would force the loop to remain stable irrespective of what else is in the loop, thus also in the case of dealing with one or even several FARCNNs. It may be easier to answer this question than to answer to original one, and, of course, the end effect would be the same. However, this is a very difficult problem that will presumably haunt us for years to come.

REFERENCES

[1] P. Antsaklis, "Defining Intelligent Control," *IEEE Control Systems*, vol. 14, no. 3, pp. 4-66, 1994.

[2] K. J. Astrom and B. Wittenmark, *Computer Controller Systems: theory and design,* Prentice-Hall, Englewood Cliffs, N. J., 1984.

[3] G. Ausiello, M. Lucertini, and P. Serafini, *Algorithm design for computer system design,* Springer-Verlag, New York, 1984.

[4] E. B. Baum, "When Are k-Nearest Neighboo and Back Porpagation Accurate for Feasible Sized Sets of Examples?," *1990 EURASIP workshop on Neural Networks,* pp. 2-25, Feb. 1990.

[5] N. K. Bose and A. K. Garga, "Neural Network Design Using Voronoi Diagrams," *IEEE Trans. Neural Networks* vol. 4, no. 5, pp. 778-787.

[6] F. E. Cellier and D. W. Yandell, "SAPS-II: A New Implementation of the Systems Approach Problem Solver," Int. J. General Systems, vol. 13, no. 4, pp. 307-322, 1987.

[7] F. E. Cellier, *Continuous System Modeling,* Springer-Verlag, New York, 1991.

[8] F. E. Cellier, A. Nebot, F. Mugica, and A. de Albornoz, "Combined Qualitative/Quantitative Simulation Models of Continuous–Time Processes Using Fuzzy Inductive Reasoning Techniques," *International Journal of General Systems,* accepted for publication, 1994.

[9] F. E. Cellier and Y.-D. Pan, "Fuzzy Adaptive Recurrent Counterpropagation Neural Networks: A Tool for Efficient Implementation of Qualitative Models of Dynamic Processes," *Journal of Systems Engineering,* accepted for publication, 1994.

[10] R. C. Conant, "Extended Dependency Analysis of Large Systems," Int. J. Gen. Syst., Vol.14, pp. 97-141, 1988.

[11] T. M. Cover and P. E. Hart, "Nearest Neighbor Pattern Classification," *IEEE Trans. Information Theory,* vol. 13, no. 1, pp. 21-27, 1967.

[12] S. Daley and K. F. Gill, "Comparison of a Fuzzy Logic Controller with a P+D Control Law," J. Dynamical Syst., Meas., Control, vol. 111, pp. 128-137, 1989.

[13] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[14] D. P. Filev and R. R. Yager, "A Generalized Defuzzification Method Via BAD Distributions," *International J. of Intelligent Systems*, vol. 6, pp. 687–697, 1991.

[15] J. A. Freeman and D. M. Skapura, "Neural Networks: Algorithms, Applications, and Programming Techniques," Addison-Wesley, New York, 1991.

[16] K. Funahashi, "On the Approximate Realization of Continuous Mapping by Neural Networks," *Neural Networks,* vol. 2, no. 3, pp. 183 - 192, 1989.

[17] M. M. Gupta, "Fuzzy Computing: theory, hardware, and applications," North-Holland, New York, 1988.

[18] B. P. Graham and R. B. Newell, "Fuzzy Adaptive Control of a First-Order Process," *Fuzzy Sets Syst.,* voll. 31, pp. 47-65, 1989.

[19] R. Hecht-Nielsen, "Counterpropagation Networks," *Applied Optics,* vol. 26, no. 23, pp. 4949-4984, 1987.

[20] T. Heckenthaler and S. Engell, "Approximately Time-Optimal Fuzzy Control of a Two-Tank System," *IEEE Control Syst.,* vol. 14, no. 3, pp. 24-30, 1994.

[21] T. Hessburg and M. Tomizuka, "Fuzzy Logica Control for Lateral Vehicle Guidance," *IEEE Control Systems,* vol. 14, no. 4, pp. 55-63, 1994.

[22] Y. Hisanaga, M. Yamashita, and T. Ae, "Set Partition of Real Numbers by Hopfield Neural Network," *Systems and Computers in Japan,* vol. 22, no. 10, pp. 88-94, 1991.

[23] G. E. Hinton and T. J. Sejnowski, "Learning and Relearning in Boltzmann Machines," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* MIT Press, Cambridge, MA, pp. 282-317, 1986.

[24] H. J. Holt and S. Semnani, "Convergence of Back-Propagation in Neural Networks using a Log-Likelihood Cost Function," *Electronics Letters*, vol. 26, no. 23, pp. 1964-1965, 1990.

[25] J. J. Hopfield, "Neural Networks and Physical Systems with emergent Collective Computational Abilities," *Proc. Nat. Acad. Sci., U. S.* vol. 79, pp. 2554-2558, 1982.

[26] L.-J. Huang and M. Tomizuka, "A Self-Paced Fuzzy Tracking Controller for Two Dimensional Montion Control," *IEEE Trans. Syst., Man, Cyber.,* vol. 20, no. 5, pp. 1115-1124, 1990.

[27] J.-S. Jang, "ANFIS: Adaptive-Network-Based Fuzzy Inference System," *IEEE Trans. Syst., Man, Cybern.,* vol. 23, no. 3, pp. 665-683, 1993.

[28] A. Kaufmann and M. M. Gupta, *Introduction to Fuzzy Arithmetic: Theory and Application,* Van Nostrand Reinhold, New York, 1991.

[29] G. J. Klir, *Architecture of Systems Problem Solving,* Plenum Press, New York, 1985.

[30] G. J. Klir, "Inductive Systems Modelling: An Overview." In M. S. Elzas, T. I. Ören, and B. P. Zeigler (Eds.), *Modelling and Simulation Methodology: Knowledge Systems' Paradigms*, Elsevier Science Publishers B.V. (North–Holland), Amsterdam, The Netherlands, 1989.

[31] T. Kondo, "Revised GMDH Algorithm Estimating Degree of the Complete Polynomial," *Trans. Soc. Instrument and Contr. engineers,* vol. 22, no. 9, pp. 928-934, 1986.

[32] B. Kosko, *Neural Networks and Fuzzy Systems: A dynamical systems approach to machine intelligence.* Englweood Cliffs, NJ: Prentice Hall, 1992.

[33] R. Kothari, P. Klinkhachorn, and R. S. Nutter, "An Accelerated Back Propagation Training Algorithm," *1991 IEEE Int. Joint Conf. Neural Networks,* pp. 165-170, Nov. 1991.

[34] L. G. Kraft and D. P. Campagna, "A Comparison between CMAC Neural Network Control and Two Traditional Adaptive Control Systems," *IEEE controls Sys.,* pp. 36 - 43, April, 1990.

[35] B. Kuipers, "Qualitative Simulation," *Artificial Intelligence,* vo. 29, pp. 289-338, 1986.

[36] A. Law and D. Kelton, *Simulation Modeling and Analysis,* 2nd Ed. McGraw-Hill, New York, 1990.

[37] J. Layne and K. Passino, "Fuzzy Model Reference Learning Control for Cargo Ship Steering," *IEEE Control Syst.,* vol. 13, no. 6, 1993.

[38] C. C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller - part I," *IEEE Trans. Syst., Man, Cybern.,* vol. 20, no. 2, pp. 404-418, 1990.

[39] C. C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller - part II," *IEEE Trans. Syst., Man, Cybern.,* vol. 20, no. 2, pp. 419-435, 1990.

[40] Y.-C. Lee, C. Hwang, and Y.-P. Shih, "A Combined Approach to Fuzzy Model Identification," *IEEE Trans. Syst., Man, Cybern.,* vol. 24, no. 5, 1994.

[41] D. Li and F. E. Cellier, "Fuzzy Measures in Inductive Reasoning," Proc. 1990 Winter Simulation Conference, New Orleans, La., pp. 527-538, 1990.

[42] G. Li, H. Alnuweiri, Y. Wu, and H. Li, "Acceleration of Back Propagation through Initial Weight Pre-training with Delta Rule," *1993 IEEE Int. Conf. Neural Networks,* San Francisco, CA, pp 580 - 585, 1993.

[43] W. M. McCulloch and W Pitts, "A logical Calculus of the Ideas Immanent in Nervous Acticity," *Bulletin of Mathematical Biophysics,* vol. 5, pp. 115 - 133, 1943.

[44] M. McInerney and A. P. Dhawan, "Use of Genetic Algorithm with Back Propagation in Training of Feed-Forward Neural Networks," *1993 IEEE Int. Conf. Neural Networks,* pp. 203-208, 1993.

[45] A. J. Morgan, *The Qualitative Behaviour of Dynamic Physical Systems,* Ph.D dissertation, Univ. of Cambridge, Nov., 1988

[46] F. Mugica and F. Cellier, "A New Fuzzy Inferencing Method for Inductive Reasoning," *Proc. Int. Symp. Artificial Interlligence,* Monterrey, Mexico, pp. 372-379, Sept. 20-24, 1993.

[47] F. Mugica and F. Cellier, "Automated Synthesis of A Fuzzy Controller for Cargo Ship Steering by Means of Qualitative Simulation," *Proc. Modelling and Simulation,* Barcelona, Spain, pp. 523-528, June 1-3, 1994.

[48] K. Murase, Y. Matsunaga, and Y. Nakade, "A Back-Propagation Algorithm which automatically Determines the Number of Association Units," *1991 IEEE Int. Joint Conf. Neural Networks,* pp. 783-788, Nov, 1991.

[49] K. S. Narendra and K. Parthasarathy, "Identificcation and Control of Dynamical Systems Using Neural Networks," *IEEE trans. Neural Networks,* vol. 1, no. 1, pp. 4 - 26, 1990.

[50] Y. H. Pao, M. Klaasen, and V. Chen, "Characteristics of the Functional Link Net: A Higher Order Delta Rule Net," 1988 IEEE Int. Conf. Neural Networks, pp. 507-514, 1988.

[51] W. Pedrycz, "Fuzzy Control and Fuzzy Systems," Wiley, New York, 1989.

[52] S. M. Phillips and C. Müller-Dott, "Two-Stage Neural Network Architecture for Feedback Control of Dynamic Systems," *Analog Integrated Circuits and Signal Processing,* vol. 2, pp. 353-365, 1992.

[53] G. Raju and J. Zhou, "Adaptive Hierarchical Fuzzy Controller," *IEEE Trans. Syst., Man, Cybern.,* vol. 23, no. 4, pp. 973-980, 1993.

[54] M. R. Ramirez and D. Arghya, "Faster Learning Algorithm for Back-Propagation Neural Networks in NDE Applications," *Artificial Intelligence and Civil Engr. second Int. Conf.,* Oxford, England, pp. 275-283, Sep. 1991.

[55] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Congnitions,* vol. 1, pp. 318-362, MIT Press, Cambridge, MA.

[56] S. Shao, "Fuzzy Self-Organizing Controller and its Application for Dynamic Processes," *Fuzzy Sets Syst.*, vol. 20, pp. 151- 164, 1988.

[57] P. Shi and R. Ward, "OSNet: A Neural Network Implementaiton of Order Statistic Filters," *IEEE Trans. Nueral Networks,* vol. 4, no. 2, pp. 234-240, 1993.

[58] R. Shoureshi and K. Rahmani, "Derivation and Application of an Expert Fuzzy Optimal Control System," *J. Fuzzy Sets and Systems,* no. 49. pp. , 1992.

[59] R. Shoureshi, "Intelligent Control Systems: Are They for Real," *Journal of Dynamic Systems, Meas. and Cont., Trans. ASME,* vol. 115, pp. 392-401, 1993.

[60] S. G. Smyth, "Design Multilayer Perceptrons from Nearest-Neighgors Systems," *IEEE Trans. Neural Networks,* Vol. 3, No. 2, pp. 329-333, 1992.

[61] M. Stinchcombe and H. White, "Universal Approximation using Feedward Networks with non-sigmoid Hidden Layer Activation Functions," *Proc. Conf. Neural Networks,* Washington, DC, vol. 1, pp. 613 - 618. June 1989.

[62] T. Sudkamp and R. Hammell, "Interpolation, Completion and Learning Fuzzy Rules," *IEEE Trans. Syst., Man, Cybern.,* vol. 24, no. 2, pp. 332-342, 1994.

[63] M. Sugeno and G. T. Kang, "Structure Identification of Fuzzy Model," *Fuzzy Sets and Systems,* vol. 28, no. 1, pp. 15-33, 1988.

[64] T. Takagi and M. Sugeno, "Fuzzy Identification of Systems and its Applications to Modelling and Control," *IEEE Trans. Syst., Man, Cybern.,* vol. 15, pp. 116-132, 1985.

[65] H. Takagi and I. Hayshi, "NN-Driven Fuzzy Reasoning," *Int. J. Approx. Reasoning,* vol. 5, no. 3, pp. 191-212, 1991.

[66] H. Takahashi, "Automatic Speed Control Device using Self-Tuning Fuzzy Logic," *Proc. 1988 IEEE Workshop on Automative Application of Electronics,* pp. 65-71, Dearborn, MI, Oct., 1988.

[67] R. Tanscheit and E. Scharf, "Experiments with the use of Rule-based Self-Organising Controller for Robotics Applications," *Fuzzy Sets Syst.,* vol. 26, pp. 195-214, 1988.

[68] T. Tollenaere, "SuperSAB Fast Adaptive back Propagation with Good Scaling Properties," *Neural Networks,* vol. 3, no. 5, pp. 561-573, 1990.

[69] R. M. Tong, "Synthesis of Fuzzy Models for Industrial Process," *Int. J. Gen. Syst.,* vil. 4, pp. 143-162, 1978.

[70] D. Wang and J. Thompson, "An Adaptive Data Sorter based on Probabilistic Neural Networks," *1991 IEEE Int. Joint Conf. on Neural Networks IJCNN'91,* Singapore, Singaproe, pp. 1296-1302, Nov. 1991.

[71] L. Wang and J. Mendel, "Generating Fuzzy Rules by Learning from Examples," *Proc. 1991 IEEE Int. Symp. Intell. Control,* pp. 263-268, Arlington, VA, Aug., 1991.

[72] L. Wang and J. Mendel, "Fuzzy Basis Functions, Universal Approximation, and Orthogonal Least-Squares Learning," *IEEE Trans. Neural Networks,* vol. 3, no. 5, pp. 807-814, 1992.

[73] A. S. Weigend and N. A. Gershenfeld, *Time Series Prediction: Forecasting the Future and Understand the Past,* Addison-Wesley, Mass., 1994.

[74] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard, Cambridge, MA, August, 1974.

[75] C. W. Xu and Y. Z. Lu, "Fuzzy Model Identification and Self-Learning for Dynamic Systems," *Trans. Syst., Man, Cybern.,* vol. 17, no. 4, pp. 683-189, 1987.

[76] R. R. Yager, D. P. Filev, and T. Sadeghi, "Analysis of Flexible Structured Fuzzy Logic Controllers," *IEEE Syst., Man, Cybern.,* vol. 24, no. 7, pp. 1035-1043, 1994.

[77] R. R. Yager and D. P. Filev, *Essentials of Fuzzy Modeling and Control.* John Wiley & Sons, New York, 1994.

[78] M. Yamada, T. Nakagawa, and H. Kitagawa, "A Super Parallel Sorter Using Binary Neural Network with AND-OR Synaptic Connections," *Analog Intergrated circuits and Signal Processing,* vol. 2, no. 4, pp. 127-131, 1992.

[79] T. Yamakawa, "A Fuzzy Inference Engine in Nonlinear Analog Mode and Its Application to a Fuzzy Logic Control," *IEEE Trans. Neural Networks,* vol. 4, no. 3, pp. 496-522, 1993.

[80] H. Yan,"A Neural Network for Improving the Performance of Nearest Neighbor Classifiers," *Neural and Stochastic Methods in Image and Signal Processing,* Vol. 1766, pp. 480-488, 1992.

[81] L. A. Zadeh, "Fuzzy Sets," *Informat. Control,* vol. 8, pp. 338-353, 1965.