*The fact that we live at the bottom of a deep gravity well, on the surface of a gas covered planet going around a nuclear fireball 90 million miles away and think this to be normal is obviously some indication of how skewed our perspective tends to be.*

–Douglas Adams, 1998

**University of Alberta**

# ADAPTIVE LOGIC PROCESSING IN SYSTEM MODELLING AND KNOWLEDGE DISCOVERY

by

**Adam Fredrick Gobi** ©

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

Department of Electrical and Computer Engineering

Edmonton, Alberta
Fall 2005

**University of Alberta**

**Library Release Form**

**Name of Author**: Adam Fredrick Gobi

**Title of Thesis**: Adaptive Logic Processing in System Modelling and Knowledge Discovery

**Degree**: Master of Science

**Year this Degree Granted**: 2005

______________________________

**Date**: __________________

# Abstract

Recognizing the high suitability of fuzzy logic-based models for the task of system modelling and knowledge discovery, this work proposes a new approach to the automatic identification and optimization of fuzzy models. With a highly effective two-phase design process, we are able to realize adaptive logic processing in the form of structural and parametric optimization. Efficient structural learning is achieved through the use of well-established methods in Boolean minimization, with the resulting structure then refined with fuzzy neurons. The combination of a purely logic-driven architecture with this novel hybrid-learning scheme helps to develop transparent and accurate models while exhibiting superb computational efficiency. Further, this adaptive fuzzy modelling framework exhibits excellent potential for driving intelligent systems that must operate in dynamic and rapidly changing environments. In further studies, we investigate this avenue and identify various critical design issues as we propose a versatile neurofuzzy hardware platform.

# Acknowledgements

For his indispensable guidance during my research at the Universtiy of Alberta, the efforts of Dr. Witold Pedrycz are gratefully acknowledged. Additionally, I would like to thank Rob Chapman for his comments and advice during our many insightful conversations.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

One cannot dispute the benefits of modelling complex real-world systems, concepts, and processes, of which there are two main desires. First, if the model is an accurate one, not only is it able to behave in the same way the target system would under familiar conditions, but also for unforeseen circumstances where we would otherwise have no way of knowing how the system would react; this is accomplished through generalization of its knowledge, effectively achieving predictive capabilities. Further, if the model is sufficiently transparent, we are able to interpret and learn the details of how the system works, gaining powerful knowledge that was previously unattainable; as well, this communication through interpretability is not a one-way street, as a user could also potentially modify its structure in order to make corrections or augmentations. The importance of these two qualities become quite evident when referring to any of a large list of useful applications, where just a few examples include: business intelligence, ex. predicting property value or customer trends and understanding the cause of them; real-time process control, where we can achieve desired behaviour while potentially discovering ways to optimize its operation; robotics, ex. learning and generalizing for navigation in unpredictable environments, where its learnt knowledge could be checked by a designer to verify appropriate behaviour; medicine, such as automatic diagnosis of diseases and characterization of what causes them; the possibilities are endless.

Successfully modelling a complex system is a difficult task. As we are fundamentally in the realm of mathematics, we must attempt to quantify the target system numerically. This requires taking measurements of a few, tens or even hundreds of independent variables (attributes), each of which could represent a significant contributing factor in dictating the target's behaviour. Since this behaviour is precisely what we are trying to learn and understand (model), we encounter the problem of having to deal with high levels of dimensionality as the number of variables, or inputs, increases. Like attempting to understand the real world itself, one can potentially consider an infinite number of perspectives that may or may not have a significant impact on what we are trying to comprehend.

Attempting to build a model by hand is often impractical, and even when feasible (i.e. only a few variables to deal with), it can be an extremely tedious and time-consuming process requiring one or more domain experts for designing its architecture; such models can often be lacking in accuracy as well. With these difficulties, some means of automatic model identification is highly desirable and often utilized, where a machine may learn from raw system data in order to build a transparent knowledge base from which to act. However, we are still far away from an ideal framework for such a solution, making it an active area of research.

## 1.1 Objectives

It is important to emphasize that any model is constructed (designed) with some fundamental objectives in mind. These objectives are directly applicable in the development of some modelling framework, so the resulting methodologies may apply to a broad range of applications. In particular, there are several dominant goals that one may strive to accomplish in system modelling:

- *Accurate approximation abilities.* The model should be able to capture the nature of the data (system) by minimizing a certain performance index (ex. sum of squared errors), where the error is measured by comparing model output (behaviour) to the desired output observed from the system. The accuracy of this approximation should remain high even when computing output from new data, of which there is no observed system reaction, as we must be able to rely on the model to produce a correct result.

- *Intuitive interpretation abilities.* The model should be a transparent one, meaning that it describes the system in a format that is easily understood by the user, helps reveal the most essential dependencies, and quantifies the relationships at some level of information granularity. There is a fundamental guiding instrument here, known as Occam's razor principle. It states that a simple model is always preferred over a more complex one if both approximate (explain) the data to the same degree. Therefore, we should strive for the most concise explanations as possible. This would also greatly help in avoiding the problem of over-fitting training data, which can quickly destroy the models generalization abilities.

- *Robustness to high dimensionality.* The model should be reasonably capable of overcoming the "curse" of dimensionality. As mentioned above, quantifying real-world entities can often involve very high levels of dimensionality;

the model should possess faculties for dealing with such situations, effectively discovering which elements have significant impact on the system's behaviour.

- *High computational efficiency.* In several ways, the model should be an efficient one, wasting as little computation time as possible. For one, the model-building process of identification and subsequent optimization should be relatively fast, without sacrificing learning quality. As well, it is desirable for the finalized model to be proficient in its information processing, quickly producing accurate output (behaviour) when presented with new input data. The application area is critical here, as each have their own timing constraints; from this statement it is obvious that increased efficiency results in applicability to a wider rage of real-world problems.

Unfortunately, the two most fundamental modelling thrusts, namely accuracy and transparency, are somewhat in conflict with each other. For instance, efforts in achieving high accuracy quite often fall back on the "black box" information processing paradigm, where the knowledge residing in the resultant model cannot be appropriately communicated to the user in any kind of easily-understood format. As well, it can be quite difficult to autonomously learn a concise, highly interpretable architecture while maintaining high levels of accuracy. Further, in the presence of highly dimensional systems, these problems are amplified considerably. Additionally, coming up with effective solutions to these apparent problems can also result in the requirement of significantly longer computation times.

Motivated by these challenges, this thesis seeks to achieve a modelling framework that is able to strike a flexible and effective balance between accuracy and transparency, promoting efficiency even when dealing with high levels of dimensionality.

## 1.2 Approach

Traditionally, methods of autonomous learning and system modelling have dealt with information directly, often presented in the form of overly-detailed numerical information. However, as the complexity of a system increases, it becomes more difficult and eventually impossible to make a precise statement about its behaviour. It is quite evident that one does not always need such precision in numerical data, and can even cause problems when considering that there is often some degree of uncertainty within the information itself. In contrast, human beings have the ability to take in and evaluate all sorts of information from the physical world we are in contact with and mentally analyze, average and summarize all of this input data into an optimum course of action. In an attempt to mimic this behaviour, we can employ methods abstracting available system data, creating conceptual entities that embrace elements of visible similarity, functional adherence, and spatial or temporal proximity. These information granules arise as "building blocks" of the problem under discussion, and can be used to describe a system's behaviour and carry out processing at the level that is the most suitable for the designer of the system as well as pertinent to its potential user. This is method of information abstraction and processing is referred to as granular computing.

Within the realm of granular computing exists the technologies of fuzzy sets and fuzzy logic. Fuzzy sets are information granules that represent notions possessing some semantic meaning, such as *high* temperature, *medium* pressure, *small* neighbourhood, *low* speed, etc.; they can capture almost any condition for which we have words. The definition of such concepts can have unclear boundaries, i.e. they are "fuzzy" terms; fuzzy sets possess inherent abilities to represent and process this uncertainty. Upon constructing fuzzy sets to represent various concepts, we need a means of working with them. Fuzzy logic is the answer here, allowing us to combine various notions or abstractions together in order to infer some high-level action, essentially computing with words (something humans do every day).

Fuzzy sets and fuzzy logic have become one of the emerging areas in contemporary technologies of information processing; recent studies spread across various areas, from control, pattern recognition, and knowledge-based systems to computer vision and artificial life. A significant number of direct real-world implementations range from home appliances to industrial installations, with fuzzy sets by themselves or together with other modern approaches. Generally, all of these applications involve some sort of model development, defining linguistic rules on how to behave and react. Hence, the fundamental idea of fuzzy modelling emerges. Fuzzy models use fuzzy sets to define a perspective of the system's environment, and employ fuzzy logic for knowledge-based processing in order to approximate system behaviour. It is in this highly suitable environment that we seek to achieve the objectives outlined in this thesis, leading to the creation of a fuzzy model development framework for effective real-world system modelling.

## 1.3 Contributions

The key contributions of this thesis are twofold.

1. We introduce a highly effective fuzzy modelling framework, suitable for a wide range of applications. The heart of this framework consists of an efficient and accurate learning methodology for model identification even in the presence of highly-dimensional systems. It is capable of discovering concise, human-interpretable logic-based structures in data, which are then further refined in order to achieve high levels of accuracy and generalization while

retaining this transparency. As a direct result of the work, a comprehensive software-based fuzzy model development environment was created, called *Third Eye*.

2. In broadening the range of potential applications even further, we propose an optimized, digital hardware-accelerated platform supporting the novel fuzzy modelling framework introduced in this thesis. The conceptual platform is thoroughly examined through qualitative and quantitative investigations, laying a solid foundation for its future implementation.

## 1.4 Thesis Organization

The flow of the thesis is a very natural one; brief overviews of key technologies are given only when they have been formally introduced as part of the modelling framework; qualitative and quantitative analyses are presented upon the proposal of any new architectures or methodologies, including extensive experimentation for theoretical validation. What follows is a brief summary of the chapters to follow.

**Chapter 2: Fuzzy Modelling Fundamentals.** In this chapter we give an overview of fuzzy modelling and comment on the state-of-the-art. We also present a general model architecture and modelling scenario that is adhered to for the remainder of the thesis.

**Chapter 3: Fuzzy Modelling through Logic Minimization.** Here we introduce the fuzzy modelling framework in detail. Several design issues are presented along with solutions, and we conduct investigations that show the importance of the two-phase development methodology. The results of extensive experimentation are also presented, demonstrating the effectiveness of the framework.

**Chapter 4: Fuzzy Neural Networks for Intelligent Hardware Engines.** In this chapter we propose an hardware-based intelligent neurofuzzy platform based upon the aforementioned approach to fuzzy model identification. The architecture of the processing core of the platform is described, along with detailed experiments for investigating pertinent hardware design issues.

**Chapter 5: Closing.** We conclude this work with a review of our contributions along with a consideration of future work in the area of system modelling and knowledge discovery.

no text

# Chapter 2

# Fuzzy Modelling: Fundamentals and Beyond

In this chapter we give a short discussion of fuzzy sets and logic, proceeding with an overview of fuzzy modelling including a brief review of the state-of-the-art. We then provide some detailed architectural considerations along with a modelling scenario that is adhered to for the remainder of the thesis.

## 2.1 Fuzzy Sets and Logic

A set is an information granule specifying some existing commonalities between certain objects, bringing them together in a group. This idea occurs frequently in human behaviour as we tend to organize, summarize, and generalize knowledge about objects; the encapsulation of the objects into a collection whose members all share some general features or properties naturally implies the notion of a set. In defining these shared similarities, the nature of a set often has some linguistic quality, in order to group elements such as *fast* cars or *tall* buildings. As a result, employing sets as an abstraction of numerical information allows machines to "think" much like we do, perceiving the environment in more manageable chunks of information. This type of perspective is highly beneficial when operating in the real world. For instance, when perceiving the temperature of a room, humans have no need to know the exact temperature, something which may be constantly changing, but very slightly. That is, we're not especially concerned if it is $21.53^{o}$ C or $22.31^{o}$ C and it happens to increase to $22.85^{o}$ C, as long as it is *close* to standard room temperature, promoting comfort – there is no time or energy wasted on constant, overly-accurate temperature estimates.

However, a problem here lies in determining membership to a set, which is fundamentally a process of dichotomization, imposing a binary, all or none classification decision: either accept (*true*, 1) or reject (*false*, 0) an object as belonging

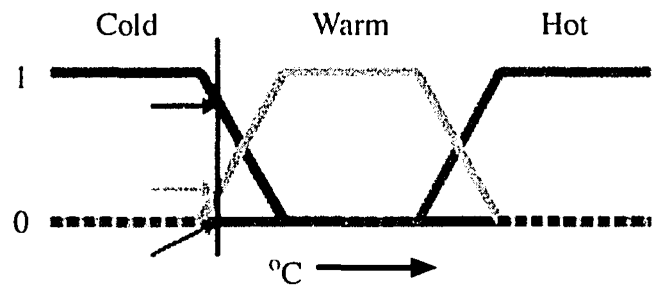


Figure 2.1: Fuzzy sets constructed over the temperature universe

to a given collection (set). This limitation is quite evident in the real world. In the context of temperature, for instance, how are we to determine if it is either exactly cold or warm? Is there some distinct boundary in which, as soon as the temperature reaches a specific level, we suddenly feel *warm* rather than *cold*? Most people would never entertain such a notion. A brief except from Borel [4], further demonstrates the issue.

*One seed does not constitute a pile nor two nor three... from the other side everybody will agree that 100 million seeds constitute a pile. What therefore is the appropriate limit? Can we say that 325,647 seeds don't constitute a pile but 325,648 do?* (Borel, 1950)

Recognizing the need for an effective way of representing this uncertainty, Lotfi A. Zadeh proposed the idea of *fuzzy* sets, coined in his seminal paper, "Fuzzy Sets" [32]. Fuzzy sets are an extension of classical set theory, characterized by a membership function which maps the elements of a universe of discourse into the unit interval. The value 0 means that the element is not included in the given set, and 1 describes a fully included member (this behaviour corresponds to the indicator function of classical sets). The values between 0 and 1 characterize fuzzy members. This idea of "fuzziness" allows a machine to deal with imprecision and uncertainty in "understanding" real-world concepts. As an example, in the universe of temperature, we can use fuzzy sets to represent the concepts of *cold*, *warm*, and *hot*, as seen in Figure 2.1. Here we see the use of trapezoidal membership functions; a point on the scale has three "truth values" or fuzzy activations – one for each of the three functions. For the particular temperature shown, the three truth values could be interpreted as describing the temperature as, say, "fairly *cold*", "slightly *warm*", and "not at all *hot*". In addition to the simplistic trapezoidal membership functions, there are many other types well-documented in the literature [21].

Table 2.1: Selected t/s-norm pairs ($X, Y \in [0,1]$)

| *Pair* | *Formulae* |
|---|---|
| Minimum | $minimum(X,Y)$ |
| Maximum | $maximum(X.Y)$ |
| Algebraic Product | $XY$ |
| Probabilistic Sum | $X+Y-XY$ |
| Lukasiewicz AND | $max(0,X+Y-1)$ |
| Lukasiewicz OR | $min(1,X+Y)$ |

Fuzzy logic is an extension of Boolean logic dealing with the idea of partial truth, captured by fuzzy sets. Hence, unlike Boolean logic, fuzzy logic allows for these partial (fuzzy) membership values between 0 and 1. It provides a means of performing operations on fuzzy sets to combine, compare, or aggregate them; these constructs are of an utmost importance in any situation involving information and data processing. Fuzzy set operations, known as triangular norms, are an extension of traditional set operations, principally including the classical set operations intersection and union, which parallel the Boolean operators AND and OR. For the AND operation, we have t-norms, and s-norms for the OR operation. While there are many different kinds of t/s-norm pairs [21], they all have the boundary conditions of Boolean AND and OR – see Table 2.1 for a list of the more popular pairs. Regarding another fundamental operation, the negation or Boolean NOT operator, this is usually taken as $1-X$, where $X$ is some truth value on the unit internal, [0,1].

Using fuzzy logic we can define rules for behaviour, in the form of "if *condition*, then *action*" statements, combining and aggregating fuzzy sets from different universes. For example, a rule-base describing outdoor comfort could partially consist of something like "if temperature is *warm* AND humidity is *low*, then comfort level is *high*, OR if temperature is *hot* AND humidity is *high*, then comfort level is *low*." Upon defining an entire rule-base for determining comfort level, machines can infer such a concept through fuzzy sets and the aforementioned logic-based t/s-norm operations.

## 2.2 Fuzzy Models

Fuzzy modelling [3,21,32] undoubtedly becomes vital whenever any application of fuzzy sets is anticipated. Briefly speaking, fuzzy models are modelling constructs featuring two main properties:

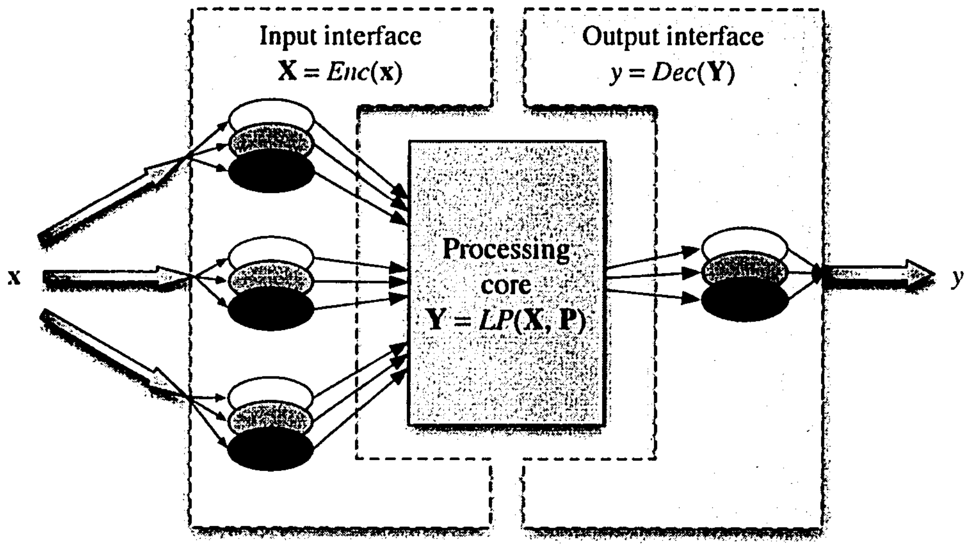

Input interface
$\mathbf{X} = Enc(\mathbf{x})$
Output interface
$y = Dec(\mathbf{Y})$
Processing core
$\mathbf{Y} = LP(\mathbf{X}, \mathbf{P})$
x
y

Figure 2.2: General topology of a fuzzy model

- They operate at the level of linguistic terms (fuzzy sets); similarly all system dependencies can be portrayed in the same linguistic format.
- They represent and process uncertainty.

As seen in Figure 2.2, a general fuzzy model has two fundamental functional components: (a) input and output interfaces and (b) a processing core. The interfaces allow interaction between the conceptual, logic-driven structure of the model and the physical world of measured variables. More specifically, the input interface realizes perception, where input variables are transformed into an internal format of information granules understood by the logic-processing core. The output interface communicates the results of processing (output fuzzy set activations) in a form understood by the external world (modelling environment). These actions can be referred to as fuzzy encoding (input interface) and decoding (output interface) or more traditionally, fuzzification and defuzzification. The processing core forms the most important component of the fuzzy model, consisting of a knowledge base containing the structure and details of system behaviour, realizing inference through granular computation. The topology of such cores often consist of fuzzy if-then rules.

### 2.2.1 State of the Art

In the realm of fuzzy modelling, there have been a number of main pursuits attempting to meet the modelling requirements outlined in the previous chapter. These goals have been the driving force behind efforts in developing effective methodologies under the framework of computational intelligence (CI) [7,17], which promotes effective synergism between fuzzy logic, (artificial) neural networks [1], and evolutionary computing [2]. Neural networks [8] possess excellent learning abilities for mapping experimental data, but the resulting constructs are unstructured and very difficult for a human to comprehend, with its learned knowledge trapped within a "black box". Consequently, they tend to complement fuzzy systems quite well, which exhibit superb reasoning capabilities and highly interpretable logic-oriented processing, but limited learning ability when presented with example data. Evolutionary computing [6] provides effective methods of global optimization and search, proving beneficial to both fuzzy and neural systems; more popular evolutionary methods include genetic algorithms and genetic programming [6].

In recent years, CI-based approaches to fuzzy modelling have become quite dominant, providing new modelling techniques or improvements to existing ones. Most commonly seen are methodologies developed using fuzzy and neural methods, often referred to as neurofuzzy or fuzzy-neural approaches. There are many existing in the literature [11–13, 16, 21, 29], some of which employ evolutionary methods [18, 22, 23, 25, 26]. Additionally, techniques using data clustering, especially fuzzy clustering [20], have been applied to fuzzy modelling [1, 27, 31], however the accuracy of resulting models were often somewhat lacking. To improve upon this, researchers have applied evolutionary techniques in cluster-based modelling for subsequent optimization [5, 14, 24, 30]. There have also been many other so-called fuzzy-evolutionary techniques [2, 9, 10, 19, 21, 28].

In reviewing the referenced literature, a number of drawbacks become apparent. Techniques involving evolutionary optimization often suffer from substantial computing requirements, as the population-based algorithms are quite computationally intensive, especially when members of the population consist of complex structures such as neural networks. Neurofuzzy models tend to gravitate toward meeting a high accuracy requirement that happens at a substantial expense of lowering their transparency; this is somewhat inevitable considering the underlying black-box processing paradigm and various topologies existing in neurocomputing. As well, within these CI-based approaches, there are numerous instances involving some sort of optimization of input interfaces in order to achieve higher levels of accuracy; in many cases the distribution and/or shape of membership functions are modified so heavily that any semantic meaning is destroyed, with them longer able to represent concrete linguistic terms such as those seen in Figure 2.1. Along with this ever-apparent balancing act between accuracy and transparency, what becomes

[1]Neural networks are parallel and distributed computational structures composed of numerous simple processing elements (neurons), inspired by biological nervous systems.

[2]Evolutionary computing encompasses any and all population-based optimization tools using mechanisms inspired by biological evolution.

quite apparent in fuzzy modelling are the growing difficulties when dealing with multi-variable systems. The transparency of the models, usually regarded as some type of rule-based system, amplify these difficulties even further. The "curse" of dimensionality becomes apparent even for quite small rule-based systems, and as such we see the vast majority of modelling problems handled in the literature to be modest in terms of the number of system variables. Overall, it is apparent that the successes of these hybrid development environments are somewhat limited. As such, we have not reached a state where large models could be built quite efficiently and accurately while retaining the semantic properties of the resulting constructs.

### A Promising Technology

It is evident that the logic-based fabric of fuzzy models is an inherent facet of this modelling paradigm that needs to be preserved. While neurocomputing is an important technology contributing to the learning abilities and subsequently to the parametric optimization of the fuzzy models, it is of paramount importance to develop a synergistic environment in which the logic transparency of the ensuing models is not compromised. Perhaps one of the more interesting approaches coming out of the literature are Pedrycz's fuzzy neural networks (FNN). Introduced and studied in [16], the idea behind fuzzy neurons and fuzzy neural networks was to develop a structure that is at the same time transparent and adaptive. The transparency facet is gained due to the logic type of processing realized by the neurons. There are two fundamental classes of the neurons that is OR and AND neurons. Interestingly, these are generalizations of standard OR and AND gates encountered in digital systems. The neurons exhibit learning abilities as they come with a collection of adjustable connectives (weights). In this setting of fuzzy neurons, the synergy of learning and transparency is well articulated. Different application-oriented aspects of the resulting fuzzy neural networks were discussed in detail in a number of previous studies [15, 16, 18].

Due to the unique and beneficial properties of fuzzy neurons, they become an important component in our model development framework introduced in next Chapter. Specifically, the work aims to provide comprehensive and efficient methods of building and optimizing fuzzy neural networks with knowledge-based structures.

## 2.3 Architectural Considerations

In order to put things into perspective for the remainder of the thesis, in this section we present our view of how the architecture of a fuzzy model should take shape, along with some details pertaining to the general identification (learning) process, with well-defined notation.

### 2.3.1 General Architecture of the Fuzzy Model

Here we concentrate on the architecture of the fuzzy model and elaborate on its functional aspects and interaction with the numeric world. The fuzzy model follows the fundamentals of fuzzy (granular) modelling. As advocated in [16], fuzzy modelling is realized at the conceptual level formed by a collection of semantically meaningful information granules defined in each variable. These are also regarded as linguistic landmarks whose choice implies a certain point of view of the data (system) under discussion. The architecture of the model follows the geometry of multidimensional data and reflects the main objective of such modelling, which is to cover the data by a series of "patches". Each patch is a fuzzy relation formed with the use of fuzzy sets defined in each variable. As we require several patches, these are combined together by a union operation, and a fuzzy model emerges. This idea gives rise to a two-level topology of the model that captures the geometry of data.

Evidently, the geometry of the model stands in a one-to-one correspondence of its logic fabric. The essence of this geometry can be captured in the form of AND and OR nodes (aggregation operations) as illustrated in Figure 2.3, forming the processing core. This figure emphasizes the structural nature of this construct. Considering the specific information granules shown there, we can translate it into the description:

$$(A_1 \text{ and } B_2 \text{ and } D_4) \text{ or } (A_3 \text{ and } C_2) \text{ or } (D_1 \text{ and } C_3)$$

where each list is composed of fuzzy sets defined in the corresponding spaces ($A$, $B$, $C, \ldots$). These lists can be considered analogous to the "if" portion (antecedent) of fuzzy rules, with the consequent ("then" statement) defined as the particular output information granule (ex. *high* comfort) it is trying to represent.

The AND and OR nodes seen in Figure 2.3 can be represented by a variety of operations for relational computation. The most fundamental logic-based aggregation operations occur within the realm of two-valued logic, where we have simple AND and OR gates for processing binary data. Upon entering the realm of continuous (fuzzy) logic, we require more sophisticated components for realizing these logic operations, coming in the form of t/s-norms we defined earlier.

The network in Figure 2.3 pertains to a single information granule (fuzzy set) in the output space. In case of a number of fuzzy sets there, the architecture is augmented with several of these structures representing logic-based descriptions of their respective output granule. With each one capturing the essence of high-level concepts, together they form a heterogeneous knowledge base describing behaviour of the entire system.

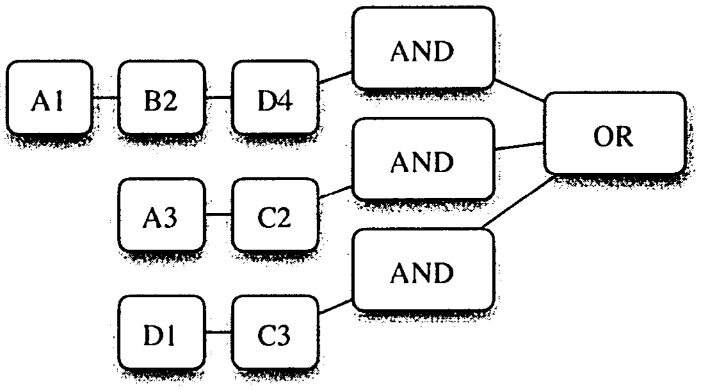


Figure 2.3: Structure of a fuzzy model represented as an aggregation of information granules (fuzzy sets)

## 2.3.2 Modelling Scenario

Referring to Figure 2.2, from the functional standpoint, the fuzzy model is a structure of three mappings put in series, that is the encoder $\mathbf{X} = Enc(\mathbf{x})$, processing core $\mathbf{Y} = LP(\mathbf{X}, \mathbf{P})$, and the decoding part $y = Dec(\mathbf{Y})$. In direct correspondence with system input and output, $\mathbf{x}$ and $y$ represent input and output of the model manifesting at the numeric level, where $\mathbf{x} \in \Re^n$ and $y \in \Re$. Internally, $\mathbf{X}$ and $\mathbf{Y}$ are information granules containing elements in the unit hypercube. Let $n', m'$ denote the number of information granules defined for input and output spaces, i.e. $\mathbf{X} \in [0,1]^{n'}$ and $\mathbf{Y} \in [0,1]^{m'}$. Finally, we have $\mathbf{P}$, representing some adjustable parameters of the model.

The experimental data measured from the target system are used as training examples, taking the form of $N$ input-output pairs, i.e. $\{\mathbf{x}(c).target(c)\}$, $c = 1,2,\ldots,N$. We require that $y(c)$, the output of the fuzzy model for the input $\mathbf{x}(c)$, is equal to $target(c)$, $y(c) \approx target(c)$. When concerned with the models internal optimization, namely the fabrication and optimization of the processing module, the original training data are converted through the model interfaces to an internal format, that is $\{\mathbf{X}(c), \mathbf{TARGET}(c)\}$, where $\mathbf{TARGET}$ denotes the $m'$ target output information granules. This leads to to the requirement $\mathbf{Y}(c) \approx \mathbf{TARGET}(c)$, allowing us to focus on the mapping of the information granules rather than the experimental data. This approach tends to promote the interpretability of the model.

During the identification process, it is evident that two distinct learning requirements must be met: 1) the resulting model should have some sort of transparent, heterogeneous structure similar to what we see in Figure 2.3, essentially providing a knowledge-based architecture and 2) for improvements in accuracy and to better describe relationships and dependencies between variables in the data, any parameters of the model should be properly adjusted (optimized).

## 2.4 Review

In this chapter we gave a brief overview of fuzzy sets and logic and their role in fuzzy models: constructs highly suitable for system modelling and knowledge discovery. Upon presenting the state-of-the-art in fuzzy modelling, comments were made on various shortfalls seen in the literature, ones which this thesis tries to provide solutions for. With the introduction of the general architecture of the fuzzy model we are interested in along with a generic modelling scenario, we may now proceed with the introduction of our framework for automatic model design.

## Bibliography

[1] R. Babŭska and H. B. Verbruggen. A new identification method for linguistic fuzzy models. In *Proc. FUZZ-IEEE/IFES'95*, pages 897–904, 1995.

[2] A. Bastian. Identifying fuzzy models utilizing genetic programming. *Fuzzy Sets Syst.*, 113(3):333–350, 2000.

[3] J. C. Bezdek. Fuzzy models – what are they, and why? *IEEE Trans. Fuzzy Syst.*, 1:1–6, 1993.

[4] E. Borel. *Probabilite et Certitude*. Press Université de France, Paris, 1950.

[5] M. Delgado, A. F. Gomez-Skarmeta, and F. Martin. A fuzzy clustering-based rapid prototyping for fuzzy rule-based modeling. *IEEE Trans. Fuzzy Syst.*, 5:223–233, 1997.

[6] A. E. Eigen and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, New York, 2003.

[7] D. B. Fogel and C. J. Robinson, editors. *Computational Intelligence: The Experts Speak*. Wiley-IEEE, New York, 2003.

[8] S. Haykin. *Neural Networks: a Comprehensive Foundation*. Prentice Hall, New Jersey, 2nd edition, 1999.

[9] C. L. Karr and E. J.Gentry. Fuzzy control of ph using genetic algorithms. *IEEE Trans. Fuzzy Syst.*, 1:46–53, 1993.

[10] D. Kim and C. Kim. Forecasting time series with genetic fuzzy predictor ensemble. *IEEE Trans. Fuzzy Syst.*, 5:523–535, 1997.

[11] D. Kukolj and E. Levi. Identification of complex systems based on neural and takagi-sugeno fuzzy model. *IEEE Trans. Syst., Man, Cybern. B*, 34:272–282, 2004.

[12] S. Mitra and Y. Hayashi. Neuro-fuzzy rule generation: Survey in the soft computing framework. *IEEE Trans. Neural Networks*, 11:748–768, 2000.

[13] K. C. Ng and M. M. Trivedi. A neuro-fuzzy controller for mobile robot navigation and multirobot convoying. *IEEE Trans. Syst., Man, Cybern. B*, 28:829–840, 1998.

[14] B. J. Park, W. Pedrycz, and S. K. Oh. Identification of fuzzy models with the aid of evolutionary data granulation. *IEEE Proc. Control Theory Appl.*, 148:406–418, 2001.

[15] W. Pedrycz. Fuzzy neural networks and neurocomputations. *Fuzzy Sets Syst.*, 56:1–28, 1993.

[16] W. Pedrycz. *Fuzzy Sets Engineering*. CRC Press, Boca Raton, FL, 1995.

[17] W. Pedrycz. *Computational Intelligence: An Introduction*. CRC Press, Boca Raton, FL, 1997.

[18] W. Pedrycz. Heterogeneous fuzzy logic networks: Fundamentals and development studies. *IEEE Trans. Neural Networks*, 15:1466–1481, Nov. 2004.

[19] W. Pedrycz. Logic-driven fuzzy modeling with fuzzy multiplexers. *Engineering Applications of Artificial Intelligence*, 17:383–391, 2004.

[20] W. Pedrycz. *Knowledge-Based Clustering: From Data to Information Granules*. Wiley, New York, 2005.

[21] W. Pedrycz and F. Gomide. *An Introduction to Fuzzy Sets: Analysis and Design*. MIT Press, Cambridge, MA, 1998.

[22] W. Pedrycz and M. Reformat. Evolutionary fuzzy modeling. *IEEE Trans. Fuzzy Syst.*, 11:652–665, 2003.

[23] W. Pedrycz and M. Reformat. Genetically optimized logic models. *Fuzzy Sets Syst.*, 150:351–371, 2005.

[24] H. Roubos and M. Setnes. Compact and transparent fuzzy models and classifiers through iterative complexity reduction. *IEEE Trans. Fuzzy Syst.*, 9:516–524, 2001.

[25] M. Russo. FuGeNeSys - a fuzzy genetic neural system for fuzzy modeling. *IEEE Trans. Fuzzy Syst.*, 6:373–388, 1998.

[26] M. Russo. Genetic fuzzy learning. *IEEE Trans. Fuzzy Syst.*, 4:259–273, 2000.

[27] S. K. Sin and R. J. P. deFigueiredo. Fuzzy system design through fuzzy clustering and optimal predefuzzification. In *Proc. IEEE Int. Conf. Fuzzy Systems*, pages 190–195, 1993.

[28] Y. Tang and Y. Xu. Application of fuzzy naive bayes and a real-valued genetic algorithm in identification of fuzzy model. *Information Sciences*, 169:205–226, 2005.

[29] L. H. Tsoukalas and R. E. Uhrig. *Fuzzy and Neural Approaches in Engineering*. Wiley, New York, 1997.

[30] Z.-Y. Xing, W.-L. Hu, and L.-M. Jia. A fuzzy clustering based approach for generating interpretable fuzzy models. In *Proc. of the 3rd Int. Conf. on Machine Learning and Cybernetics*, pages 2093–2097, 2004.

[31] Y. Yoshinari, W. Pedrycz, and K. Hirota. Construction of fuzzy models through clustering techniques. *Fuzzy Sets Syst.*, 54:161–180, 1993.

[32] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

no text

# Chapter 3

# Fuzzy Modelling through Logic Minimization

In this chapter we conceptualize, construct, demonstrate and validate a highly effective fuzzy modelling framework, starting with a brief overview of the approach. We then describe the details of the learning process, followed by overviews of the key technologies involved. In an effort to finalize details of the framework, important design issues are discussed and resolved. We then proceed to demonstrate the importance of structural learning, followed by extensive experiments in validating the worth of this new approach to fuzzy model identification.

## 3.1 Overview

Here we provide an overview to our approach for the development of fuzzy models. Primarily focusing on the core of the model, we propose a two-phase design process realizing adaptive logic processing in the form of structural and parametric optimization. Taking a significant departure from the main design direction, the fundamental link between binary and fuzzy logic is exploited. The wealth of design tools of two-valued logic is immense [2,4–9,17,20], with techniques capable of handling large problems and dealing with hundreds of binary variables. The underlying objective there is to minimize Boolean functions so that the ensuing realization could be made as compact as possible. One of the tools, Espresso [2], helps satisfy this important goal, originally intended for digital circuit synthesis. Given the state of the art of handling and simplifying Boolean functions, our objective is to capitalize on this well-established framework and treat it as an important phase in the design of fuzzy models. The proposed conceptual setting can be succinctly represented in the form shown in Figure 3.1.

It is instructive to briefly elaborate on the main phases of this scheme:

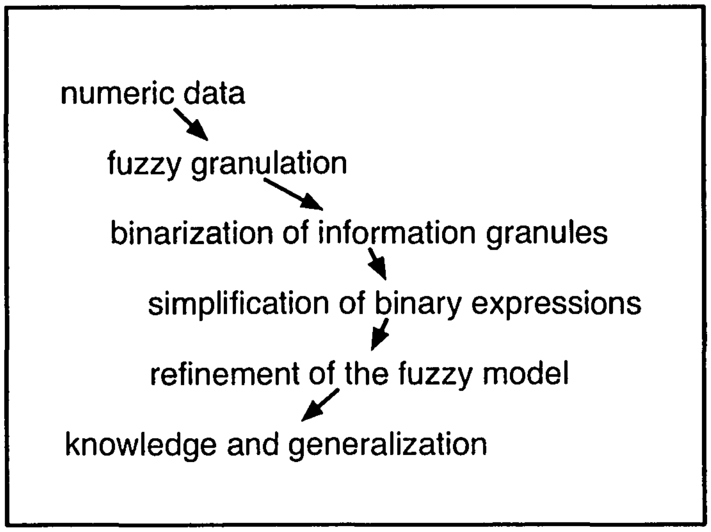

numeric data
fuzzy granulation
binarization of information granules
simplification of binary expressions
refinement of the fuzzy model
knowledge and generalization

Figure 3.1: General design flow of the fuzzy model development.

1. Fuzzy granulation is the same as encountered in any fuzzy model; we granulate each system's variable through a collection of semantically meaningful information granules (fuzzy sets).

2. The binarization helps encode these fuzzy sets into a binary format and consists of two steps, i.e. thresholding and encoding.

3. Optimization of the binary model, which is realized in terms of well-known techniques of two-valued logic synthesis. The result of such minimization leads to the underlying structure of the fuzzy model.

4. Given the structure of the model we refine it, moving back to the fuzzy data (produced by the fuzzy granulation) and proceeding with a neurofuzzy augmentation of the model with the help of fuzzy AND and OR neurons.

This unique combination of the logic-driven architectures of the models along with the hybrid-learning scheme helps to develop transparent and accurate models while maintaining excellent computational efficiency. From the very beginning of the design, the logic fabric of the structure of the model is present, assuring a large degree of interpretability. As well, with suitable neural enhancements the specific logic nature of its underlying structure is preserved while proceeding with parametric refinement of the model for achieving high levels of accuracy and more intuitive interpretability. This design methodology clearly distinguishes between structural and parametric learning, being viewed as two distinct phases of model building. In this sense, we do not bias the optimization by attempting to learn structure and parameters simultaneously.

## 3.2 Two-phase Model Development

Through the interfaces, the processing core of a fuzzy model receives information that has been abstracted into granules, upon which it performs intelligent computation. The core emulates aspects of human thought and cognition, dealing with information at a granular level and using logic-oriented reasoning in order to solve a problem. Its primary purpose is to learn and approximate a system's behaviour, building and drawing upon a knowledge base that may be interpreted and modified by the user. Here the logic-processing core is created in two phases, taking advantage of a "bottom-up" approach. Starting from a completely unstructured state, the entity learns from example, using the training data measured from the target system. With this information it can adapt itself into a structure closely resembling the target system's underlying mechanisms; these are the hidden inner workings that make up the very essence of the system and its behaviour. Following structural optimization, the core continues to learn during a second phase, this time focusing on the finer details of the system in order to improve upon behavioural approximation and provide more intuitive interpretation.

### 3.2.1 Phase 1: Structure Discovery

The nature of the information granulation within a fuzzy model deals with partial (fuzzy) membership to fuzzy sets, with aggregation carried out using triangular norms. Using fuzzy versus binary membership is advantageous in overcoming real-world problems of input imprecision, uncertainty, and output accuracy. However, traditional Boolean logic processing has previously unseen potential as an important component of a comprehensive fuzzy modelling environment. Given the fundamental link between two-valued and fuzzy logic processing, there exists here an excellent opportunity to take advantage of existing tools and methodologies of logic minimization.

In understanding this link, recall the well-known Shannon's theorem, a powerful concept from two-valued systems [19]. It states, in part, that any Boolean function realizing a mapping $\{0,1\}^n$ to $\{0,1\}$ can be uniquely represented as a logical sum of products. We refer to a product as an AND combination of input variables of the function, and sum as the final OR aggregation. View that this is precisely what we are trying to achieve in the continuous domain, using information granules in the logic-processing realized by the two-level structure seen in Figure 2.3.

For building and optimizing structure, this first phase of the cores "birth" employs tools of logic optimization. By temporarily converting the fuzzy granules into ones with strict membership boundaries, the information becomes inherently binary in nature, giving us a rough approximation of the target system. Of course, some information is lost through this process, yet the most important features and relationships explaining the system's fundamental behaviour are retained. In this form we can then access established methodologies of logic minimization, providing an excellent means of discovering a concise, logic-based structure within the system data. Upon returning to the domain of continuous logic, this newly discovered structure may be directly utilized in processing the original fuzzy information granules, replacing its binary AND and OR operations used during the logic minimization with triangular norm computations. With this augmentation we could achieve the structure seen in Figure 2.3.

### 3.2.2 Phase 2: Parametric Refinement

It is worth stressing that even though the information granules convey detailed numeric information in the format of their membership functions, the resulting structure in Figure 2.3 does not include any other numeric quantification. A calibration of the structure is possible by equipping it with some parametric flexibility, leading to increased accuracy in behavioural approximation, as well as a higher level of interpretability as we gain more insight into the interaction between system variables and behavioural rules. In the second phase of core development, the refinement of this nature can be completed by introducing fuzzy AND and OR neurons in place of the existing nodes, of which there is a direct linguistic correspondence.

Through fuzzy and neural methods we are able to derive a parameterized knowledge-based network; here we take advantage of the profound learning abilities of neural networks with their parametric architecture and effective training algorithms, combined with the transparency and logic-oriented processing of the fuzzy model itself. By endowing the fuzzy neuron-based core with the ability to adjust its connections, it can learn the finer details of the system, with its knowledge-based structure helping the core from falling into pitfalls of over-training and data memorization. Thus, when subjected to the appropriate learning process, this neurofuzzy core can improve upon accuracy, transparency, robustness and generalization.

## 3.3 Key Technologies

In this section, we provide more detailed overviews of the key technologies employed in our model development framework, namely logic minimization and fuzzy neural networks.

### 3.3.1 Two-Valued Logic Minimization

Boolean logic minimization is best known as the main part of logic synthesis, which converts a logic function to a circuit. Logic design consists of the manipulation of a logic representation without modifying the functionality, with logic minimization seeking a representation with a minimal number of implicants and literals. Through binarization of the information granules formed in the interfaces of a fuzzy model, we are able to realize a novel application for logic minimization in the form of structural learning; these efficient tools can help us in sorting through large amounts of data, eliminating redundancy and producing a simplified, compact and equivalent result in the form of a logic-based structure.

#### Background

In providing some background on logic synthesis, the following definitions are found in [17].

The set of binary values are defined as $\mathbf{B} \equiv \{0,1\}$. $\mathbf{B}^n$ can be modelled as a binary $n$-dimensional hypercube, where each element $e = (e_1,\ldots,e_n) \in \mathbf{B}^n$ is called a **minterm**. Boolean algebra comes from combining the set $\mathbf{B}$ together with the operations $+$ (disjunction, sum, OR), and $\bullet$ (conjunction, product, AND).

A Boolean function $f$ for $n$ variables, $x_1,\ldots,x_n$, is a mapping $f : \mathbf{B}^n \mapsto \{0,1,*\}$, with $*$ being a *don't care* condition for when the value of the function is irrelevant. Each minterm in $\mathbf{B}^n$ tells us values of the function variables, i.e. $x_1 = e_1, x_2 = e_2$, etc. All minterms for which $f$ has value 1 form the ON-set of the function, with the OFF-set and DC-set defined as sets of minterms where $f$ is 0 and $*$, respectively. If a Boolean function has more than one output, it realizes a mapping $f : \mathbf{B}^n \mapsto \{0,1,*\}^m$, each output having their own ON, OFF, and DC sets.

Each variable $x_i$ has two **literals** associated with it: $x_i$ and its complement $\bar{x}_i$. The literal $x_i(\bar{x}_i)$ represents a Boolean function evaluating to 1 (0) for minterms with $e_i = 1$, and to 0 (1) for minterms with $e_i = 0$. A **product** term is a Boolean product (AND) of literals, which evaluates to 1 for a minterm $e$ if each literal included in the product evaluates to 1, otherwise evaluating to 0. In the former case, the product is said to contain $e$. Since a product corresponds to a set of adjacent minterms in the binary $n$-cube, a product may also be referred to as a **cube**. A sum-of-products is a Boolean sum (OR) of products, evaluating to 1 for a given minterm if some product contains the minterm.

An **implicant** of a Boolean function is a cube that contains no minterm in the OFF-set. A **prime implicant** is an implicant contained in no other implicant of the function. An **essential prime implicant** is a prime implicant containing at least one ON-set minterm which is not contained in any other prime implicant.

A **cover** of a Boolean function is a set of implicants interpreted as a sum-of-products, which evaluates to 1 for all minterms in the ON-set, and none of the

OFF-set. The term **prime cover** is used to refer to a cover containing only prime implicants.

## Minimization Algorithms

The problem of two-level logic minimization is to find a cover for $f$ that minimizes a given cost function. Such a cover can be implemented as a minimum-cost sum-of-products equation. The cost, or size of a cover often includes parameters such as the number of cubes (products) in the cover, or the number of literals.

The Quine-McCluskey method [8] was one of the first exact methods for two-level logic minimization, based on the observation that the implicants in a minimum-cost cover can be restricted to prime implicants. The algorithm consists of two steps: (1) generate the set of all prime implicants; and (2) select a minimum number of prime implicants such that each minterm in the ON-set is contained. Although exact algorithms are useful, the exact two-level minimization problem involves computationally intractable problems. In many cases, getting satisfactory results (near optimal perhaps) in far less time is often more important, leading to the development of heuristic logic minimization tools. ESPRESSO-II [2] is a state-of-the-art algorithm for heuristic logic minimization, forming the main component of the Espresso software distribution, developed in the 1980s as a tool for programmable logic array (PLA) design.

The output of ESPRESSO-II is a sum-of-products cover, which in practice is almost always near minimum in cardinality. The algorithm's basic goal is to take a verbose representation of a logic function and produce a condensed representation, essentially learning its underlying structure. The general 8-step ESPRESSO-II algorithm is described in detail in [2], and is briefly outlined here:

1. Complement: Compute the complement of the input and the DC-set, i.e compute the OFF-set.

2. Expand: Expand each implicant into a prime and remove covered implicants.

3. Essential Primes: Extract the essential primes and put them in the DC-set.

4. Irredundant Cover: Find a minimal (optionally minimum) irredundant cover.

5. Reduce: Reduce each implicant to a minimum essential implicant.

6. Iterate 2,4,5 until no improvement.

7. Lastgasp: Try reduce, expand, and irredundant cover one last time using a different strategy, If successful, continue the iteration.

8. Makesparse: Include the essential primes back into the cover and make the PLA structure as sparse as possible.

The fundamental Espresso algorithm has since been modified to include some improvements and further heuristic tuning, with the source code for the software readily available on the Internet [1]. While there may be newer algorithms claiming superiority in one or more specific problem areas [4, 5, 17], the software and source code are not as easily obtained. Regardless, Espresso is still regarded as the standard two-level logic minimization tool in the (VLSI) design automation community, shown to be quite capable of handling situations involving even hundreds of Boolean variables.

### 3.3.2 Fuzzy Neural Networks

Fuzzy neurons seamlessly combine transparent, logic-oriented processing with learning abilities stemming from their adjustable connections. These adaptive logic-processing elements connect to each other in forming a heterogeneous fuzzy neural network. An $n$-input single output OR neuron is described in the form:

$$y = OR(\mathbf{x};\mathbf{w})$$

where $\mathbf{x}, \mathbf{w} \in [0,1]^n$. The connections, $w_1$, $w_2$, , $w_n$ are arranged in a vector form $(\mathbf{w})$. Rewriting the previous expression in a coordinate-wise manner, we obtain

$$\mathop{\mathbf{S}}_{i=1}^{n} x_i \,\mathbf{t}\, w_i$$

where the **t** and **s** operators are realized by t/s-norms, respectively. Essentially, the neuron realizes an s-t composition of the corresponding finite sets **x** and **w**. The AND neuron $y = AND(\mathbf{x};\mathbf{w})$ is governed by the expression

$$\mathop{\mathbf{T}}_{i=1}^{n} x_i \,\mathbf{s}\, w_i$$

Computationally, this neuron realizes a t-s composition of **x** and **w**.

The role of the connections in both neurons is to weight the inputs and in this way furnish them with required parametric flexibility. In case of OR neurons, the higher the connection, the more essential the associated input. For AND neurons, an opposite situation holds: lower connection indicates that the respective input is more essential. In general, a certain thresholding operation can be sought. For any OR neuron, we may consider the input irrelevant if the associated connection assumes values lower a certain threshold. An input of the AND neuron can also be eliminated if the connection value *exceeds* a specified limit.

[1]http://www-cad.eecs.berkeley.edu/software.html

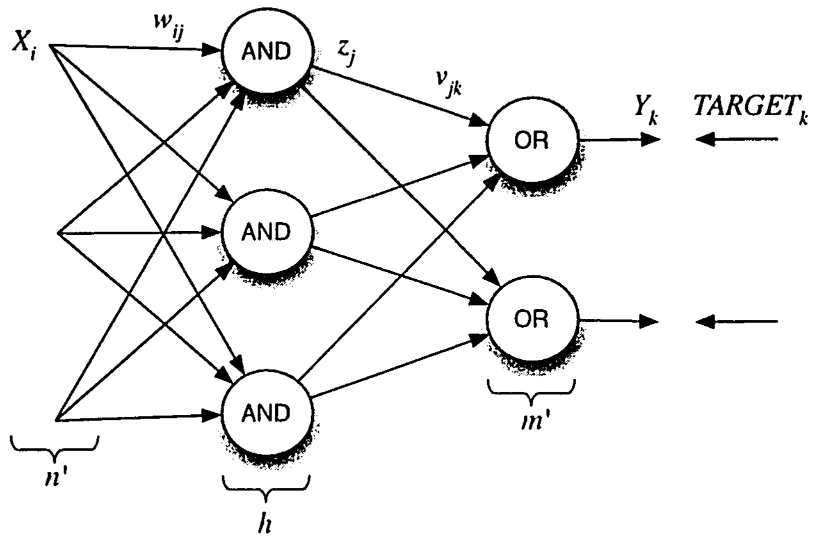


Figure 3.2: Generic, fully-connected fuzzy neural network along with detailed notation.

This demonstrated parametric flexibility is essential to developing the learning capabilities of fuzzy neural networks formed by these neurons. Combined with their inherent interpretability, they become excellent candidates for forming the processing core of our fuzzy model, with the structure derived from results of logic minimization. Such a knowledge-based structure comes as a realization of logic expressions capturing the behaviour of experimental data representing real-world concepts. It is essentially a rule-based description in the form of a collection of *if-then* statements combined together in an OR-wise manner. These rules directly correspond to a two-layer structure of the FNN, as seen in Figure 3.2 for a fully-connected (i.e. structureless) example. The first layer consists of AND neurons forming a collection of conditions of the rules, with the output layer of OR neurons aimed at aggregation of rules having the same conclusion.

Returning to the notation presented in Section 2.3.2, the network may be fully described by two matrices of connections. We treat $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_h$ and $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{w}_{m'}$ as vectors of connections for individual neurons. Corresponding to Figure 3.2, we have the $j$th AND neuron ($j = 1,2,\ldots,h$) reading as $z_i = AND(\mathbf{X}; \mathbf{w_j})$, where $\mathbf{X} = X_1, X_2, \ldots, X_{n'}$; the $k$th OR neuron ($k = 1,2,\ldots,m'$) reads as $Y_k = OR(\mathbf{z}; \mathbf{v_k})$, where $\mathbf{z} = z_1, z_2, \ldots, z_h$.

## Training the Network

To be of any use, a fuzzy neural network's connections must be properly adjusted during an appropriate learning process. A principal idea of parametric learning, no matter how it is implemented, can be portrayed as follows. Consider the task of learning the nature of a single output information granule, $Y_k$. For a given collection of $N$ input-output pairs of data $\{\mathbf{X}(c), TARGET_k(c)\}, c = 1,2,\dots N$, we want to modify the network's parameters (connections) to minimize the performance index $Q$, defined here as a squared error:

$$Q(c) = [Y_k(c) - TARGET_k(c)]^2$$

where $Y_k(c)$ is the activation level of the output fuzzy set $Y_k$ $(k = 1,2,\dots,m')$ for a given training instance $c$.

In terms of speed and efficiency, an on-line, gradient-based method of learning is desirable, having been well-developed and thoroughly documented in the literature [11, 12]. The general scheme of learning can be qualitatively described as

$$\Delta\mathbf{connections} = -\alpha\frac{\partial Q}{\partial\mathbf{connections}}$$

where $\alpha$ denotes a learning rate. Subsequently, the parameters of the network are adjusted following these increments:

$$new\_\mathbf{connections} = \mathbf{connections} + \Delta\mathbf{connections}$$

In the case of our model, the underlying structure is already pre-defined through logic optimization, causing the size of the learning problem to be greatly reduced from traditional fully connected networks as seen in Figure 3.2. This allows us to concentrate on only the most important connections rather than training the network from scratch.

In addition to the numeric calibration of the network, the connections of the fuzzy neural network help prune the original structure, done by applying the thresholding operation discussed above. The network after pruning can be represented in an equivalent rule-based format:

If $condition_i$ and $condition_j$ and ... then $conclusion_k$

The format of the rules varies as each rule may have a different number of conditions. In this setting, the connections of the fuzzy neural network can be interpreted as calibration factors of the conditions and rules. The connections of the AND neuron modify the membership functions of the fuzzy sets contributing to the Cartesian product of the overall condition part of the rule. The higher the value of the connection, the less specific is the fuzzy set. In the limit, when the connection is equal to 1, we end up with the corresponding fuzzy input being eliminated from the rule (in this way the rule becomes more general). The connections of the OR neuron determine confidence of the rule, meaning that the Cartesian product (overall condition of the rule) is quantified in terms of its relevance.

With the details of learning given, it is beneficial to define a few terms used in measuring the performance of the networks. With $Q$ defined as above, we use a sum-of-squared-errors (SSE) measure for an indication of training progress of individual fuzzy output granules:

$$Q_N = \sum_{c=1}^{N} Q(c)$$

Additionally, we use a root-mean-squared-error (RMSE) formula to measure performance of the model in terms of its numeric output (decoded from fuzzy output granules) and target system output:

$$V = \sqrt{\frac{1}{N}\sum_{c=1}^{N}[y(c) - target(c)]^2}$$

## 3.4 Design Issues

With a general framework in place for effective fuzzy model development, there are several design issues that need to be identified, considered, and resolved. In this section we discuss these problems and propose solutions.

### 3.4.1 Granular Interfacing

As mentioned previously, in the realm of fuzzy modelling information granulation is carried out as an interface to continuous system variables. These information granules take the form of semantically meaningful fuzzy sets, distributed fully over each variable's universe of discourse. There have been a number of theoretical and practical investigations into the nature of the interfaces where numerous issues concerning the number of fuzzy sets, their distribution and ensuing optimization have been discussed, cf. [11]. Apparently, the optimization of the fuzzy sets standing in these interfaces could be beneficial to the model and contribute to the enhancement of its quality. Accepting this point of view, we must emphasize the main focus of this study: the logic processing core of the fuzzy model and its optimization. In the interest of both simplicity and semantic integrity, by default we choose to use membership functions spread evenly across variable spaces, although this could certainly be customized if there happened to be an expert who was able to suggest better interval distributions. Regardless, our main goal here is to construct

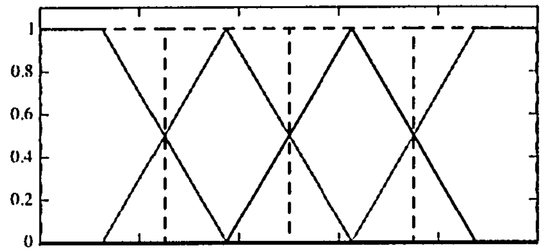


Figure 3.3: Example interface consisting of triangular fuzzy membership functions and their induced binary representations (shown with dotted lines).

semantically meaningful entities, allowing us to attach intuitive linguistic labels such as *low*, *medium*, and *high*.

Due to the nature of the core, there are still important interfacing issues to be dealt with. During the two-phase development scheme, we deal with differing types of information granules; while we are always processing information at the level of semantic set membership across each variable's own universe of discourse, we need to consider both binary and fuzzy abstractions. Therefore, we require an interface that allows easy transitions from two to multiple-valued logic during data granulation, and vice-versa. For this task, triangular membership functions appear to be quite suitable, offering both flexibility and simplicity; note that we keep them at $\frac{1}{2}$ overlap, as is the norm seen in the literature [12]. Figure 3.3 shows how they are able to translate smoothly into traditional binary sets. With a numeric value $v$ existing in a particular continuous variable's universe of discourse, we calculate a series of fuzzy memberships to fuzzy sets existing in the frame of cognition of the variable (fuzzy encoding). For a binary set-based abstraction, we first determine the fuzzy set from which $v$ claims the highest membership. The encoding of $v$ is given full membership to the induced binary representation of this "winning" fuzzy set and zero membership to the rest, realizing a discrete, binary representation.

For error-free reconstruction of all possible numeric output values in the data, purely triangular membership functions used for encoding, i.e. there are no trapezoidal shapes on the edges of the output space. This results in a slight variation in the interval sizes, however the binary abstraction is performed as above for numeric target values coming from the training data. For computing a numeric output from the model, we use the centre of gravity (COG) decoding (defuzzification) scheme:

$$\frac{\sum_{k=1}^{m'} Y_k S_k}{\sum_{k=1}^{m'} Y_k}$$

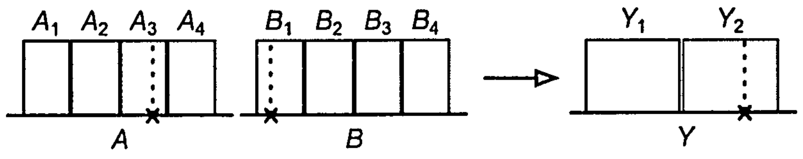


Figure 3.4: An example of binarization of information granules for obtaining binary membership from data from a two-input, one-output continuous system.

where $S_k$ are the modal values of the membership functions on the output variable's universe of discourse.

### 3.4.2 Binarization: From Continuous to Two-valued Logic Representation

In order to take advantage of logic minimization tools for structural learning, we must build a binary truth table out of the abstracted system data. As mentioned above, here the values for each variable (input and target output) are represented as simple binary memberships to a few semantically meaningful sets distributed over their universes of discourse; recall that these sets are mutually exclusive, so a value would have full membership to just one set in the space, with zero membership to the rest. See Figure 3.4 for an illustration of this, where we have four induced binary sets defined for each input variable, and two for the output variable, presenting a single input-output training instance for some continuous system. Table 3.1 shows the membership values obtained.

#### Binary Encoding

We first consider a binary coding approach. With each set in a variables space numbered, it may be converted to its binary equivalent; for example, a variable broken up into two granules can be coded with one binary input. Note that we are confined to powers of two for the number of sets per variable. With this in mind we would limit ourselves to two, four, or eight sets, noting a well-known suggested upper limit of seven ± two linguistic terms for a frame of cognition [12].

Table 3.1: Set memberships obtained during binarization in Figure 3.4

| **Set** | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $B_1$ | $B_2$ | $B_3$ | $B_4$ | $Y_1$ | $Y_2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| **Mem.** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

To encode the results seen in Table 3.1 we would require four Boolean inputs $(x_1,,x_4)$ and one output $(y)$. View that each of the four sets for an input space can be represented with two bits, where '00' would denote the first set (ex. $A_1$), '01' the second, and so forth. The encoding results in the Boolean function (representing the target system) having a value of 1 for the minterm $(1,0,0,0)$. With a complete truth table defined, to gain a logic description for both $Y_1$ and $Y_2$, we can present the Boolean minimizer with both $y$ and its complement $\bar{y}$.

After encoding and subsequent minimization, we would need to decode the binary variables in order to obtain actual set-based logic descriptions. To accommodate *don't cares* within the minimization results, it would be necessary to build a three-level logic equation for proper representation. Going back to the example, after minimization its feasible to encounter a product term such as $y = x_1 \bullet \bar{x}_2 \bullet x_3$, where $x_4$ is not present, i.e. it is a *don't care*. Because of these possible *don't cares* there are situations where digits of an encoding could be either 1 or 0, resulting in a number of possible sets rather than just one, essentially forming a union of sets in the same universe. Thus, the above expression decodes into a two level product-of-sums equation, $Y_1 = A_3 \bullet (B_3 + B_4)$. The third level of logic is constructed when summing the remaining product terms to form the cover.

There are a few drawbacks to the binary coding scheme described above. The first, already mentioned, is the limitation of the number of granules (sets) that may be used in a variables universe, being two, four, or eight instead of a more flexible range of two to nine inclusive. Such a limitation is undesired, and could restrict an expert wishing to design the interface. The second is the necessary binary decoding scheme; view that with inevitable *don't cares* we are restricted to certain unions of sets in each variables domain. For instance, for $A$ in Figure 3.4, there are four possible unions if $x_1$ or $x_2$ is a *don't care*: $(0,*) = A_1 \cup A_2$, $(1,*) = A_3 \cup A_4$, $(*,0) = A_1 \cup A_3$, and $(*,1) = A_2 \cup A_4$. This can limit the logic optimization from finding appropriate set unions.

## Set-Based Encoding

A more suitable approach is a straight translation of set memberships. For the results in Table 3.1 we would directly use those membership values, resulting in an eight inputs $(x_1,\ldots,x_8)$ and two outputs $(y_1,y_2)$. While this results in more Boolean variables in comparison to binary coding, the logic optimization methodologies are quite capable of efficiently handling large Boolean systems.

View that with each set membership coded as its own Boolean variable, we encounter the issue of complements. While complements are not often considered in fuzzy models, they nevertheless retain meaning, ex. variable $A$ is *not low*. As well, similar to the binary decoding scheme mentioned above, we can derive a three-level logic description and avoid complements altogether if desired; note how one or more negations of binary sets in a universe can be equivalently represented as a union of the remaining sets; from Figure 3.4, notice how the negation of $A_1(\bar{A}_1)$ is equivalent to $A_2 \cup A_3 \cup A_4$. Here the logic optimization algorithms are not limited as with binary coding in expressing set unions to form new, broader sets when the nature of the data shows these relationships to be pertinent.

Consider these possible logic representations, i.e. three levels with no complements, or two levels with complements. In the interest of conciseness and knowledge interpretability, it is advantageous to conceive a hybrid approach in describing states of system variables, where we use both complements and three levels of logic for decoding. From the Figure 7 example, suppose we encountered the product term $y_1 = \bar{x}_1 \bullet \bar{x}_2 \bullet \bar{x}_8$. Using this hybrid-decoding scheme, we end up with $Y_1 = (A_3 + A_4) \bullet \bar{B}_4$. View that it is more suitable to state "$A_3$ or $A_4$" rather than "not $A_1$ and not $A_2$", and that "not $B_4$" is a more concise statement than "$B_1$ or $B_2$ or $B_3$". Interestingly, with the freedom to express these set unions and complements, the logic minimization is effectively realizing a simple, dynamical form of fuzzy interface optimization.

## 3.4.3 Deriving Knowledge-Based Neural Networks

With the minimization of the binary training data, a sum (union) of product terms (rules) for each output set will make up the result. At this point we essentially have several ($m'$) collections of crisp rules describing the concepts captured within each output set. With AND and OR fuzzy neurons we are able to derive a complete knowledge-based neural architecture from this structurally optimized core. The hybrid idea of a three-level logic description with complements (negations) can directly translate into a three-layer network, as seen in Figure 3.5 for a single fuzzy output granule. The heterogeneousness of the architecture can translate directly into the description:

($A_1$ or $A_4$) and (not $B_1$) and $C_2$ or
$A_3$ and (not $C_4$) or
($B_3$ or $B_4$) and $C_1$

Note how product terms are now processed with AND neurons, with the aggregative sum handled by an OR neuron. Also, note that we fix the connections of the first hidden layer of OR neurons. The inputs to these elements are not weighted because they only deal with fuzzy sets existing in the same universe of discourse, and hence can be viewed as a union of fuzzy sets to create a new, more general membership function for a particular continuous variable. The degree of membership to this new fuzzy set is weighted like any other input when fed into the hidden AND layer, which deals with processing the fuzzy relations formed from

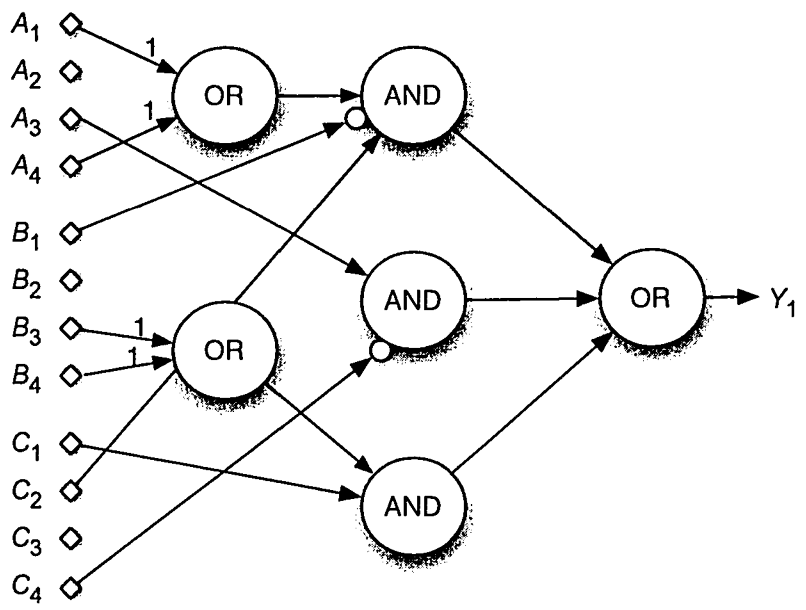


Figure 3.5: Three-layered logic processing topology, forming the core of the fuzzy model. The small circles denote negations.

separate variable spaces. In this way, the two-level topology shown in both Figure 2.3 and Figure 3.2 is preserved.

## 3.5 The Importance of Structural Learning

Thus far, the fuzzy modelling approach proposed in this study has been shown have much promise in achieving a good balance between learning accuracy and knowledge interpretability. Recognizing the importance of structural optimization is paramount here, as training the fuzzy neural network in a fully-connected, unstructured state fails in achieving both accuracy and transparency. To validate this, an attempt was made to train such architectures with the Boston housing data, taken from the UCI repository of machine learning databases [2]. Further detailed during the case study presented later in the paper, this 13-input dataset concerns median housing prices in the Boston area. For this experiment we perform interfacing as detailed in section 3.4.1, with three fuzzy sets for each input yielding $n' = 12*3 = 36$ inputs plus one binary input, for a total of 37 inputs. For the output space (median housing prices), two fuzzy sets are used, i.e. $m' = 2$.

Two methods of parametric optimization were investigated. First, the gradient-

[2]http://www.ics.uci.edu/ mlearn/MLRepository.html

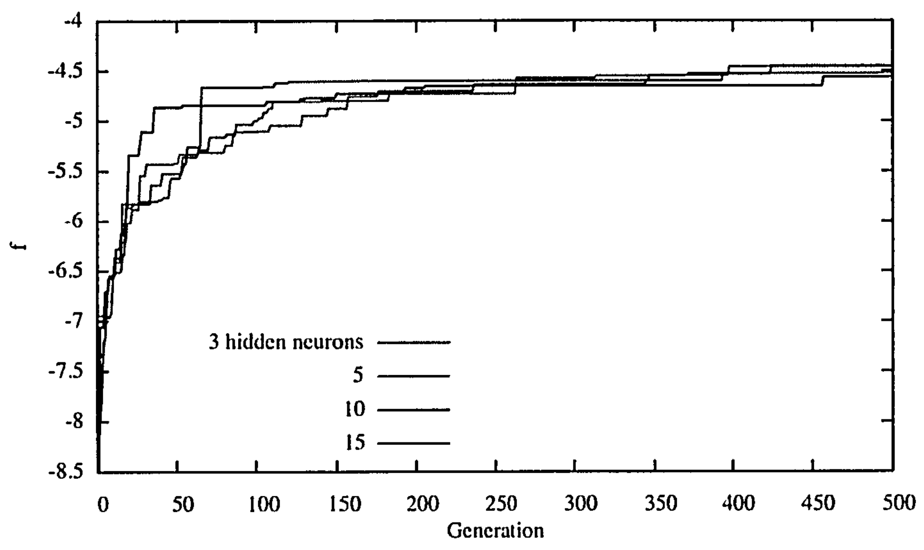


Figure 3.6: Typical progress of the GA for various FNN configurations being trained with the Boston housing data.

based method outlined in the previous section was employed, using three AND neurons in the hidden layer and two OR neurons in the output (one for each output fuzzy set), with the learning rate set to 0.01. This attempt, run for 1200 epochs, failed to achieve any level of optimization, yielding no reduction of error. The second method made use of evolutionary means via a genetic algorithm (GA). The chromosome structure consists of all connections in the network, realizing a real-coded GA with gene alleles in the unit interval. Standard operators were used for the GA [10], with parameter values determined to be suitable through preliminary experimentation: a tournament based selection with an elitist mechanism (the best individual is always carried into the next generation); a standard mutation operator at a rate 0.06; a multi-point crossover operator (with the number of points varying randomly between pairs of individuals) at a rate of 0.7; a fitness function $f$, taken as the negative of $V$ defined previously, making the highest possible fitness at 0, $f = -V$. In a typical fashion, all experiments were conducted using a population of 200 individuals, running for a maximum of 500 generations. These values were found experimentally to be justifiable. Figure 3.6 and Table 3.2 show the results of GA optimization for varying hidden layer sizes, carried out on a G4 1.33GHz PowerPC CPU. As one can see, optimization ensues, however reaching an unremarkable level of accuracy. Additionally, the amount of computation needed is significant, steadily increasing as more neurons (rules) are added.

It must be stressed here that any method of optimizing the weights of a fully-connected fuzzy neural network does not support its inherent interpretability, in

Table 3.2: Testing performance and CPU times for GA-trained networks

| *No. Hidden Neurons* | *V (testing set)* | *CPU time(seconds)* |
|---|---|---|
| **3** | 4.93 ± 0.39 | 343 ± 7.7 |
| **5** | 4.80 ± 0.43 | 559 ± 18 |
| **10** | 4.92 ± 0.38 | 1100 ± 35 |
| **15** | 4.81 ± 0.32 | 1654 ± 29 |

fact going against it. With all fuzzy set activations from all input variables fed into each and every hidden AND neuron, the learning algorithm has an enormous task of sorting through them in order to arrive at some sort of solution. This attempt is also a blind one, having to figure out for itself that it simply doesn't work for an AND neuron to, for instance, accept inputs from both *low* and *high* information granules of the same variable. Further, the small number of rules that are able to be discovered is quite limiting, with computation times steadily increasing as more AND neurons are added. Compare these evident problems with our approach in structural discovery. Employing methods of logic minimization ensures that the logic-processing nature of the model is fully utilized. We are also not hindered by attempting to select the most suitable number of rules to describe the target systems behaviour: this is accomplished for us automatically.

Mentioned earlier, there have been numerous synergistic approaches to fuzzy modelling employing evolutionary methods, with some geared toward structural optimization, cf. [1, 13–16]. However, their success has been limited, and rely heavily on computationally intensive methods. As well, they often have to perform numerous experiments (on a case-by-case basis) to determine suitable values for critical learning parameters, such as the number of rules or the maximum number of rule arguments. The approach proposed here is not only extremely computationally efficient, it is also quite natural in freely discovering an optimal or near-optimal structure (rule-base) with no imposed restrictions. Rather than using a general purpose search tool, the logic minimization techniques are more intimate with the data, understanding its logic nature as it strips away redundant information to reveal a concise behavioural description.

## 3.6 Experimental Studies

Experiments were carried out with creation of a multi-platform software package, *Third Eye* (Appendix A), forming a comprehensive fuzzy model development environment encompassing the framework introduced in this chapter.

Proceeding with experimental studies, we consider both synthetic and real-world data. In the former, we provide a low dimensional system for meaningful visualization of its behaviour and validation of the knowledge learnt by the model. The latter provides a comprehensive case study concerning a real-world system detailing housing prices in the Boston area (the Boston housing dataset). We also briefly report on results achieved for several other datasets. Regarding the execution times given for various modelling tasks, all experiments were run on a G4 1.33GHz PowerPC-based system.

Regardless of data, the underlying goal of the modelling is to realize a knowledge based mapping that intuitively describes the behaviour (output) of an $n$-input system, i.e. $\Re^n \mapsto \Re$. This is completed through the construction and selective aggregation of information granules, having $n'$ fuzzy sets for $n$ inputs and $m'$ fuzzy sets for the output; within the granular interfaces of the model, the mapping takes the form $[0,1]^{n'} \mapsto [0,1]^{m'}$. It is here where the learning takes place. With regards to model development, there were no options to be set for structure discovery: all that the logic minimizer requires is a binary truth table representing the granulated training data. As for the parametric learning, in all experiments the learning rate, $\alpha$, was set to 0.01, with the algorithm running for 1200 epochs. These typical values were found experimentally to be justifiable. As well, note that we consider t/s-norms to be a product operation and probabilistic sum, respectively.

When discretizing a continuous system into a small number of binary sets, it is difficult to avoid conflicts in the data, i.e. having two or more identical input patterns showing inconsistent output patterns. To circumvent this, an evolutionary method of determining optimal interval widths was investigated for eliminating conflicts and forming a blueprint for fuzzy/binary set distribution. However, it was not accepted because it can easily destroy the semantic integrity of set labels when largely varying intervals are produced. As well, any improvement in conflict reduction did not warrant the additional computation. For equal width intervals, the amount of conflicting data appears to be reasonable in many situations; refer to Table 3.3 for a quantitative analysis of conflicts appearing within the Boston housing data at varying levels of granularity for input and output interfaces. Note the low amounts of conflicting data, as well as the trend of decreasing conflicts as output granularity decreases and input granularity increases, an expected result. As a general observation, note that the actual number of omitted data points will decrease when training with only sub-sets of the data, and also with increasing system dimensionality we see longer and more complex input patterns, often resulting in less conflicts. It is also important to stress that this data is not lost, as we may re-introduce any previously conflicting data during parametric learning, where we deal with fuzzy information granules rather than binary. We may envision that these data points could be more difficult to learn.

As a final note, there are times when we may want to distinguish between various versions of the model during its development. We refer to the logically

Table 3.3: Percentages of data points removed from the Boston housing dataset due to discretization conflicts

| | *Output sets* | | | | | |
|---|---|---|---|---|---|---|
| ***Input sets*** | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 13.6 | 12.8 | 18.2 | 27.1 | 29.6 | 34.2 |
| 3 | 5.3 | 9.5 | 12.5 | 18.2 | 17.8 | 24.7 |
| 4 | 4.0 | 5.7 | 6.3 | 9.3 | 11.9 | 13.8 |
| 5 | 0.6 | 3.2 | 3.2 | 6.7 | 6.1 | 8.5 |
| 6 | 0.4 | 3.4 | 2.2 | 2.8 | 4.2 | 6.1 |
| 7 | 0.4 | 2.2 | 1.8 | 2.6 | 3.8 | 4.7 |

optimized model with binary information granules as the binary model; upon introducing fuzzy granules and t/s-norm aggregation to this underlying structure, we call this the fuzzy model; after parameterizing the fuzzy model with the use of fuzzy neurons, we refer to it as the neurofuzzy model.

### 3.6.1 Synthetic Data

We start with synthetic data, as shown in Figure 3.7. Here we have single input, single output system, where we have intentionally set the dimensionality low so we may see a clear visual representation of its behaviour. As an "expert" attempting to model the system, one can infer that three equally distributed fuzzy sets could be suitable for both input and output, where we can assign the labels *low*, *medium*, and *high*. With these information granules defined we would expect the knowledge base to take the form of three rules:

If $x$ is *low* then $y$ is *medium*
If $x$ is *medium* then $y$ is *low*
If $x$ is *high* then $y$ is *high*

A dataset of 50 data points was created through uniform sampling of the synthetic, continuous system. A random selection of 30 points (60%) for training was completed ten times for a ten-fold cross-validation, with the remaining data used for testing.

Using our proposed methodology, we attempt to automatically construct an accurate and transparent model of this synthetic data. Since we have just one input to the system ($n$=1), we must realize a mapping $\Re \mapsto \Re$. As a first step, three uniformly distributed fuzzy sets are constructed over input and output in their respective universes of discourse, and through this interface we focus on the internal mapping $[0,1]^3 \mapsto [0,1]^3$. Binary representations of these information

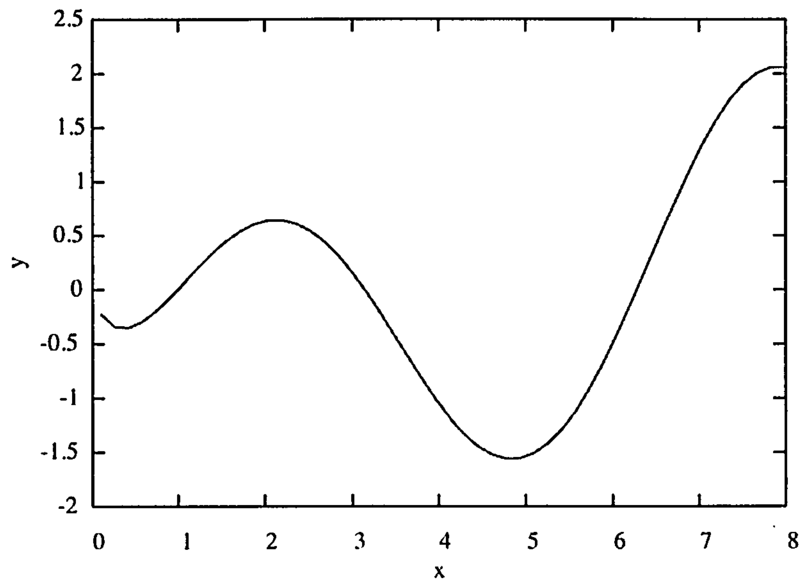


Figure 3.7: Synthetic dataset with one input ($x$) and one output ($y$)

granules are induced, with the approximation task temporarily assuming the form $\{0,1\}^3 \mapsto \{0,1\}^3$. The training data is passed through this binary interface, and any discretization conflicts are detected; these occur when the binary input patterns (vectors) of two or more data points are equivalent, i.e. $\mathbf{X}(c) = \mathbf{X}(c')$, while their corresponding binary output vectors are not, $\mathbf{Y}(c) \neq \mathbf{Y}(c')$. When considering a number of these conflicts, the data points with output patterns occurring least often are removed. During this binarization step, the number of conflicting data points removed averaged at $6.9 \pm 1.8$.

After cleansing the binary data, each information granule is encoded as its own Boolean variable and a decision (truth) table is formed. This marks the first phase of creation and optimization of the processing core of our model, where a logic-based representation of the underlying structure of the system is discovered through logic minimization, namely ESPRESSO-II. In all ten experiments, the learnt structures were identical to each other, matching the exact representation as we, the "experts", detailed previously. Using this blueprint, a parameter-free fuzzy neural network is constructed, where binary AND and OR operations are replaced with AND and OR fuzzy neurons. At this point the neurons are essentially functioning as fuzzy AND and OR gates realized by t/s-norm aggregation. With structural optimization complete, refer to Table 3.4 for the performance results (with all data including previously removed conflicts) when using either binary or fuzzy information granules for the interfacing (recall that t/s-norm operations function as Boolean AND and OR with two-valued data). Notice how the accuracy

Table 3.4: Performance results (expressed in terms of $V$) for the synthetic data after structural learning

| . | *Sets* | *Fuzzy sets* |
|---|---|---|
| ***Training*** | $1.1 \pm 0.20$ | $0.9 \pm 0.07$ |
| ***Testing*** | $1.3 \pm 0.25$ | $1.1 \pm 0.09$ |

improves with the fuzzy sets, along with less variation between cross-validation iterations.

Through gradient-based parametric optimization of the fuzzy neural network we can achieve higher levels of accuracy and gain further insight into the behaviour of the system: so begins phase two. With three output fuzzy sets describing the system's behaviour, each is trained individually to learn their respective concepts, that is *low*, *medium*, and *high* values of $y$; the progress of training can be seen in Figure 3.8. The results of learning showed further improvements in performance, with $V$ equating to $0.7 \pm 0.04$ for training and $0.8 \pm 0.07$ for testing. Of the ten iterations, the interpretation for the network leading to the best testing performance is shown (note that all networks were quite similar to each other):

If $x$ is *low*(0.07) then $y$ is *medium*(0.77)
If $x$ is *medium*(0.42) then $y$ is *low*(0.92)
If $x$ is *high*(0.03) then $y$ is *high*(0.42)

One of the most evident changes here is in the confidence level of the last rule. View how this tends to correspond with the behaviour shown in Figure 3.7; we see that $y$ is high for only about 20% of the range of $x$, rather than an even split expected by the parameter-free rules.

For a visual indication of these performance results, refer to Figure 3.9, where we show the approximated curve produced by each version of the model: binary, fuzzy, and neurofuzzy. View how the fit becomes increasingly better with each version of the model. One also notices the impact seen from granulation of such a low-dimensional system: although sufficient for describing general behaviour, three fuzzy sets are perhaps not enough for truly accurate representation. In the interest of increasing this accuracy even further, we performed the same experiment with six sets for input and output. The performance results of the neurofuzzy model were $0.29 \pm 0.1$ and $0.32 \pm 0.09$ for testing, a significant improvement as seen in Figure 3.9.

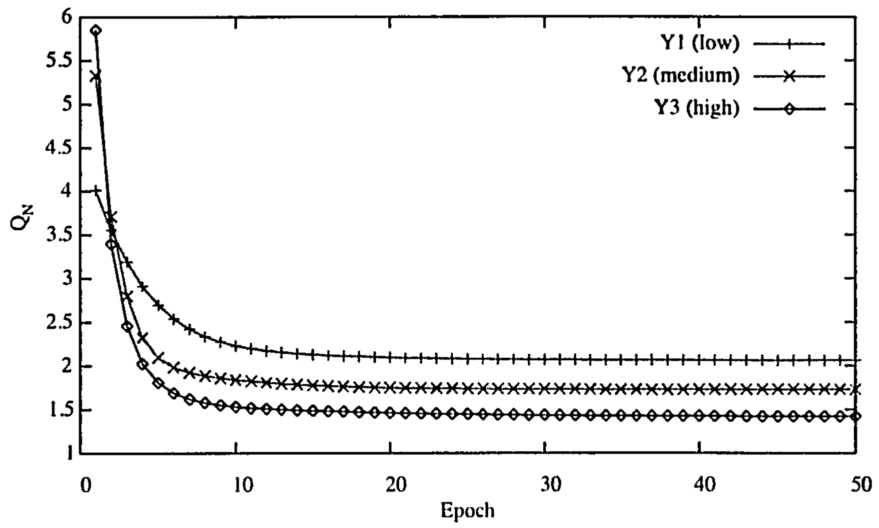


Figure 3.8: Parametric training progress of each fuzzy output granule for the synthetic data. No visible improvements seen beyond 50 epochs.

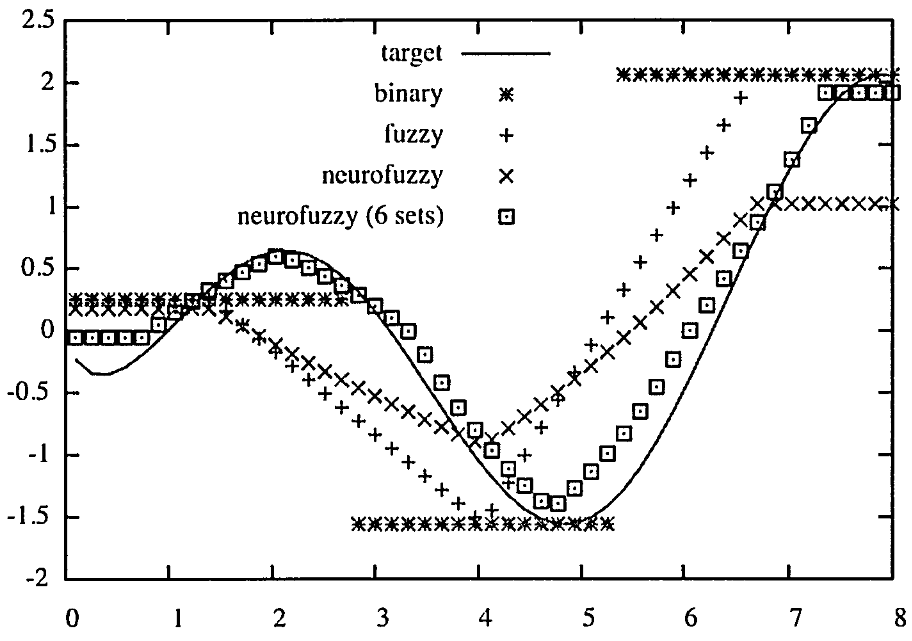


Figure 3.9: Behaviour of each model in comparison with target system

Table 3.5: Attributes of the Boston Housing Data

| | |
|---|---|
| CRIM | per capita crime rate by town |
| ZN | proportion residential land zoned for lots over 25,000$ft^2$ |
| INDUS | proportion of non-retail business acres per town |
| CHAS | Charles River dummy variable (1 if tract bounds river; 0 otherwise) |
| NOX | nitric oxides concentration (parts per 10 million) |
| RM | average number of rooms per dwelling |
| AGE | proportion of owner-occupied units built prior to 1940 |
| DIS | weighted distances to five Boston employment centres |
| RAD | index of accessibility to radial highways |
| TAX | full value property-tax rate per $10,000 |
| PTRATIO | pupil-teacher ratio by town |
| B | $1000(B_k - 0.63)^2$, $B_k$ is the proportion of African-Americans by town |
| LSTAT | % lower status of the population |
| MEDV | Median value of owner-occupied homes in $1000s |

### 3.6.2 Boston Housing Data

Here we conduct a case study on the Boston housing dataset, concerning a description of real estate in the Boston area where housing is characterized by a number of features including crime rate, number of rooms, age of houses, etc. and the median price of houses. The dataset consists of 506 14-dimensional points, each representing a single attribute (see Table 3.5). The construction of the fuzzy model is completed for 304 data points (60%) treated as a training set, again using ten-fold cross-validation.

The goal is to create a model with three outputs, each representing an information granule defined on the output space, i.e. the median housing price. These outputs take the labels of *low*, *medium*, and *high*. We use three granules for each continuous input, assigning these same linguistic labels. Note that the choice of these numbers was arbitrary, seeming like a reasonable number of information granules to use for generating rules and interpreting the knowledge to be gained. The binary variable CHAS was simply taken as-is. Overall, we have a total of 37 inputs.

Proceeding with the first phase of model development, during binarization the amount of conflicting data was fairly small, amounting to $26.3 \pm 2.1$ data points: on average less than 9% of the training set. The discovered structures showed consistency between cross-validation iterations, with the number of rules (product terms) equalling $22.0 \pm 2.4$, and the number of literals at $81.1 \pm 11$. Note how

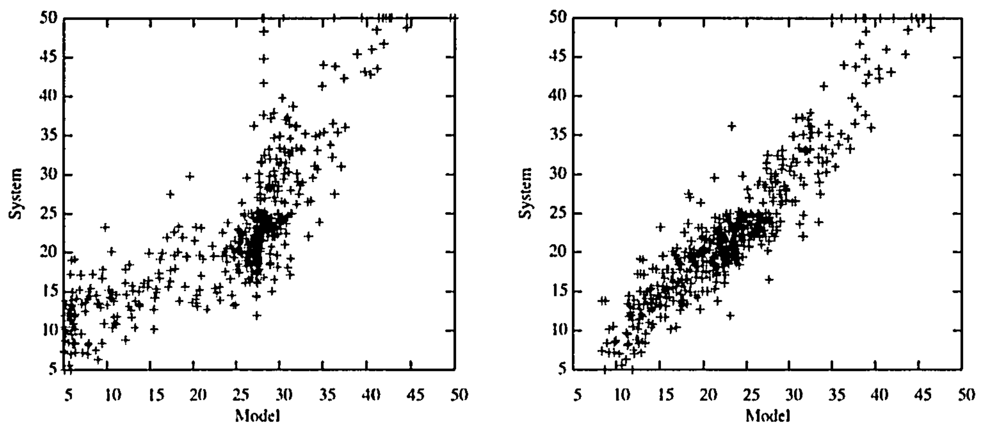
Table 3.6: Performance results for the Boston housing data after structural learning

| | *Sets* | *Fuzzy sets* |
|---|---|---|
| ***Training*** | 7.72 ± 0.22 | 5.95 ± 0.49 |
| ***Testing*** | 9.24 ± 0.67 | 6.38 ± 0.56 |

the data is fairly complex, requiring a large number of non-parameterized rules to describe its behavioural structure. The performance results of structural learning are shown in Table 3.6, for both binary and fuzzy interfaces. Here we see excellent performance gains with the triangular fuzzy sets. As well, notice the improved performance from training to testing, showing the high level of robustness of the fuzzy interface in achieving effective generalization.

Proceeding with parametric optimization, the results were 3.76 ± 0.38 for training and 4.08 ± 0.37 for testing, both significant improvements over the parameter-free fuzzy model. To view these improvements visually, refer to the scatter plots presented in Figure 3.10. In addition to the performance gain, the neural augmentation to the model was able to reduce the structural complexity considerably. The overall best performing model (determined as the highest average performance on training and testing data) out of the ten training instances was pruned using the thresholding process detailed in Section 3.3.2, with performance-optimal thresholds of 0.5 for OR neurons and 0.05 for AND neurons. As a result, the size of the rule-base was simplified from 21 rules with 75 literals, to 8 rules with 15 literals. The interpretation of this learnt knowledge knowledge can be found in Table 3.7, showing a concise logic description. Note how the rules quite intuitive; for instance, for some high price houses, it makes sense that the average number of rooms is *high* and the property tax is *not high*. Another example is how the proportion of lower status of the population being not low (i.e. *medium* or *high*) results in low housing prices. Although the resultant structure of each of the ten networks were not exactly the same due to the different training data, there were many similarities and common rules among them. We may view the more common rules as occurring more prominently within the experimental data.

Finally, the computational efficiency of the model development should be emphasized; for a single instance of learning the Boston housing dataset, both structural and parametric optimization were completed together in roughly ten seconds, from raw data to its logic representation.



(a) Fuzzy model (b) Neurofuzzy model

Figure 3.10: Boston housing scatter plots showing system output vs. model output for the (a) fuzzy model and (b) neurofuzzy model

Table 3.7: Collection of quantified rules derived from the Boston housing model

| *if-condition* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **CRIM** | medium (0.008) | | | | | | | |
| **NOX** | | | | high (0.000) | | | medium (0.000) | |
| **RM** | | | | | | medium (0.003) | | high (0.000) |
| **AGE** | | | | | | | high (0.000) | |
| **RAD** | | | | | | | high (0.000) | |
| **TAX** | | | | | | | | not high (0.000) |
| **PTRATIO** | | | high (0.000) | | | not high (0.002) | | |
| **B** | | | | medium (0.009) | | | | |
| **LSTAT** | | not low (0.005) | medium (0.000) | | low (0.001) | | low (0.000) | |
| *then-conclusion* | | | | | | | | |
| **MEDV** | low | low | medium | medium | medium | medium | high | high |
| *confidence* | 0.691 | 0.641 | 0.555 | 0.682 | 0.999 | 0.748 | 0.832 | 0.912 |

### 3.6.3 Further Experiments with Real-World Data

Several experiments were conducted with other so-called regression datasets found at the ML repository; again a 60%/40% split of training/testing data was used. Here the number of fuzzy sets used for input and output interfaces were independently varied from two to five per variable space, with the best performing configuration subjected to ten-fold cross-validation. Any nominal variables encountered were encoded in the form 1-out-of-n. The results are shown in Table 3.8, where we report on various performance measures. Note that the normalization is carried out by dividing $V$ over the range of possible output values for the respective system (dataset). As well, the CPU time is formed from the times for both structural and parametric learning. An interesting detail here is that in all cases, structure discovery through logic minimization amounted to 5% (or less) of total training time, with the rest of computational effort dedicated toward the gradient-based optimization. Further, note that we selected a rather conservative number of learning epochs: the majority of error reduction occurred within the first few hundred.

In general, we see excellent performance with all of the data, including very short execution times. The longest times seen were with the Abalone data, which was expected considering the size of the dataset ($N = 4177$). A point that should be made here is that for each dataset, the accuracy did not suffer to a large degree when varying the number of sets for input and output, with variations falling within reasonable ranges. This is important, meaning that the user, if he or she desires, may have some freedom in designing the granular interface without having to worry about poor accuracy. For instance, the user could want a more detailed linguistic description for the resulting model, which would require a higher number of sets in the input and/or output spaces.

In the interest of seeing the knowledge gained by the system modelling efforts, we show the derived rule-bases after pruning for each of the datasets: Table 3.9 for Abalone with pruning of 0.5 for OR neurons and 0.4 for AND neurons; Table 3.10 for Auto-Mpg, with pruning thresholds of 0.4 and 0.3; Table 3.11 for the Computer hardware data, with thresholds of 0.5 and 0.4. Since we have already conducted a detailed case study of the Boston housing data, no further interpretations are shown, although interestingly the model with two fuzzy sets in the output space performed better than three, as we used previously.

By viewing these intuitive logic descriptions of the respective data, we are able to gain insight into their nature. For instance, we see in Table 3.10 how, among other attributes, weight and the model year have a large effect on automobile mileage, where a *high* model year would indicate a newer vehicle. For describing computer hardware performance, Table 3.11 shows how important main memory and cache memory are. As well, from Table 3.9 one can infer which are the most revealing measurements for determining age levels of abalones.

Table 3.8: Modelling results for several well-known regression datasets

| *dataset* | *Abalone* | *Auto-Mpg* | *Boston Housing* | *Computer HW* |
|---|---|---|---|---|
| ***Fuzzy sets per input*** | 3 | 3 | 3 | 3 |
| ***Output fuzzy sets*** | 2 | 3 | 2 | 5 |
| ***Total # inputs*** | 24 | 23 | 37 | 48 |
| ***Conflicting data*** | 5.8% ± 0.3 | 8.5% ± 0.8 | 4.5% ± 0.8 | 2.5% ± 1.1 |
| ***CPU time (seconds)*** | 84.1 ± 24.6 | 5.20 ± 0.60 | 11.4 ± 1.8 | 1.51 ± 0.36 |
| ***Rules (unpruned)*** | 17.2 ± 3.6 | 13.6 ± 1.3 | 20.8 ± 2.4 | 7.3 ± 1.3 |
| ***Testing perf. (V)*** | 2.32 ± 0.07 | 3.07 ± 0.31 | 3.82 ± 0.40 | 68.5 ± 30.6 |
| ***V (norm.)*** | 0.0829 ± 0.0025 | 0.0814 ± 0.0082 | 0.0849 ± 0.0089 | 0.0599 ± 0.028 |

Table 3.9: Quantified logic description of the Abalone data

| *if-condition* | | | | | *then-conclusion* | *confidence* |
|---|---|---|---|---|---|---|
| **Length** | **Diameter** | **Whole weight** | **Shucked weight** | **Shell weight** | **Age** | |
| high (0.002) | | | | low (0.304) | low | 0.550 |
| not high (0.002) | | | | not med. (0.000) | low | 0.566 |
| | not high (0.004) | | medium (0.002) | | low | 0.813 |
| not high (0.286) | | medium (0.332) | not med. (0.002) | | high | 0.884 |
| | | | | high (0.010) | high | 0.694 |

Table 3.10: Quantified logic description of the Auto-mpg data

| *if-condition* | | | | *then-conclusion* | *confidence* |
|---|---|---|---|---|---|
| **Horsepower** | **Weight** | **Model year** | **Cylinders** | **Mileage** | |
| | high (0.005) | | | low | 0.703 |
| medium (0.206) | | not high (0.001) | | low | 0.457 |
| | | | not 4 (0.000) | low | 0.420 |
| low (0.235) | | | | medium | 0.420 |
| | | | 4 (0.200) | medium | 0.670 |
| | medium (0.008) | high (0.000) | | medium | 0.766 |
| | low (0.000) | high (0.092) | | high | 0.825 |

Table 3.11: Quantified logic description of the Computer hardware data

| *if-condition* | | | *then-conclusion* | *confidence* |
|---|---|---|---|---|
| **Minimum main memory** | **Maximum main memory** | **Cache memory** | **Relative performance** | |
| | not medium (0.094) | low (0.031) | low | 0.853 |
| | low (0.000) | not low (0.101) | medium-low | 0.816 |
| | medium (0.033) | | medium-low | 0.878 |
| medium (0.004) | | | medium | 0.575 |
| | high (0.000) | | high | 1.000 |

Table 3.12: Performance results for the Boston housing data using various regression algorithms and the proposed fuzzy modelling framework.

| *Learning method* | *Kernel type* | *Testing set performance* |
|---|---|---|
| Ridge regression [18] | Polynomial | 3.23 |
| Ridge regression [18] | Splines | 2.92 |
| Ridge regression [18] | ANOVA Splines | 2.77 |
| Relevance vector machine [21] | Gaussian | 2.73 |
| Support vector machine [22] | Gaussian | 3.20 |
| Regression w/ Gaussian processes [23] | Gaussian | 3.02 |
| BSVR w/ $\beta = 0.3$ [3] | Gaussian | 3.51 |
| Regression w/ Gaussian processes [23] | ARD Gaussian | 2.88 |
| BSVR w/ $\beta = 0.3$ [3] | ARD Gaussian | 2.64 |
| Logically optimized neurofuzzy model | N/A | 3.62 |

### 3.6.4 Comparison with Standard Regression Techniques

Throughout this study it has been shown how the proposed method embraces its logic-based foundations in order to provide a truly transparent and highly understandable knowledge-base. However, although the high accuracy of the approach has also been demonstrated, there has not been any direct comparisons to other frameworks. In this section, performance comparisons are made with techniques focused solely on regression. Having only the goal of accuracy, these methods are generally top performers, not worrying about knowledge discovery, instead providing a black-box solution. In [3] the authors propose a Bayesian support vector regression approach (BSVR), comparing performance on the Boston housing data with a number of other regression techniques: ridge regression [18], relevance vector machine [21], support vector machine [22], and regression with Gaussian processes [23]. The results are cited here for comparison with our model. As described in [3], the data was randomly partitioned into 481/25 training/testing splits, carried out 100 times and averaged, with the results seen in Table 3.12. Note that we cannot expect the model to surpass the performance of such algorithms making use of uninterpretable means including a plethora of non-linear elements. Nevertheless, it manages to provide comparable results. While there obviously must be some trade-off between accuracy and interpretability, this result is quite encouraging for such a heterogeneous, human-friendly knowledge-based architecture.

## 3.7 Review

In this chapter we have proposed and validated an effective and novel design methodology for fuzzy modelling. The heterogeneous development process takes into account the two fundamental requirements of granular modelling, namely accuracy of behavioural approximation and transparency of the learnt knowledge. The two key technologies used here for model development, logic minimization and fuzzy neural networks, are instrumental in achieving overall accuracy with inherent abilities to provide a completely interpretable architecture. The work has resulted in the creation of *Third Eye*, an efficient, software-based fuzzy model development environment.

## Bibliography

[1] Andreas Bastian. Identifying fuzzy models utilizing genetic programming. *Fuzzy Sets Syst.*, 113(3):333–350, 2000.

[2] Robert King Brayton, Alberto L. Sangiovanni-Vincentelli, Curtis T. McMullen, and Gary D. Hachtel. *Logic Minimization for VLSI Synthesis*. Kluwer Academic Publishers, Boston, MA, 1984.

[3] W. Chu, S. S. Keerthi, and C. J. Ong. Bayesian support vector regression using a unified loss function. *IEEE Trans. Neural Networks*, 15:29–44, January 2004.

[4] Petr Fiser, Jan Hlavicka, and Hana Kubatova. FC-Min: A fast multi-output Boolean minimizer. In *Proc. of the Euromicro Symposium on Digital System Design (DSD'03)*, pages 451–454, 2003.

[5] Jan Hlavicka and Petr Fiser. BOOM - a heuristic Boolean minimizer. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design (ICCAD 2001)*, pages 439–442, 2001.

[6] Roman Lysecky and Frank Vahid. A codesigned on-chip logic minimizer. In *Proc. of the First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 109–113, 2003.

[7] Roman Lysecky and Frank Vahid. On-chip logic minimizer. In *Proc. of the Design Automation Conference*, pages 334–337, 2003.

[8] E. J. McCluskey. Minimization of boolean functions. *Bell Syst. Tech. Jour.*, 35:1417–1444, April 1956.

[9] Patrick C. McGeer, Jagesh V. Sanghavi, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. ESPRESSO-SIGNATURE: A new exact minimizer for logic functions. *IEEE Trans. VLSI Syst.*, 1:432–440, December 1993.

[10] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1998.

[11] Witold Pedrycz. *Fuzzy Sets Engineering*. CRC Press, Boca Raton, FL, 1995.

[12] Witold Pedrycz and Fernando Gomide. *An Introduction to Fuzzy Sets: Analysis and Design*. MIT Press, Cambridge, MA, 1998.

[13] Witold Pedrycz and Marek Reformat. Evolutionary fuzzy modeling. *IEEE Trans. Fuzzy Syst.*, 11:652–665, 2003.

[14] Witold Pedrycz and Marek Reformat. Genetically optimized logic models. *Fuzzy Sets Syst.*, 150:351–371, 2005.

[15] Marco Russo. FuGeNeSys - a fuzzy genetic neural system for fuzzy modeling. *IEEE Trans. Fuzzy Syst.*, 6:373–388, 1998.

[16] Marco Russo. Genetic fuzzy learning. *IEEE Trans. Fuzzy Syst.*, 4:259–273, 2000.

[17] Samir Sapra, Michael Theobald, and Edmund Clarke. SAT-based algorithms for logic minimization. In *Proc. of the 21st International Conference on Computer Design (ICCD'03)*, pages 510–517, 2003.

[18] C. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *Proc. of the 15th Int. Conference on Machine Learning*, pages 515–521, 1998.

[19] Winfrid G. Schneeweiss. *Boolean Functions with Engineering Applications and Computer Programs*. Springer-Verlag New York, Inc., New York, NY, 1989.

[20] Michael Theobald and Steven M. Norwick. Fast heuristic and exact algorithms for two-level hazard-free logic minimization. *IEEE Trans. Computer-Aided Design*, 17:1130–1147, November 1998.

[21] M. E. Tipping. The relevance vector machine. In *Advances in Neural Information Processing Systems 12*, Cambridge, MA, 2000. MIT Press.

[22] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, NY, 1995.

[23] C. K. I. Williams. Prediction with gaussian processes: from linear regression to linear prediction and beyond. In *Learning in Graphical Models*, Norwell, MA, 1998. Kluwer.

no text

# Chapter 4

# Fuzzy Neural Networks for Intelligent Hardware Engines

The adaptive fuzzy modelling framework presented in the previous chapter exhibits excellent potential for driving intelligent systems that must operate in dynamic and rapidly changing environments. However, to fully exploit the potential of the underlying fuzzy neural networks and their parallel nature, efficient hardware implementations are highly desired. In this chapter, our objective is to investigate this avenue and identify various critical design issues as we propose a versatile neuro-fuzzy platform with a topology strongly influenced by theories of fuzzy modelling. With the novel hybrid-learning scheme presented in Chapter 3, we demonstrate how fuzzy neural networks are well suited in forming the adaptive logic-processing core of this platform, supporting intelligent information processing. Emulating aspects of human thought and using logic-oriented reasoning to solve a problem, an entity based upon this platform would be able to learn and approximate real-world concepts, building a knowledge base that may be interpreted and modified by the user. Drawing upon this knowledge, a hardware implementation has potential for performing inference of many simultaneous concepts in real-time, realizing cognition as it perceives the current state of its environment.

## 4.1 Overview

With the growing need for the deployment of intelligent, highly autonomous systems, it would be beneficial for them to seamlessly combine robust learning capabilities with a high level of knowledge interpretability. With the fuzzy modelling framework presented in Chapter 3 we are able to achieve this powerful combination. Realized in the form of knowledge-based fuzzy neural networks, they are able to build comprehensive knowledge-bases representing meaningful real-world concepts. Given input data, a network is able to use its knowledge to infer the relevance or degree of truth of these concepts in relation to the current state of its environment. With their adaptive nature, this knowledge can be gained autonomously through learning by example; as well, the logic-based architectures realized by these computationally intelligent entities facilitates communication between themselves and humans, where their learnt structures may be modified or augmented by the user if desired.

To fully exploit the potential of fuzzy neural networks and their parallel nature, an efficient hardware implementation is highly desirable. Perhaps the main motivation for such a platform is the sheer amount knowledge-based processing that can be efficiently accomplished. Consider a sensor-rich system with real-time constraints. Using fuzzy neural networks to realize the logic-processing core of an adaptive fuzzy modelling system, we can build a comprehensive knowledge base of many different concepts and states that it must detect (infer) and act upon. This could lead to an architecture composed of hundreds, perhaps thousands of fuzzy neurons that may be executed in parallel. An example would be an autonomous underwater vehicle with a low-level sensor array consisting of elements such as sonar readings and video. We can abstract this raw data into information granules and use FNNs to learn the environment, identifying and determining the size of schools of fish, perceiving the shape of underwater terrain, distinguishing between plant species, detecting obstacles, etc. In a real-world situation, the vehicle may need to infer whether all or many of these concepts are relevant, repeatedly gathering sensor information to be processed through the knowledge base; with such a parallel architecture this is feasible in real time. Upon understanding the state of its environment, a higher-level, user-defined behavioural rule-base may be executed, essentially realizing autonomous behaviour with user instruction.

Along with an overview of a general topology of our proposed neurofuzzy hardware architecture and a review of current fuzzy and neural hardware technologies, we identify and analyze key issues involved in the design of FNN-based intelligent hardware engines. In particular, we attempt to address the following:

- Presentation of a novel configurable fuzzy neuron design in hardware, along with identification of critical design issues.
- Proposal of a specialized, hardware-focused hybrid-learning framework for adaptive logic processing, where we consider both structural and parametric optimization as separate mechanisms for on-chip implementation.
- Boolean network development showing effects input dimensionality can have on modern optimization methods such as genetic algorithms (GA).
- Extensive experimentation with real-world continuous data in order to provide a quantitative analysis of the identified design issues.

Note that throughout this chapter, we frequently refer to the granularity of neural connections and fuzzy set membership, which is analogous to the idea of discretization (granulation) of continuous data in digital systems. In addition to the novelty of our proposed hardware architecture, we intend to show that high-precision within the unit interval is unnecessary if employing fuzzy neural networks as a logic-processing core; when using a low-granularity representation of the unit interval, learning and representation need not be compromised.

### 4.1.1 Fuzzy and Neural Hardware Technology

While more concerned with current technologies of hardware-based neural networks, a brief study was conducted on recent efforts in fuzzy hardware realization. It was found that most of the research was geared towards implementation of fuzzy set operations for fuzzy inference engines and fuzzy controllers. Solutions include enhanced microcontrollers with fuzzy instructions and algorithmic tuning [6, 30], largely parallel, parameterized hardware [7, 13], or some combination of the two, such as microprocessors with separate fuzzy coprocessing [27]. In general, development of these systems seems to be focused towards digital systems, although there has been some work done in the analog domain [9, 28].

Current neural network hardware technological advancements were more closely examined, and it was found that while there was a broad range of implementation techniques, the majority of efforts could be classified as an analog [4, 8, 17], digital [10, 23, 29, 31, 32], or hybrid (mixed-signal) [12, 20, 28] solution. Analog systems are preferred for large productions, very low power, very high sample rate or bandwidth, and small size. However, connection storage is usually volatile, circuit behaviour is far from ideal, and the platforms are often specialized for specific architectures only, making development of these systems a difficult and normally very application-specific. Hybrid systems avoid many of these problems, but still remain somewhat difficult to work with during development. Digital systems, although they are subject to discretization and higher power consumption and circuit size, are preferred for higher accuracy, high repeatability, low noise sensitivity, better testability, higher flexibility, and compatibility with other types of preprocessing. Digital systems can also be designed more easily, thanks to the improvements in computer-aided design tools.

In particular, field-programmable gate arrays (FPGA) provide an excellent general purpose development platform for digital systems. FPGAs are digital programmable devices that can be configured (programmed) to realize virtually any digital system, with accessibility, re-programmability, and low costs permitting quick and non-expensive implementations. An FPGA platform supports development of fast, compact solutions, providing powerful integration of hardware design with the software-programming paradigm. Specifically, this integration is made possible with the use of a hardware description language (HDL) such as Very High Speed Integrated Circuits HDL (VHDL) [2] or Verilog [1], which also allows us to create designs that may be migrated to other platforms such as different FPGA chips or even application-specific integrated circuits (ASIC).

The research is concerned with developing a highly configurable system that appeals to a wide range of application areas. With VHDL and an FPGA device we have an excellent opportunity to explore fuzzy neural network architectures with the power of hardware in the familiarity of a software-programming environment. In fact, nearly all research in digital hardware based neural networks takes advantage of FPGAs, as seen in [10, 16, 21, 22, 31, 32]. Another prospect with creating FNN designs on an FPGA platform involves its dynamic reconfiguration abilities; one can envision a system that is able to physically reconfigure itself in order to adapt to a changing environment, essentially evolving into a better solution. As well, in conjunction with FGPA development, it could be beneficial to consider fast analog computational modules such as multipliers or Lukasiewicz logic arrays [18], which consume less power and are of smaller size than their digital counterparts.

## 4.2 FNN Hardware Design

In representing, learning, and inferring real-world concepts, we are essentially modelling their behaviour, and hence the task is analogous to the identification and optimization of fuzzy models.

It is the fuzzy modelling topology shown in Figure 2.2 that we are adopting for our neurofuzzy hardware platform. In this study, we are concerned with the processing core, as it is the most difficult component to implement efficiently. This type of logic-processing is what fuzzy neural networks were designed for, and we intend to realize their structures as highly configurable elements forming intelligent hardware engines.

### 4.2.1 A Granulated Core

It is important to note that fuzzy neuroprocessing is fundamentally based upon fuzzy logic, which deals with continuous values on the unit interval. To realize this theory in the setting of digital computing, whether we are in a high-level environment (software), or a low-level one (hardware), we need to apply some level of granulation (discretization) to the unit interval. That is, we must map the continuous variable into discrete values on [0,1]. As the level of discretization increases, intervals become broader, and the idea of granular knowledge-based neural networks becomes clearer.

Conceptually, the processing core of a fuzzy model deals with continuous truth-values indicating membership to fuzzy sets defining various perspectives of the data. When augmenting this core with fuzzy neurons, further continuous logic is introduced with the neurofuzzy connections. Naturally, applying granulation to this architecture would result in a loss of information, as it would no longer be able to assume any value on the continuous unit interval. This raises questions of the need of precision, and how it affects a networks ability to learn. On the positive side, dealing with these larger granules could easily result in significantly less computation and resources (ex. memory for storing connections), an important consideration when dealing with low-level digital hardware. We express this level of granularity applied to the processing core as $g$, measured in bits. For example, two bits of granularity would indicate four distinct levels of truth.

### 4.2.2 Fuzzy Neurons in Programmable Logic

For the creation of hardware-based fuzzy neurons, we take advantage of computer-aided digital design tools. In particular, VHDL is a well-known hardware description language that allows the creation of digital systems with the ease and familiarity of a software-programming environment. Through VHDL-based tools, digital hardware can be designed and simulated on a standard PC. This design can then be synthesized and programmed onto a physical device such as an FPGA chip or ASIC.

With VHDL and programmable logic we are able to implement highly configurable hardware-based fuzzy neurons. There is much freedom in their creation, with no need to deal with any sort of fixed specifications; data and control paths may have variable widths, and designs can be easily modularized. Granularity can be set anywhere from one bit to as large as the target platform can accommodate, largely depending upon the size of the knowledge base. In fact, the entire processing core can be dynamically reconfigurable, considering parameters such as granularity, t/s-norm selection, neural connections, and the underlying structure.

Aggregative AND or OR neurons are composed of t/s-norm operators; using VHDL, we can describe asynchronous parts that perform these operations, accepting two inputs and producing an output. From these basic components we can then build AND and OR neurons as synchronous circuits. As an example, Figure 4.1. shows the hardware architecture for a 4-input OR neuron. View that we can clearly separate the structure into pipelined stages to provide high levels of speed and efficiency, with larger numbers of inputs simply requiring more pipe stages. Further, pipelining is also inherent between layers of a neural network. These realizations lead us to a conclusion that the overall clock rate of a fuzzy neural network can be dependent on the largest critical logic path of the most complex triangular norm. Preliminary testing with FPGA synthesis of fuzzy neurons showed the potential to

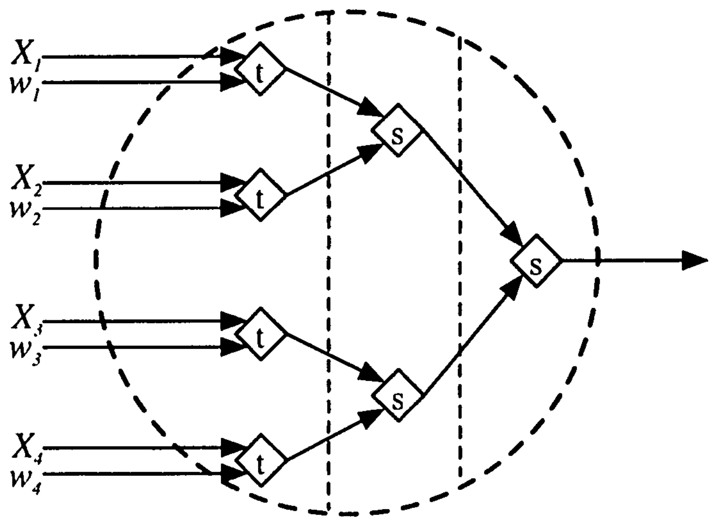


Figure 4.1: Hardware architecture for an 4-input OR neuron, having 3 pipe stages. The diamonds represent digital circuits performing t/s-norm computations.

achieve network clock rates in excess of 66 MHz, with the latency, i.e. the number of cycles it takes to get an output from corresponding input, dependent upon the size of the network (number of inputs, layers, etc.). To give a view of the potential for a hardware-based FNN running at this clock speed, consider a dataset of 1000 data points: it would take only $15\mu s$ to compute network output for all of this data, showing excellent potential for very high speed learning.

### 4.2.3 Design Issues

The notion of granulation raises questions of practical relevance, as there must be some trade-off between system performance and learning accuracy. How precise does this form of information processing need to be to retain an adequate approximation of the learning and representation capabilities of continuous fuzzy neural networks? Another important issue in an attempt to model fuzzy neurons in hardware is the choice of triangular norms. Here computational simplicity is of importance, as we would like to avoid complex operations such as multiplication and division, which would result in slower processing and more hardware. However, we still want our networks to retain their representation and inferential capabilities. A disadvantage of minimum and maximum is the lack of interactivity between operands, with the result reflecting the influence of only one operand. Conversely, algebraic product and probabilistic sum provide this interactivity, but at a high computational cost. Lukasiewicz AND and OR operations provide a balance between the two, although there is a question of how well they strike this balance.

The method of network optimization is a critical feature of the overall system, addressed in detail later in the paper. Regardless of the learning mechanism, it is intended to become a modular, autonomous optimization machine, located on-chip or, at the very least, onboard. The idea is to have a system that continually learns and adapts if necessary, in order to adapt to dynamic environments, working toward a better solution.

Qualitative analysis of these issues is insufficient. Later in the chapter, we also conduct detailed experiments in order to address them quantitatively as well.

## 4.3 Hardware-Targeted Learning

An important FNN design issue deals with the learning effectiveness of the networks. As we have already shown, straight parametric training becomes very slow when the size of the network gets large, with no guarantee of convergence. The main cause of this problem is the lack of any preliminary knowledge of about the structure of the network, instead having to use a fully connected topology where all neurons are connected with the neurons in the neighbouring layer. In following the framework presented in Chapter 3, rather than attempting to train the fuzzy neural network from scratch, we concentrate first on discovering an underlying structure that reflects the logic nature of the data. This effectively reduces the number of connections to be adjusted during the second phase of this hybrid-learning scheme; here we concentrate on the parametric optimization of the network in order to learn the finer details of the data (system, concept) for improved accuracy.

### 4.3.1 Structure Discovery

Structural optimization as part of a hybrid learning approach has been studied in detail in this thesis. Earlier studies employed methods of evolutionary computing including genetic algorithms and genetic programming, with considerable success [5, 24–26]. However, the computational efficiency was somewhat lacking, due to the resource-intensive nature of such methods. The parallelization of hardware for such population-based methods would be helpful here, but there would be a need of considerably more effort in designing hardware-based evolutionary algorithms, going above and beyond implementations of fuzzy neurons. Regardless, studies conducted during the development of this thesis have taken a different path, making use of logic minimization algorithms for discovering a concise, logic-based structure within system data.

This novel approach to structural learning has considerable potential for implementation as part of an autonomous optimization vehicle on the proposed neurofuzzy platform. Efforts in the development of hardware-based, on-chip logic minimizers [3, 14, 15] have achieved significant success with very fast performance and effective minimization. Whether implemented on separate hardware or on the FPGA itself, an on-chip minimizer quite effectively complements the dynamic reconfiguration abilities of the FPGA. Once training data is abstracted into binary information granules and provided to the minimizer, it can quickly perform the optimization and directly communicate the results to the configuration control of the FPGA, using a portion of its resources to create a custom fuzzy neural network, ready for the next step in core optimization.

### 4.3.2 Parametric Refinement

The FNN combines its optimized knowledge-based structure with inherent adaptive abilities in order to learn finer details. Although we have previously taken advantage of gradient-based backpropagation methods, when considering a hardware implementation, it would be beneficial to avoid such methods, which requires difficult circuit implementations and costly gradient calculations. Instead, methods of simultaneous perturbation (SP) are suitable here, having already achieved success in hardware implementations [11, 16]. The advantage of these methods is simplicity, only needing values of the performance index for making weight adjustments. This makes implementation a much easier task, as the algorithm does not have to take error backpropagation circuits into account. In this study a unique simultaneous perturbation algorithm is proposed. We use $Q_N$ to express the network approximation error for all training data (recall its definition in Chapter 3 as a sum-of-squared errors. View that it can be considered a function of the network's connections, $Q_N(\mathbf{w})$, as their values ultimately control its output.

A perturbation magnitude $p$ is used in conjunction with a sign vector $\mathbf{s}$ to modify the connections $\mathbf{w}$ of the network, where $\mathbf{w}$ and $\mathbf{s}$ are vectors of length equal to the number of connections in the network. The sign vector contains elements that have been randomly assigned a value of 1 or -1, which determines whether corresponding connections in $\mathbf{w}$ receive positive or negative perturbations. As well, a decay constant, $d$, is applied to $p$ after each learning epoch to help the algorithm converge, realizing coarse to fine grained weight perturbations as $d$ gracefully decays towards zero. The complete algorithm is shown in Figure 4.2.

Note that we need only one forward operation of the FNN for each learning epoch, a distinct advantage over a traditional backpropagation method. From the point of view of hardware, the simplicity of this learning method is highly beneficial. If we used a backpropagating method, we would have to include error-propagation circuits for all weights in the network. This error propagation through the weights, the wiring for all weights in the network, and the overall circuit design becomes difficult. Instead, the SP algorithm needs only values of $Q_N$ in order to update all weights, requiring only one circuit realizing this learning mechanism.

€

```
randomly initialize weights
determine initial $Q_N$(w)
while($Q_N$(w) > goal OR #epochs < maximum)
loop
   randomize s
   determine $Q_N$(w + s*$p$)
   if($Q_N$(w + s*$p$) < $Q_N$(w))
      $Q_N$(w) = $Q_N$(w + s*$p$)
      w = w + s*$p$
   fi
   $p = p - d$
done
```

Figure 4.2: Proposed simultaneous perturbation (SP) learning algorithm.

The calculations required for the learning rule are also extremely simple, requiring no complex circuits for multiplication or division, just ones for subtraction, addition, comparisons, and sign changes. Since we are dealing with a granular integer representation of the unit interval in error calculations, the squared error measure is simplified as well. Additionally, the algorithm allows the utilization of non-differentiable t/s-norms if desired, and is very friendly to granular weight adjustments within the unit interval, which was the main motivation behind coarse-grained perturbations during the earlier learning epochs.

With the final details of the platform introduced, we show a block diagram in Figure 4.3, detailing its topology.

## 4.4 Experimental Studies

In this section, we present results of comprehensive experiments carried out dealing with both synthetic and real-world data. The presentation and discussion of the results of several sets of experiments are divided into sub-sections.

### 4.4.1 Boolean Network Development

Here we conduct experiments on synthetic binary data, where we intend to give a quantitative view of the effect that input dimensionality has on a complexity of the learning problem. The experiments here complement ones carried out in the previous chapter, further demonstrating the importance of structural optimization,

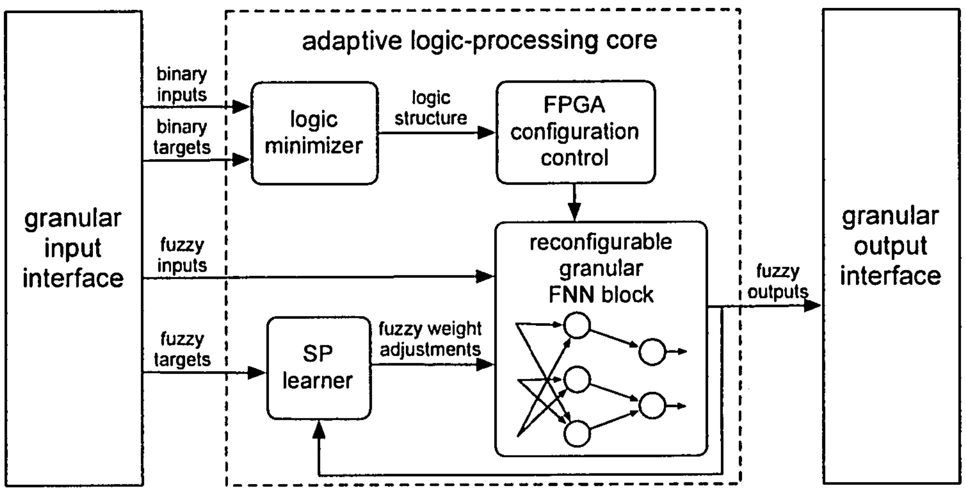


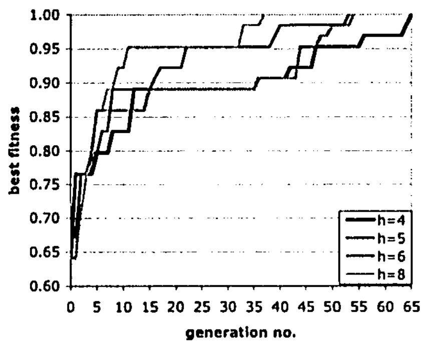
Figure 4.3: Topology of the proposed hardware-based adaptive logic-processing platform.

as well as the value of logic minimization algorithms.

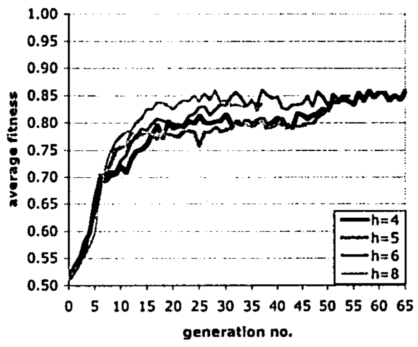
Using binary connections for fuzzy neurons processing binary data, recall that they become simple AND and OR gates. Having on/off connections such as these, we can realize structural optimization in the selection of neural inputs, i.e. connections equal to 1 for AND neurons make their corresponding inputs unnecessary, and similarly for OR neurons when their connections are equal to 0. Noting the sum-of-products topology in Figure 3.2, we attempt to train these Boolean logic processors (LP) using evolutionary means via a genetic algorithm, where the number of hidden neurons $h$ is constant for a given training instance. The chromosome structure consists of all connections in the network and hence is a binary version. As in the previous chapter, standard operators are used for the GA [19], with parameter values determined to be suitable through preliminary experimentation: a tournament based selection with an elitist mechanism (the best individual is always carried into the next generation); a standard mutation operator at a rate 0.04; a multi-point crossover operator (with the number of points varying randomly between pairs of individuals) at a rate of 0.5; a fitness function $f$, taken as a variation of $Q$, where we consider all $N$ input-output pairs at once as an average (noted that a squared error is unnecessary since we are dealing with binary errors):

$$Q_{GA} = \frac{1}{N}\sum_{c=1}^{N} |Y_k(c) - TARGET_k(c)|$$

Note here that $k = 1$ as these networks have a single Boolean output. As we normally try to maximize the fitness function for GA, $f$ is taken as $1 - Q_{GA}$, which



(a) Best fitness



(b) Avg. fitness

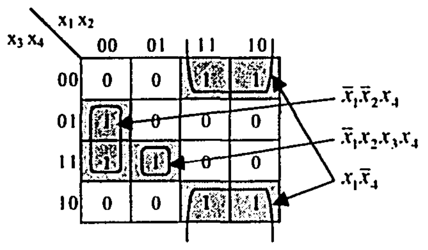
Figure 4.4: Best and average fitness plots of the 6-variable data set shown in Figure 4.5(c).

has a maximum value of 1. In typical fashion, all experiments were conducted using a population of 200 individuals and running for a maximum of 1000 generations. These values were found experimentally to be justifiable.

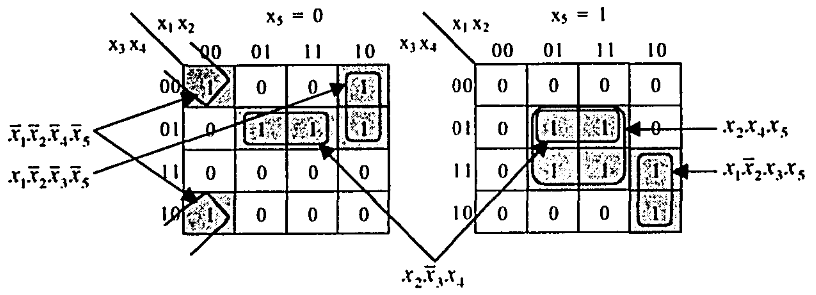
Here the learning goal is to find a minimal sum-of-products representation of Boolean datasets. The data was synthetically generated using a Boolean LP with randomly generated connections in the hidden layer, composed of three AND neurons. For $n$ inputs, these datasets cover the complete truth table of $2^n$ entries. With increasing dimensionality comes an exponential explosion in problem complexity; the problem is also further complicated by the introduction of complements into the learning process, effectively doubling the number of inputs for training.

We first consider truth tables of 4, 5, and 6 variables. This low dimensionality allows the results to be easily visualized with Karnaugh maps (Figure 4.5), providing an effective means for evaluating the performance of a GA-trained Boolean LP in finding a simplified expression for the logical input-output relationship. After training a Boolean LP to achieve zero error in each case (progress of the GA for $n$ = 6 shown in Figure 4.4), the resulting network structures were translated into the Boolean relationships they represent (Table 4.1). These results can then be applied to the K-map representations of the respective functions (Figure 4.5). Note that for all cases, the learning process results in discovery of the least possible number of terms (i.e. number of hidden neurons, $h$) necessary to represent the relationship. For illustrative purposes, the architecture of the trained LP representing the 4-variable function in Figure 4.5(a) is shown in Figure 4.6.

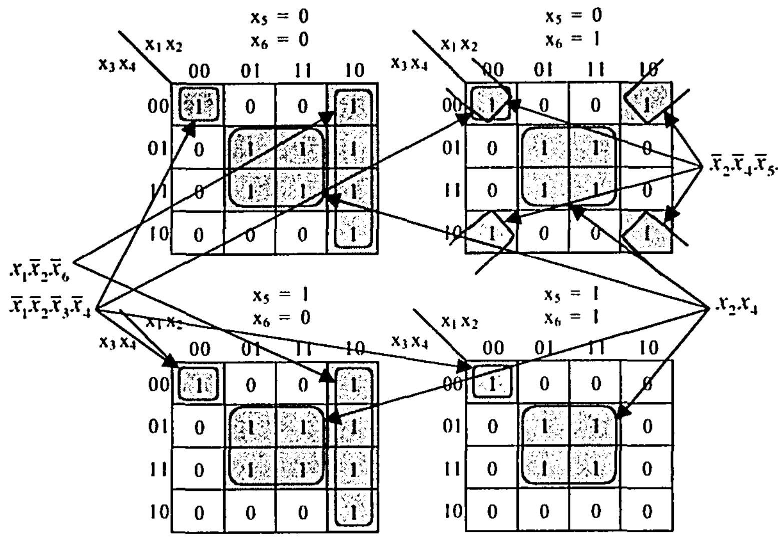
Next, we train Boolean logic processors with increasing dimensionality. The dimensionality of the problem, $n$, ranged from 2 to 10 inputs, and 50 different Boolean datasets were generated for each value of $n$. We then trained network configurations with this data, varying the number of neurons in the hidden layer.



(a) 4 variables


$x_5 = 0$
$x_5 = 1$
$\bar{x}_1.\bar{x}_2.\bar{x}_4.\bar{x}_5$
$x_1.\bar{x}_2.\bar{x}_3.\bar{x}_5$
$x_2.x_4.x_5$
$x_1.\bar{x}_2.x_3.x_5$
$x_2.\bar{x}_3.x_4$

(b) 5 variables


$x_5 = 0$, $x_6 = 0$
$x_5 = 0$, $x_6 = 1$
$x_5 = 1$, $x_6 = 0$
$x_5 = 1$, $x_6 = 1$
$x_1.\bar{x}_2.\bar{x}_6$
$\bar{x}_1.\bar{x}_2.\bar{x}_3.\bar{x}_4$
$\bar{x}_2.\bar{x}_4.\bar{x}_5.x_6$
$x_2.x_4$

(c) 6 variables

Figure 4.5: Results of training shown on K-maps for Boolean systems with 4, 5, 6 input variables. Shadowed regions visualize the simplification (reduction) effect obtained through the learning.

Table 4.1: Boolean LP interpretations

| $n$ | **Boolean Expression** |
|---|---|
| 4 | $y = [\bar{x}_1 \bullet x_2 \bullet x_3 \bullet x_4] + [x_1 \bullet \bar{x}_4] + [\bar{x}_1 \bullet \bar{x}_2 \bullet x_4]$ |
| 5 | $y = [\bar{x}_1 \bullet \bar{x}_2 \bullet \bar{x}_4 \bullet \bar{x}_5] + [x_2 \bullet \bar{x}_3 \bullet x_4] + [x_1 \bullet \bar{x}_2 \bullet \bar{x}_3 \bullet \bar{x}_5] + [x_2 \bullet x_4 \bullet x_5] + [x_1 \bullet \bar{x}_2 \bullet x_3 \bullet x_5]$ |
| 6 | $y = [\bar{x}_1 \bullet \bar{x}_2 \bullet \bar{x}_3 \bullet \bar{x}_4] + [x_1 \bullet \bar{x}_2 \bullet \bar{x}_6] + [x_2 \bullet x_4] + [\bar{x}_2 \bullet \bar{x}_4 \bullet \bar{x}_5 \bullet x_6]$ |

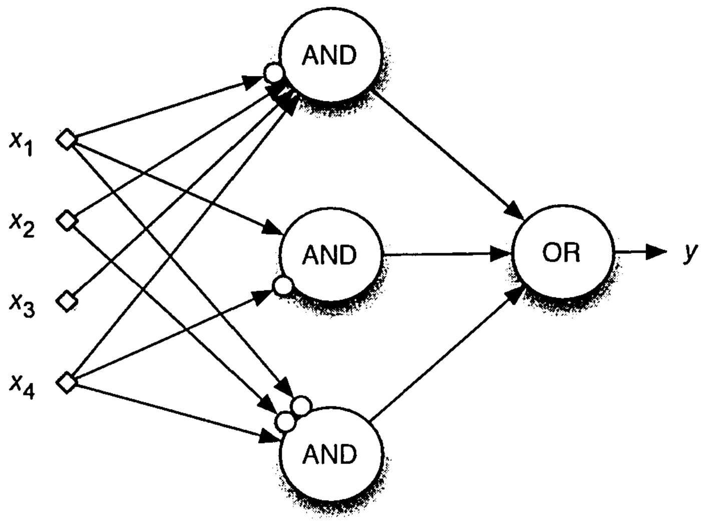


Figure 4.6: The trained Boolean network for the 4-variable function shown in Figure 4.5(a) (small circles denote complements). Note that for the AND neurons, only the essential connections are displayed, as the others have no effect.

These configurations always admitted both inputs and their complements, essentially doubling the number of inputs (up to 20). Figure 4.7 shows the training results. As a performance measure, we count the number of datasets completely represented (out of 50), i.e. the datasets in which the network produced zero error.

Since the datasets were randomly generated with three minterms, it is not surprising that the network configuration with $h = 3$ performed best. From the results of the smaller networks, we see that the GA is able to find simplified relationships among the data (i.e. less terms), if they exist. This is evident when viewing final networks trained for a particular dataset, where we see that, for example, if the relationship can be represented with $s < 3$ terms, all networks with $h \geq s$ are able to find the solution. However, the most important observation to make from Figure 4.7 is how dimensionality severely affects training success. Note the general pattern in the results, showing that as dimensionality increases, the training becomes less effective. This leads to the conclusion that evolutionary methods can have limited applicability to such highly dimensional problems; interestingly, the Espresso was able to find solutions to every one of these problems, in a matter of milliseconds.

### 4.4.2 Real-World System Modelling

We now present experiments conducted with some of the machine learning datasets used in the previous chapter. These experiments are important, allowing us to evaluate how granularity levels and triangular norms affect learning and generalization. As well, we can now validate the effectiveness of our hybrid-learning scheme, particularly the simultaneous perturbation learning. To simulate the proposed neurofuzzy hardware platform, models of granular fuzzy neurons were built in a software environment along with an implementation of the SP learning algorithm. As we have not yet implemented any logic minimization architecture in hardware, we made use of Espresso for heuristic logic minimization. These software implementations provided a versatile and highly configurable simulation and testing environment for the hardware-based FNN processing core. As for the granular interfaces, we used triangular membership functions for input and output fuzzy sets, with their simplicity making them suitable candidates for hardware implementation. We used three fuzzy sets distributed evenly over each continuous variables universe of discourse, and any nominal variables were encoded using 1-out-of-$n$. To maintain a good level of accuracy in the output interface, we use a centre-of-gravity (COG) decoding (defuzzification) scheme. Additionally, for all experiments we use a root-mean-squared-error (RMSE) formula to measure performance of the resultant models ($V$ as defined previously).

## Auto-Mpg data

The *Auto-Mpg* dataset consists of 392 8-dimensional points. Here the goal is to create a model of this data to predict the mileage of an automobile (MPG) based on the remaining variables. With three fuzzy sets defined for each variable, their labels are taken as *low*, *medium*, and *high*. Therefore, we are attempting to learn the concepts of low, medium, and high mileage within our FNN-based processing core, in order to arrive, through the output interface, at a numeric value predicting mileage. Note that the learning is completed for 60% of the data treated as a training set, randomly selected ten times to complete a ten-fold cross-validation.

In the first phase of learning we attempt to find a structure in the data through logic minimization. From this structure we derive a parameter-free FNN, varying the granularity, $g$, of these fuzzy set memberships, in order to view the effects of such direct discretization. The results are shown in Figure 4.8, where $g$ ranges from 1 to 10 bits, and we also include the result obtained for a continuous representation. Note that for 1 bit of granularity, we simply provide the performance results of the binary structure with binary data. Interestingly, when $g$ reaches a level of 4-5 bits, the performance becomes practically equal to a continuous representation. As well, note that the minimum/maximum t/s-norm operators appear to perform best here, which could perhaps be attributed to the fact that they most closely resemble Boolean AND and OR operations.

Proceeding with parametric optimization of the discovered structure, we now have an opportunity to view the effectiveness of our SP learning method in conjunction with different t/s-norms and varying fuzzy input/connection granularity. The training was run for 2500 epochs with a perturbation magnitude of 0.5, reaching zero by the last epoch. In view of the application area, we are not simply training the network with high levels of granularity and performing subsequent granulation (discretization). This method is not compatible with the idea of a granular logic-processing core in hardware, where we likely would not have the resources to first train with high levels of granularity. Instead, it is much more desirable to train from the specified granularity, and see how well each configuration performs. To give a baseline performance comparison, we also train a continuous FNN (product/probabilistic sum) with the standard gradient-based technique used previously, setting the learning rate to 0.01 and running for 2500 epochs. The training results are presented in Figure 4.9, showing significant performance increases over that of Figure 4.8, stemming from the neural augmentation. Again we see that low granularities of just 4-5 bits are able to perform as well as much higher levels, with very good performance seen even when $g = 2$ bits. As well, view that the performance of the SP learning method comes very close to that of backpropagation, all while exhibiting significantly less computational complexity. Further, another positive observation of these results is the performance of each set of t/s-norms. View how

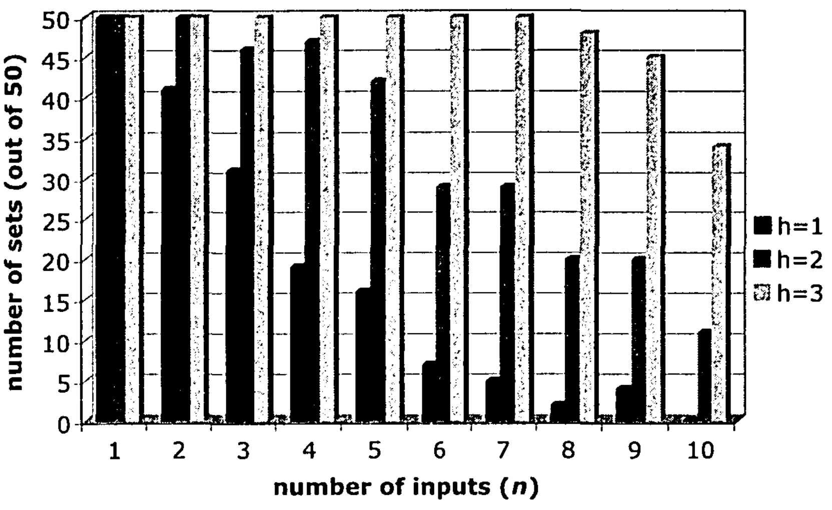
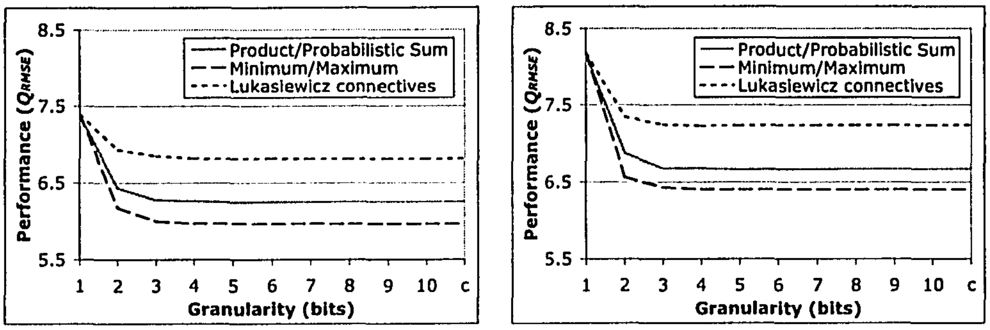


Figure 4.7: Performance results of training the Boolean SOM LP while varying the size of the hidden layer, measured as the number of datasets completely represented (out of 50).

the Lukasiewicz appear to strike a good balance between learning accuracy and computational simplicity, approaching the performance of product and probabilistic sum.

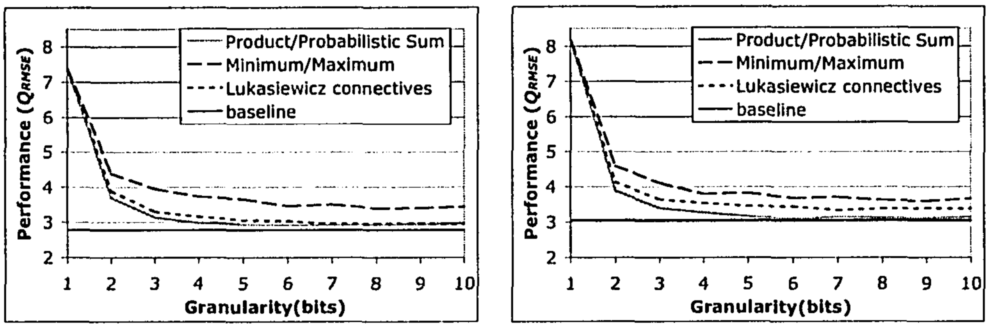
Choosing the best performing Lukasiewicz FNN at a granularity of 4 bits, we compare it with the best performing continuous FNN, showing their respective scatter plots in Figure 4.10. We also show the training progress for these networks, seen in Figure 4.11. Here the differences between the learning algorithms are quite apparent: the progress of the gradient learner is smooth, while the SP learner is very rough. On a per-epoch basis, the gradient method learns faster, but is actually much slower when the amount of computation is taken into account. This behaviour of the SP learner is largely due to its stochastic nature. As well, note that the SP algorithm detailed previously shows that the performance-comparing step of the learning should never result in a worse error than the current one as it progresses. This behaviour is not seen here due to the simultaneous training of three network structures for each of the three fuzzy outputs of the network, which cannot guarantee the performance wont regress once a crisp value is decoded from the results of the latest weight adjustments.

As we have already extensively shown how structurally and parametrically optimized fuzzy neural networks provide highly-understandable knowledge, there is no need to go into detail of interpretation results here. However, like before, pruning



(a) Training (b) Testing

Figure 4.8: Average performance results (Auto-Mpg data) of parameter-free FNN structure used with fuzzy inputs at varying levels of granularity (in bits) for the training set and testing set. 'c' on the x-axes denote a continuous representation.



(a) Training (b) Testing

Figure 4.9: Average performance results (Auto-Mpg data) after SP parametric training of structurally-optimized granular FNNs. The baseline indicates the performance of a backpropagation-trained continuous FNN.

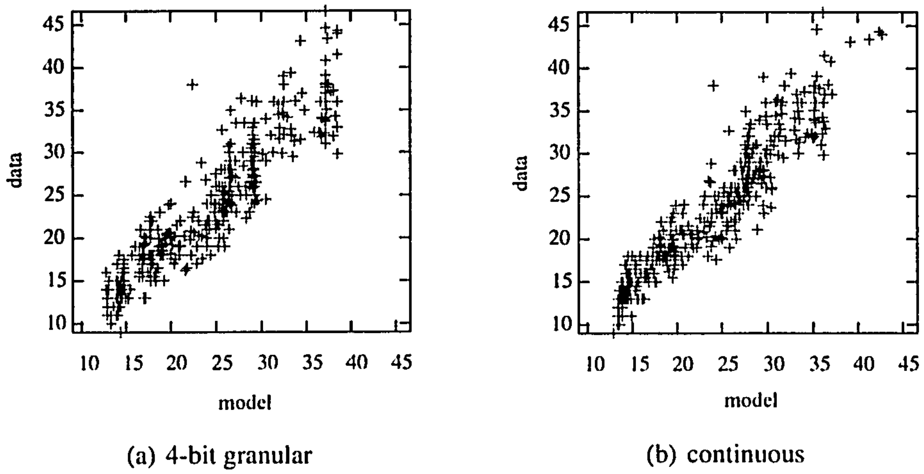


Figure 4.10: Scatter plots showing (Auto-Mpg) data vs. model for a 4-bit granular FNN with Lukasiewicz connectives and a continuous FNN with product/probabilistic sum

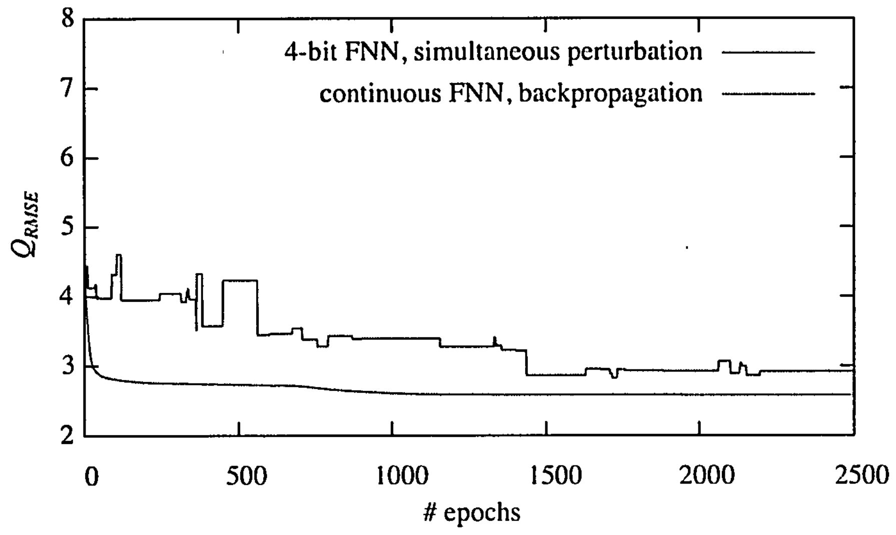


Figure 4.11: Training progress for 4-bit granular FNN with Lukasiewicz connectives and continuous FNN with product/probabilistic sum.

could be used to reduce structural complexity if desired, and the networks themselves contain highly intuitive rules. The knowledge gained was of the quality encountered in the previous chapter, with the only difference amounting to discretized rather than continuous weights.

**Boston Housing data**

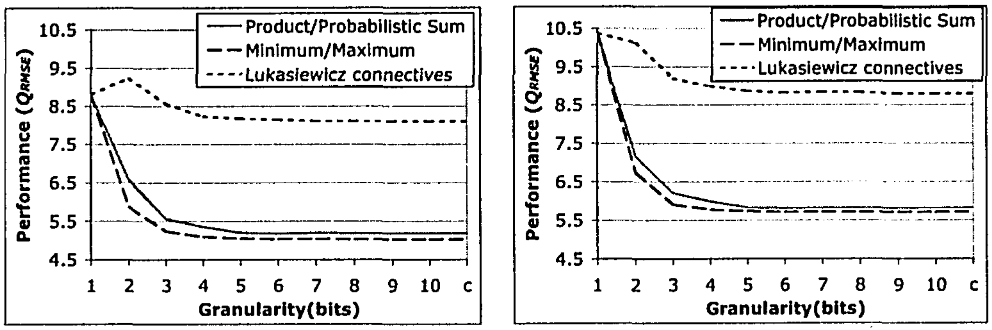
Here we work with the Boston housing data, where we are attempting to learn the concepts of *low*, *medium*, and *high* median housing prices. Like before, the learning task is completed for 60% of data treated as a training set using ten-fold cross-validation.

After structural optimization, the derived parameter-free FNN was subjected to varying granularity as , with the results shown in Figure 4.12, with $g$ ranging from 1 to 10 bits in addition to a continuous representation. Again we see how the performance is practically identical at 4-5 bits as compared to a continuous representation. Noting the large performance gap between Lukasiewicz connectives here in comparison to the other t/s-norms, it is apparent that different realizations of logic operators (AND and OR) come with different abilities to approximate data. Without any ensuing parametric optimization, we have found that some of them could perform poorly.

Proceeding with parametric optimization of the optimized structure, the same procedure as outlined with the Auto-Mpg data was carried out here, with the results shown in Figure 4.13. Again, we see significant performance increases. The same observations made during the Auto-Mpg case study can also be made here, showing very positive results for low levels of granularity with all t/s-norms. Interestingly, we also see Lukasiewicz connectives outperforming product/probabilistic sum. Again choosing the best performing Lukasiewicz FNN at a granularity of 4 bits, we compare it with the best performing continuous FNN, showing their respective scatter plots in Figure 4.14.

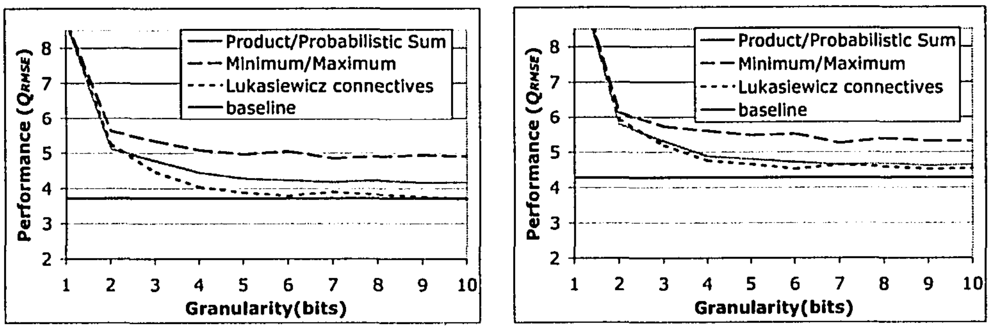
## 4.5 Review

Hardware-based fuzzy neurocomputing is a critical step towards the design of transparent, adaptive, and highly autonomous intelligent systems. In this chapter, we proposed a promising neurofuzzy platform based upon the principles of adaptive fuzzy modelling, and performed extensive qualitative and quantitative analysis.



(a) Training (b) Testing

Figure 4.12: Average performance results (Boston housing data) of parameter-free FNN used with fuzzy inputs at varying levels of granularity (in bits) for the training set and testing set. 'c' on the x-axes denote a continuous representation.



(a) Training (b) Testing

Figure 4.13: Average performance results (Boston housing data) after SP parametric training of structurally-optimized granular FNNs. The baseline indicates the performance of a backpropagation-trained continuous FNN.

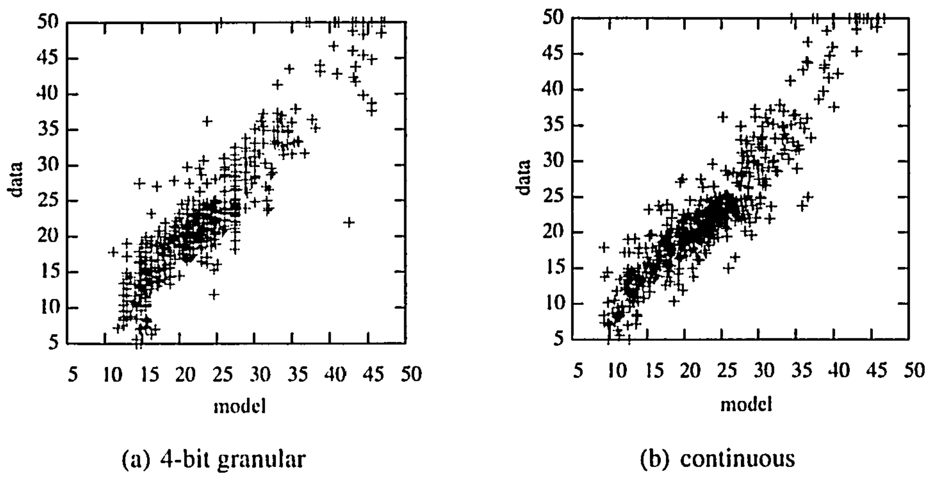


Figure 4.14: Scatter plots showing (Boston housing) data vs. model for a 4-bit granular FNN with Lukasiewicz connectives and a continuous FNN with product/probabilistic sum

## Bibliography

[1] IEEE standard verilog hardware description language. *IEEE Std. 1364-2001*, pages 0_1–856, 2001.

[2] IEEE standard vhdl language reference manual. *IEEE Std. 1076-2002 (Revision of IEEE Std 1076, 2002 Edn.)*, pages 0_1–300, 2002.

[3] S. Ahmand and R. Mahapatra. M-trie: An efficient approach to on-chip logic minimization. In *Proc. IEEE/ACM Conf. Comp. Aided Des.*, pages 428–435, 2004.

[4] S. Aunet, Y. Berf, and T. Saether. Real-time reconfigurable linear threshold elements implemented in floating-gate CMOS. *IEEE Trans. Neural Networks*, 14:1244–1256, 2003.

[5] Andreas Bastian. Identifying fuzzy models utilizing genetic programming. *Fuzzy Sets Syst.*, 113(3):333–350, 2000.

[6] A. Costa, A. DeGloria, F. Guidici, and M. Olivieri. Fuzzy logic microcontroller. *IEEE Micro*, 17:66–74, 1997.

[7] A. Costa, A. DeGloria, and M. Olivieri. Hardware design of asynchronous fuzzy controllers. *IEEE Trans. Fuzzy Syst.*, 4:328–338, 1996.

[8] A. Gopalan and A. H. Titus. A new wide range Euclidean distance circuit for neural network hardware implementations. *IEEE Trans. Neural Networks*, 14:429–438, 2003.

[9] S. Guo, L. Peters, and H. Surmann. Design and application of an analog fuzzy logic controller. *IEEE Trans. Fuzzy Syst.*, 4:429–438, 1996.

[10] H. Hikawa. A digital pulse-mode neuron with piecewise linear activation function. *IEEE Trans. Neural Networks*, 14:1028–1037, 2003.

[11] V. F. Koosh and R. M. Goodman. Analog vlsi neural network with digital purturbative learning. *IEEE Trans. Circuits Syst. II*, 49:359–368, 2002.

[12] J. Kowalski. 0.8 /spl mu/m CMOS implementation of weighted-order statistic image filter based on cellular neural network architecture. *IEEE Trans. Neural Networks*, 14:1366–1374, 2003.

[13] G. Louverdis and I. Andreadis. Design and implementation of a fuzzy hardware structure for morphological color image processing. *IEEE Trans. Circuits Syst. Video Technol.*, 13:277–288, 2003.

[14] Roman Lysecky and Frank Vahid. A codesigned on-chip logic minimizer. In *Proc. of the First IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 109–113, 2003.

[15] Roman Lysecky and Frank Vahid. On-chip logic minimizer. In *Proc. of the Design Automation Conference*, pages 334–337, 2003.

[16] Y. Maeda and T. Tada. FPGA implementation of a pulse density neural network with learning ability using simultaneous perturbation. *IEEE Trans. Neural Networks*, 14:688–695, 2003.

[17] M. Milev and M. Hristov. Analog implementation of ANN with inherent quadratic nonlinearity of the synapses. *IEEE Trans. Neural Networks*, 14:1187–1200, 2003.

[18] J. W. Mills, M. G. Beavers, and C. A. Daffinger. Lukasiewicz logic arrays. In *Proc. of the 20th Int. Symp. on Multiple-Valued Logic*, pages 4–10, 1990.

[19] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1998.

[20] R. J. Navas-Gonzalez, F. Vidal-Verdu, and A. Rodriguez-Vazquez. Neuro-fuzzy chip to handle complex tasks with analog performance. *IEEE Trans. Neural Networks*, 14:1375–1392, 2003.

[21] N. Nedjah and L. M. Mourelle. FPGA-based hardware architecture for neural networks: Binary radix vs. stochastic. In *Proc. of the 16th Symp. on Integrated Circuits and Systems Design*, pages 111–116, 2003.

[22] A. R. Omondi and J. C. Rajapakse. Neural networks in FPGAs. In *Proc. of IEEE Int. Conf. on Neural Information Processing*, pages 954–959, 2002.

[23] Witold Pedrycz, C. H. Poskar, and P. Czezowski. A reconfigurable fuzzy neural network with in-situ learning. *IEEE Micro*, pages 19–30, August 1995.

[24] Witold Pedrycz and Marek Reformat. Evolutionary fuzzy modeling. *IEEE Trans. Fuzzy Syst.*, 11:652–665, 2003.

[25] Witold Pedrycz and Marek Reformat. Genetically optimized logic models. *Fuzzy Sets Syst.*, 150:351–371, 2005.

[26] Marco Russo. Genetic fuzzy learning. *IEEE Trans. Fuzzy Syst.*, 4:259–273, 2000.

[27] Z. Salcic. High-speed customizable fuzzy-logic processor: Architecture and implementation. *IEEE Trans. Syst., Man, Cybern.*, 31:731–737, 2001.

[28] F. A. Samman and R. S. Sadjad. Analog mos circuit design for reconfigurable fuzzy logic controller. In *Proc. of Asia-Pacific Conf. on Circuits and Systems*, pages 151–156, 2002.

[29] S. Sato, K . Nemoto, S. Akinoto, M. Kinjo, and K. Nakajima. Implementation of a new neurochip using stochastic logic. *IEEE Trans. Neural Networks*, 14:1122–1127, 2003.

[30] G. Viot. Meet the 68HC12. In *Northcon/96*, pages 180–185, 1996.

[31] F. Yang and M. Paindavoine. Implementation of an RBF neural network on embedded systems: Real-time face tracking and identity verification. *IEEE Trans. Neural Networks*, 14:1162–1175, 2003.

[32] S. B. Yun, Y. J. Kim, S. S. Dong, and C. H. Lee. Hardware implementation of neural network with expansible and reconfigurable architecture. In *Proc. of IEEE Int. Conf. on Neural Information Processing*, pages 970–975, 2002.

no text

# Chapter 5

# Closing

The benefits of modelling complex real-world systems, concepts, and processes are numerous, most evident when a model can be efficiently and successfully designed with an appropriate balance between accuracy and transparency. In this thesis we introduced and validated an efficient modelling framework for automatically identifying transparent and accurate models from real-world system data, proceeding with the construction of a solid foundation for potential implementation in high-speed electronic hardware for real-time applications.

## 5.1 Discussion

As the complexity of a system increases, it becomes more difficult and eventually impossible to make a precise statement about its behaviour. It is quite evident that one does not always need need such precision that computing platforms provide. In contrast, human beings have the ability to take in and evaluate all sorts of information from the physical world we are in contact with and to mentally analyze, average and summarize all this input data into an optimum course of action. We are essentially granular computers, providing a great deal of motivation to try and mimic this behaviour in system modelling. Consequently, modelling platform taken in this thesis was based heavily on fuzzy sets and fuzzy logic, allowing machines "think" like we do as much as possible. The idea of fuzzy models emerges as we use fuzzy sets to define a human-like perspective of the system's environment, and employ fuzzy logic for intuitive knowledge-based processing in order to approximate system behaviour.

Despite the user-friendliness of fuzzy models, their development can still be quite difficult and time-consuming if attempted manually. As a result, it is often highly desirable to employ some means of automatic model identification, where a machine may learn from raw system data, using it as training examples in order to build a transparent knowledge base from which to act and from which we, as users, can learn or make intuitive modifications. This desire lead to the goal of realizing an effective data-driven design framework for fuzzy models. In seeking this goal, one must note that the two fundamental modelling thrusts, namely accuracy and transparency, are somewhat in conflict with each other. In the presence of highly dimensional systems, these problems are amplified considerably. Additionally, coming up with effective solutions to these apparent problems can also result in the requirement of significantly longer computation times.

Motivated by these challenges, we recognized the fundamental link between Boolean and fuzzy logic, leading to the novel application of established methods in Boolean logic minimization to fuzzy model identification. They are able to find compact logic-based structures in data, revealing the most pertinent details of the target system's behaviour. To improve upon accuracy, these structures discovered from minimization were directly utilized in designing the architecture of heterogeneous fuzzy neural networks for the adaptive, knowledge-based logic-processing core of the fuzzy model. As a result, the framework is capable of discovering concise, accurate, and human-interpretable logic-based structures in real-world data.

In broadening the range of potential applications even further, we conducted detailed investigations into the potential implementation of the adaptive fuzzy modelling framework in high-speed digital electronics. This lead to the proposal of an optimized hardware-accelerated platform supporting the realization of intelligent systems for operation in dynamic, real-time environments. The conceptual architecture was thoroughly examined through qualitative and quantitative investigations, showing excellent potential for its future implementation. By effectively learning and developing a knowledge base of real-world concepts, such a hardware implementation can realize cognition, inferring the relevance of many concepts simultaneously in real-time in order to form a high-level opinion of the current state of its environment.

## 5.2 Conclusions and Future Work

With the use of heuristic methods of logic minimization in conjunction with neuro-fuzzy augmentation, many problems faced by researchers today in the field of fuzzy model identification have been overcome. In designing the adaptive processing core, the most critical component of a fuzzy model, there is no longer any need to rely on methods that provide inaccurate behavioural approximation, difficult interpretability, computational complexity, or limiting parameters (such as choosing a fixed number of rules) that must be experimented with. As well, we have demonstrated an ability to handle high-dimensional problems while retaining excellent computational efficiency. Employing methods of logic minimization ensures that the logic-processing nature of the model is fully utilized. Through understanding the logic nature of real-world data from the very beginning of the design, they reveal an intuitive and concise structure that directly forms a blueprint for a heterogeneous fuzzy neural network architecture.

The research conducted in the construction of the framework has resulted in its implementation as an efficient, multiplatform software-based fuzzy model development environment, *Third Eye*. However, there are still further issues worth investigating in the pursuit of an ideal adaptive fuzzy modelling framework:

- Interpretation mechanisms where emphasis is placed on a more sophisticated pruning process, where we could consider the importance of various criteria such as accuracy (how much the pruning affects the performance index) and interpretability, where structural complexity could be measured by such parameters as the number of rules and/or literals.

- Further optimization of the granular interface is certainly worth detailed consideration, with potential to improve accuracy. They may be better constructed by capturing the nature of the data: methods such as fuzzy equalization [3] or various techniques of fuzzy clustering [2] could be of interest here.

- A newer logic minimizer, BOOM [1], claims superiority over ESPRESSO-II in a variety of areas. Interestingly, one of the claims made involves being better equipped to handling sparsely defined, high-dimensional Boolean systems. Given that this is precisely what we get from the granulation of a real-world continuous dataset, BOOM may well be worth investigating. Although ESPRESSO-II had no problems in quickly discovering minimal structures for the data experimented with here, BOOM could prove useful for much larger problems that ESPRESSO-II might struggle with, ex. thousands of variables.

With regards to the hardware platform, the hybrid-learning approach detailed here presented very hardware-friendly methods for both structural and parametric learning, with great potential for forming an on-chip autonomous optimization vehicle for creating FNN-based hardware engines. From experiments, the effectiveness of this learning was validated with a variety of triangular norms; in particular, Lukasiewicz connectives have been shown to provide comparable inferential capabilities to product and probabilistic sum, yet remaining relatively simple computationally, thus making them good candidates for hardware implementation. As well, an important discovery was made regarding the necessity of precision in representing the unit interval for fuzzy information processing, as we demonstrated how fuzzy neurons are able to retain their learning capabilities while dramatically reducing this precision (granularity). We have determined that just a few bits of resolution, at the most 4 or 5 bits, are necessary to achieve performance on par with that of a continuous representation, with good results seen for as little as 2 bits of precision. This reduction in granularity is important in a hardware environment, where would potentially need to store and process hundreds, even thousands of neural connections and information granules. In contrast, if we needed to maintain high granularities of 8 or 16 bits, this would severely impact hardware resources.

Concerning hardware development, an FPGA platform and the use of VHDL will be ideal for our efforts, as the environment allows much freedom in the creation of and experimentation with hardware-based fuzzy neurons. In particular, one of the most interesting aspects is the combination of on-chip logic-minimization with dynamic FPGA reconfiguration for autonomously discovering and implementing logic-based neurofuzzy structures. The idea of a system that is able to physically reconfigure itself in order to adapt to its changing environment is extremely appealing, and worth detailed investigations.

# Bibliography

[1] J. Hlavicka and P. Fiser. BOOM - a heuristic Boolean minimizer. In *Proc. of the IEEE/ACM International Conference on Computer Aided Design (ICCAD 2001)*, pages 439–442, 2001.

[2] W. Pedrycz. Conditional fuzzy clustering in the design of radial basis function neural networks. *IEEE Trans. Neural Networks*, 9:601–612, Aug. 1998.

[3] W. Pedrycz. Fuzzy equalization in the construction of fuzzy sets. *Fuzzy Sets Syst.*, 119:329–335, 2001.

# Appendix A

# *Third Eye*: Demonstration through example

We present a demonstration through example of the *Third Eye* fuzzy model development environment, using the Auto-Mpg dataset. This demonstration is carried out on a UNIX shell. The software can be compiled and run in any similar environment, such as Sun SparcOS, HPUX, IBM AIX, BSD, Linux, Mac OS X, or Windows XP (via Cygwin).

The raw system data should be in standard .csv (comma-separated values) format. Row vectors (datapoints) must be consistent in length. Alphanumeric data is permitted. The tool `eyeds` takes in the raw input data and tries to automatically detect continuous and nominal variables:

```
$ eyeds auto-mpg.csv
8 variables, 392 datapoints
variable 1 (v1) is continuous; min = 9, max = 46.6
variable 2 (v2) is nominal; 3 4 5 6 8
variable 3 (v3) is continuous; min = 68, max = 455
variable 4 (v4) is continuous; min = 46, max = 230
variable 5 (v5) is continuous; min = 1613, max = 5140
variable 6 (v6) is continuous; min = 8, max = 24.8
variable 7 (v7) is continuous; min = 70, max = 82
variable 8 (v8) is nominal; 1 2 3
```

An output file, `auto-mpg.ds`, is created. This is also in CSV format, with a special header, where each header line is signified by '!':

```
!1,v1,v2,v3,v4,v5,v6,v7,v8
```

```
!t,C,N,C,C,C,C,C,N
!p,"9.00, 46.60","3, 4, 5, 6, 8","68.00, 455.00","46.00, 230.00",
"1613.00, 5140.00","8.00, 24.80","70.00, 82.00","1, 2, 3"
```

`!l` indicates variable labels.
`!t` indicates variable type: 'C' for continuous, or 'N' for nominal.
`!p` indicates variable parameters. If continuous, its minimum and maximum values are specified here. If nominal, each unique value is specifed.

Since the parameters were auto-detected okay, we only modify the variable names:

```
!l, mpg, cylinders, displacement, horsepower, weight, acceleration,
model year, origin
```

Note that it is possible to encode only selected values of a nominal variable, if desired: simply specify only the desired ones in the above header. For instance, if we only cared about cars having 4, 6, or 8 cylinders, we would only note these in the `!p` header, creating 3 binary variables when using 1-out-of-n encoding; these would all take the value of 0 when running into an automobile (data point) with 3 or 5 cylinders.

If we want to split the data into training/testing sets, we can use `dsplit`:

```
DSplit - randomly split datasets for training and testing
usage:  dsenc <ratio> <dataset>
options:
-r rand() seed for split
-o create file showing ordering

$ dsplit 0.6 auto-mpg.ds
```

Running this command peforms a random 60/40 training/testing split, creating `auto-mpg.ds.train` and `auto-mpg.ds.test`, which contain headers identical to `auto-mpg.ds` to preserve variable details.

We can now build a granular interface, using `gface`:

```
gface - granular interfacing
usage:  gface [options] <dataset>
options:
```

```
-s print dataset stats
-U use uniform distribution for <arg> triangular fuzzy sets (input)
-O encode nominal inputs using 1-out-of-n
-y use variable <arg> (starting from 1) as the target output
-u use uniform dist.  for <arg> triangular fuzzy sets (output)
-o encode nominal output using 1-out-of-n
-n include complements (negations) of encoded inputs
-b create interface as Espresso-compatible truth table
-i import configuration from <arg>
-x export interface configuration
```

First we need to create a binary interface:

```
$ gface -U3 -O -y1 -u3 -b auto-mpg.ds.train
conflicts:  18
```

Running the above command creates 3 fuzzy sets for each continuous input variable, encodes each nominal input variable with 1-out-of-n, and creates 3 fuzzy sets for the output, which is selected as variable 1, 'mpg'. This creates 23 inputs and 3 outputs. Using the `-b` option, the interface is binarized, and `auto-mpg.ds.train` is run through it, finding and omitting any conflicting data.

The output is `auto-mpg.ds.train.tt`, which is in the Espresso file-format. This can be run through any Boolean minimizer supporting this format. Here we'll use Espresso:

```
$ espresso auto-mpg.ds.train.tt > auto-mpg.ds.min
```

The output, `auto-mpg.ds.min`, looks like this:

```
.i 23
.o 3
.p 13
------1----1--1------1- 001
0----------1----------- 100
-0-----------1--1-----1 100
----1--------0-1------- 100
--------1-------------- 100
0--1------1-0---------- 010
----1--------1--1---0-- 100
--------------000------ 010
```

```
0------1----1---------- 100
------1----0----------- 010
-------1-0------------0 010
--------------10-----0- 010
--------0----1--0------ 010
.e
```

Here we see 13 rules, separable by binary outputs. To translate this into a fuzzy neural network structure, we use `fnngen`:

```
FNNgen - generate fuzzy neural network structure files
usage:  fnngen [options] <output filename>
options:
-e generate structure(s) from espresso file (specify filename)
-c espresso data contains (arg) continuous inputs
-f espresso data contains (arg) fuzzy sets per continuous input

$ fnngen -e auto-mpg.ds.min -c5 -f3 FNNb
```

This creates three parameter-free 3-layer logic networks (topology discussed in Chapter 3), `FNNb.1`, `FNNb.2`, and `FNNb.3`, one for each fuzzy output. `FNNb.1` is shown to demonstrate the FNN file format:

```
46 3 6 2 5
2,0 1 2
0,23 0.000 0,11 0.000

2,1 1 4
0,24 0.000 0,13 0.000 0,16 0.000 0,22 0.000

2,2 1 3
0,4 0.000 0,36 0.000 0,15 0.00000

2,3 1 1
0,8 0.000

2,4 1 4
0,4 0.000 0,13 0.000 0,16 0.000 0,43 0.000

2,5 1 3
```

```
0,23 0.000 0,7 0.000 0,12 0.000

3,0 0 6
2,0 1.000 2,1 1.000 2,2 1.000 2,3 1.000 2,4 1.000 2,5 1.000
```

In the first line:

`46` inputs, where 46 = 23 * 2, to admit negations
`3` layers
`6` is the maximum layer size
`2` indicates product is to be used for the t-norm
`5` indicates probabilistic sum for the s-norm

The rest of the file is an order-insensitive list of neurons. Looking at the first entry,

`2,0` means this neuron is located in the 2nd layer, 1st position.
These are coordinates of the neuron's position in the network, essentially a 2D matrix, where the first hidden layer is layer 1, and the output is (for this example) layer 3. Note that in this example there was no need for the first hidden layer of OR neurons for set unions (see topology in Chapter 3), so there are no neurons listed in this layer.

`1` is the type of neuron:
`1` - AND
`0` - OR

`2` indicates 2 inputs to the neuron
`0,23 0.000 0,11 0.000`
These are the inputs to the neuron, where the first two numbers are coordinates to the output of some other neuron in the network. Note that 0 is reserved for the input layer, which is why the first hidden layer is 1, as mentioned above. Following the coordinates is the weight of each input, here 0 to indicate their full relevance.

To train the networks, we must use gface to get the fuzzy data:

```
$ gface -U3 -O -y1 -u3 -x -n auto-mpg.ds.train
$ gface -U3 -O -y1 -u3 -x -n auto-mpg.ds.test
```

Like before, running the above command creates 3 fuzzy sets for each continuous input variable, encodes each nominal input variable with 1-out-of-n, and creates 3 fuzzy sets for the output, which is selected as variable 1, mpg. We don't use the `-b` option (which induces binary sets), but specify -n because we need negations for the FNNs we created. The -x option was specified here for illustrative purposes, outputting an interface configuration in EYE format:

```
3, input, fuzzy, 3, 132.5000,261.5000,390.5000
4, input, fuzzy, 3, 76.6667,138.0000,199.3333
5, input, fuzzy, 3, 2200.8333,3376.5000,4552.1667
6, input, fuzzy, 3, 10.8000,16.4000,22.0000
7, input, fuzzy, 3, 72.0000,76.0000,80.0000
2, input, 1-out-of-n
8, input, 1-out-of-n
0, output, fuzzy, 3, 9.0000,27.8000,46.6000
```

Here we see each variable number, distinguished as an input or output, encoding type, and (if fuzzy) the number of sets followed by the locations of their modal values (peaks). This could be modified and then utilized by importing the file with the -i option.

The gface command creates the encoded data (auto-mpg.train.enc) in a format accepted by fnntrain, employing gradient-based optimization:

```
FNNtrain - gradient-based optimization of a fuzzy neural network
usage:  fnntrain [options] <encoded data> <network1> <network2> ...
options:
-v verbose (shows training progress)
-e specify maximum number of learning epochs (default:  1000)
-r learning rate (default:  0.01)
-s rand() seed
-o create output files for training progress and final network
(arg:  output file extension)

$ fnntrain -o trained auto-mpg.ds.train.enc FNNb.1 FNNb.2 FNNb.3
```

The result is a combined and trained (parameterized) network, FNN.trained. Using fnntest we can measure performance:

```
FNNtest - fuzzy neural network simulation
usage:  fnnsim [options] <encoded data> <network1> <network2> ...
options:
-p prune with given threshold
-v verbose when processing (show network output and target)
```

```
$ fnnsim -D auto-mpg.ds.train.enc FNN.trained
SSE = 2120.05
Avg.  Error = 2.2297
RMSE = 3.00358

$ fnnsim -D auto-mpg.ds.test.enc FNN.trained
SSE = 1980.67
Avg.  Error = 2.65576
RMSE = 3.55186
```

`fnnsim` could also be used with the original binary FNNs in a similar way:

```
$ fnnsim -D auto-mpg.ds.train.enc FNNb.1 FNNb.2 FNNb.3
SSE = 4356.81
Avg.  Error = 3.31679
RMSE = 4.30577
```

The `-v` option can be used to show results for each datapoint, useful for thing such as generating scatter plots. The networks can be pruned using the `-p` option to reduce its size. Rules can be directly interpreted from the `FNN.trained` file, or optionally run through the `fnnrules` tool to generate an HTML table showing the quantified rule-base, similar to the tables presented in Chapter 3. Finally, all of this functionality may be automated through the use of the eye3 shell script, which permits easy n-fold cross-validation.