THEORY AND DESIGN OF A HYBRID PATTERN RECOGNITION SYSTEM

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
John A. Drakopoulos
May 1995

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Barbara Hayes-Roth
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

David E. Rumelhart

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Nils J. Nilsson

Approved for the University Committee on Graduate Studies:

Abstract

Pattern recognition methods can be divided into four different categories: statistical or probabilistic, structural, possibilistic or fuzzy, and neural methods. A formal analysis shows that there is a computational complexity versus representational power trade-off between probabilistic and possibilistic or fuzzy set measures, in general. Furthermore, *sigmoidal theory* shows that fuzzy set membership can be represented effectively by sigmoidal functions. Those results and the formalization of sigmoidal functions and subsequently multi-sigmoidal functions and neural networks led to the development of a hybrid pattern recognition system called tFPR.

tFPR is a hybrid fuzzy, neural, and structural pattern recognition system that uses fuzzy sets to represent multi-variate pattern classes that can be either static or dynamic depending on time or some other parameter space. Given a set of input data and a pattern class specification, tFPR estimates the degree of membership of the data in the fuzzy set that corresponds to the current pattern class. The input data may be a number of time-dependent signals whose past values may influence the evaluation of the pattern class.

The membership functions of the fuzzy sets that represent pattern classes are modeled in three different ways. In case of relatively simple pattern classes or pattern classes that can be described concisely by a fuzzy set expression, the membership functions of the corresponding fuzzy sets would be modeled by a collection of sigmoidal functions. The choice of sigmoidal functions was motivated by their ability to represent efficiently and concisely different multi-variate pattern classes via fuzzy set membership. However, when the pattern class under question would depend on some parameter space (such as time) a structural pattern recognition method (that may involve fuzzy components) would be employed in order to match curves (rather than points) in the input domain. Finally, whenever it would be difficult to obtain a formal definition of the membership function of a fuzzy set representation for a pattern class, tFPR would model the membership function via multi-sigmoidal neural networks. Thus, tFPR should be able to learn a definition of the membership function by input-output relationships given to the system as training data.

Although efficiency is a very important consideration in tFPR, the main issues are knowledge acquisition and knowledge representation (in terms of pattern class descriptions). The fuzzy and structural components of tFPR have been implemented in Lisp, while the neural component has been implemented in C using the SNNS simulator. tFPR has been embedded in the BB1 blackboard architecture but it can also run as a stand-alone system. It is currently being applied in a system for medical monitoring.

Acknowledgements

I would like to thank my advisor, Barbara Hayes-Roth, for her continuing support and guidance over the years I have spent in her research group. Her trust and support on me even during my most difficult years at Stanford made this research a reality.

I would also like to thank my other committee members, David Rumelhart and Nils Nilsson as well as the additional members of my defense committee, David Gaba and Ousama Khatib for their time and interest.

Special thanks are due to the rest of the BB1 and AIS research group members which over the years have listened to my talks and have offered me valuable advice.

I would also like to pay special tribute to the administrative staff at KSL for their excellent service, help, and support.

This research was supported by NASA grant NAG2-581 (ARPA Order #8607) and Teknowledge Federal Systems, Inc. contract 71715-1 (ARPA contract DAAA21-92-C-0028).

Contents

List of Figures

Chapter 1

Introduction

Notions of quantity are possible only when there already exists a general concept which admits particular instances.

On the hypotheses which lie at the foundations of Geometry
Georg F.B. Riemann 1826–1866

For the man who should loose me is dead,
Fighting with the Duke in Flanders,
In a pattern called a war.
Christ! What are patterns for?

Patterns
Amy Lowell 1856–1943

Pattern recognition is defined in [Uhr, 1973] as "the many-to-one mapping from a very large set of possible pattern instances to a much smaller set of pattern names." It is a form of information reduction that provides us with a concise description of a state or situation that is represented by a large amount of input data that are often only partially available. That description, in turn, may signal the satisfaction of some conditions or the existence of some features in the current state that require a change in the system's behavior. Hence, pattern recognition is an essential ingredient of tasks such as perception, decision making, reaction, design, and planning.

1.1 Contributions

This thesis describes the theoretical foundations that led to the design of a hybrid fuzzy, neural, and structural pattern recognition system called tFPR as well as the functional structure of the system itself. The theoretical results consist of the following:

- An analytic comparison of some measure classes (namely probabilities, possibilities, and fuzzy sets).
- The development of sigmoidal theory which formalizes and studies sigmoidal functions and defines multi-sigmoidal functions and neural networks. Sigmoidal theory also shows that sigmoidal functions can be coupled very effectively with possibilities and fuzzy sets.
- Some results regarding the representational capabilities of two-layer multi-sigmoidal neural networks and a learning algorithm for multilayer multi-sigmoidal neural networks.

Practical contributions and empirical studies include the following:

- An empirical evaluation of multi-sigmoidal neural networks.
- The development of a greedy heuristic algorithm for a form of structural pattern matching.
- The development of a small data base containing descriptions of more than one hundred medical pattern classes.

1.2 A guide to the thesis

In this chapter, an overview of the basic pattern recognition paradigms is presented. In chapter 2, a formal analysis and comparison of the most common measures (namely probabilities, possibilities, and fuzzy sets) used in pattern recognition systems today is presented. In chapter 3, the sigmoidal theory is introduced which formally defines and analyzes sigmoidal and multi-sigmoidal functions and shows that those functions can represent fuzzy set membership very effectively.

In chapter 4 multi-sigmoidal neural networks are presented. Those networks are a straightforward generalization of ordinary sigmoidal neural networks by sigmoidal theory.

A preliminary analytic study of two-layer multi-sigmoidal neural networks and an empirical evaluation of multi-layer multi-sigmoidal neural networks are presented in that chapter, too.

Chapters 2 through 4 define the theoretical foundations of tFPR which is the subject of chapter 5. A definition of the system is given along with a description of its greedy heuristic used for structural pattern matching. Finally, current and future applications of tFPR are discussed.

1.3 Overview of Basic Pattern Recognition Paradigms

Currently there are so many pattern recognition approaches and methods that their complete taxonomy is a difficult and arduous task. In most of those approaches, the input can be described or be transformed into a number n of computable properties (often called *features*) and as such can be represented as a point in an n-dimensional space (which is often called *feature space*). Subsequently, pattern classes correspond to sets of input data and can be described by the boundaries of their corresponding regions in feature space. Those boundaries are usually called *decision boundaries*. Hence, pattern recognition can be regarded as the process of establishing the decision boundaries of a number of pattern classes.

Following the above interpretation, a classification of pattern recognition methods and techniques appears in [Lippmann, 1989] based on the way decision regions are formed (e.g. probabilistic, receptive fields, exemplars, etc.) and the form of the decision boundaries (i.e. hyperplanes, hyper-spheres, quadratic surfaces, etc.).

On the other hand, a number of other properties of classifiers which could be taken into account in classifying pattern recognition methods are mentioned in [Jain, 1987]. For example, a classifier can be statistical or structural, parametric or non-parametric, and supervised or unsupervised. Finally, a more appealing taxonomy of pattern recognition methods appears in [Schalkoff, 1992] where classifiers are divided into categories based on their computational nature. That taxonomy consists of three different classes of pattern recognition methods: statistical, structural and neural pattern recognition methods.

Here, we describe a taxonomy that is based on the form of the measures used internally by the classifier and the nature of their elementary computations, more or less along the lines of the taxonomy presented in [Schalkoff, 1992]. Subsequently, within each category, classifiers can be partitioned into classes based on other properties such as the form of their decision regions, their parametric or non-parametric formalization, and their supervised or unsupervised learning characteristics.

Our taxonomy consists of four categories:

1. Statistical pattern recognition methods (also known as stochastic, decision-theoretic, or probabilistic)
2. Structural or syntactic pattern recognition methods
3. Fuzzy or possibilistic pattern recognition methods
4. Neural network or connectionist pattern recognition methods

The first category contains the most conventional classifiers that are based on some statistical or decision-theoretic measures to classify their input data. A good description and introduction to most of those techniques appear in early classic textbooks on pattern recognition [Duda and Hart, 1973, Fukunaga, 1972]. A more recent textbook with a good description of many statistical pattern recognition methods is [Schalkoff, 1992].

Structural pattern recognition methods differ from all other pattern recognition methods for they do not directly form decision regions in some feature space. Instead, they try to decompose their input data into a number of smaller pieces that match some *primitive pattern elements* or *subpatterns*. Those subpatterns are then combined hierarchically to form other higher level subpatterns of the overall pattern class. Most of structural pattern recognition methods are presented in [Fu, 1982, Schalkoff, 1992]. Despite the conceptual differences of statistical and structural pattern recognition methods, a unifying view is given in [Fu, 1980].

Fuzzy or possibilistic pattern recognition methods are based on possibilistic measures to define pattern classes. Fuzzy sets were invented in 1965 by Zadeh [Zadeh, 1965]. Since then, fuzzy set theory has received increasing attention which has resulted in a number of applications in many different domains and disciplines. Fuzzy sets naturally fit pattern recognition problems. In the most general case, those problems involve the computation of the degree to which a data object x matches a pattern class description p. In the fuzzy set interpretation of those problems, p is usually represented by a fuzzy set A_p which, in general, models a cluster, class, or pattern object. The grade of membership of x in A_p represents the degree to which x matches pattern class p. An original study of fuzzy sets in the context of pattern recognition problems appeared in [Bellman *et al.*, 1966]. An early and inspiring application of fuzzy sets to pattern recognition problems and *clustering* in particular, has been presented in [Ruspini, 1969, Ruspini, 1970]. A subsequent and very influential method was the fuzzy ISODATA (or fuzzy c-means) algorithm [Dunn, 1973, Bezdek, 1973]. Since then a number of other methods and techniques have been developed and used in applications. A good description of most of those techniques appears in [Bezdek, 1981, ch.3-5], [Kandel, 1982, ch.3-5], [Pal and Dutta-Mazamder, 1986, ch.3, 5-8]. More recent collections of fuzzy pattern recognition methods appear in [Dubois *et al.*, 1993, ch.6], and [Bezdek and Pal, 1992].

Although the first and very influential works on neural network research appeared about fifty years ago [McCulloch and Pitts, 1943, Hebb, 1949] and some early and very important developments took place in late 1950s and early 1960s [Rosenblatt, 1958, Rosenblatt, 1961, Widrow, 1962, Nilsson, 1965] the field of neural networks as a computational paradigm has grown rapidly only in the last few years (see [Hecht-Nielsen, 1990, chapter 1] for an overview of the history of neurocomputing). Neural network pattern recognition methods, which are based on neural networks to define and describe pattern classes in the input data, also have received a proportional amount of interest in the last few years. The measures used are usually (classification error) metrics sometimes coupled with measures of generalization or network size (such as Rissanen's minimum descriptive length) [Rumelhart *et al.*, 1986a, Hinton, 1989, Hanson and Pratt, 1989, Mozer and Smolensky, 1989]. The basic computational element is a *unit* or *node* that performs a relatively simple function and communicates with other units through connections of variable strength (or *weights*). Usually, each unit maps an n-dimensional input vector into a real value x using a projection operator (such as *dot product*, $\sum$, sup, *Euclidean distance*, etc.) and then computes its output by applying a simple non-linear function to x. Knowledge is usually stored in the weights while the overall computation emerges through the interactions of the units.

For classification problems, the weights specify the boundaries of regions in feature space. Depending on the form of the projection operators used, neural network classifiers may use hyperplanes, hyperspheres, or combinations of them with other multi-dimensional isopotential surfaces in order to form regions in feature space.

Early neural network approaches were focused in linear discriminating functions and, in particular perceptrons ([Duda and Hart, 1973, Nilsson, 1965, ch. 5]). Now there is a wide range of neural network architectures and learning algorithms that find applications in many different domains and disciplines. Most of those algorithms are described in neural network textbooks [Haykin, 1994, Hecht-Nielsen, 1990, Hertz *et al.*, 1991, Rumelhart *et al.*, 1986b].

A good introduction to pattern recognition techniques and algorithms of all of the above categories, except fuzzy methods, is given in [Schalkoff, 1992]. Of course there are hybrid methods that could belong to more than one of those categories. For example, concepts originated in fuzzy set theory have been incorporated into neural network methods and neural networks have been used in fuzzy set implementation and fuzzy reasoning [Cohen and Hudson, 1992, Keller *et al.*, 1992, Takagi and Hayashi, 1991, Takagi, 1990]. Furthermore, fuzzy neural networks and hybrid fuzzy and neural classification systems have been developed. Those include the nested generalized exemplar (NGE) algorithm [Salzberg, 1990], the fuzzy min-max classifier (FMMC) [Simpson, 1992], the fuzzy ARTMAP neural network architecture [Carpenter *et al.*, 1992], and fuzzy multilayer perceptrons and neurons [Pal and Mitra, 1992, Yamakawa and Furukawa, 1992]. Furthermore, hybrid probabilistic and neural network methods exist such as neural implementations of the k-nearest neighbor classifier and perceptron trees [Lippmann, 1989, Utgoff, 1988]. In addition, a pattern recognition system or architecture may use many different pattern recognition methods and so be a hybrid system.

Chapter 2

A Comparison of Some Measure Classes

Probable impossibilities are to be preferred to improbable possibilities.

24.1460a. Aristotle 384–322 B.C.

How often have I said to you that when you have eliminated the impossible, whatever remains, however improbable, *must be the truth?*

Sir Arthur Conan Doyle 1859–1930.

Probabilities, possibilities, and fuzzy sets are all measures that find applications in pattern recognition problems and have been used to formalize and quantify uncertainty. There is an on-going debate regarding the usefulness of each measure. Arguments in favor of probabilities can be found in [Lindley, 1987, Cheeseman, 1986] while arguments more in favor of possibilities and fuzzy sets are presented in [Kosko, 1990, Klir, 1989].

A brief presentation and qualitative comparison of the above measures as well as MYCIN's certainty factors ([Shortliffe, 1976]) and Dempster-Shafer theory ([Dempster, 1968, Shafer, 1976]) appear in [Henkind and Harrison, 1988]. However, a quantitative comparison of those measures in terms of both efficiency and expressiveness is not given. Such a comparison is necessary in order to evaluate and characterize those measures. Some attempts in this direction regard a notion of *consistency* between probabilities and possibilities [Dubois and Prade, 1983, Delgado and Moral, 1987] and transformations from probabilities and possibilities to Dempster-Shafer theory [Klir and Parviz, 1992]. However, those transformations cannot tell us about the exact relationship of probabilities and possibilities. Furthermore, the transformations from possibilities to Dempster-Shafer theory as appear in [Klir and Parviz, 1992] require the elements of the universal set to be ordered in descending possibility values. However, such an ordering does not always exist when the universal set is infinite. This limits the applicability of the transformation to finite universal sets. Finally, there has been no study so far of the extensions of the universal sets that are necessary in order to create mappings among probabilities, possibilities, and fuzzy sets.

Here, we present a number of theorems that specify some important relationships between probabilities, possibilities, and fuzzy sets and as such provide some answers to the issues mentioned above. The theorems use some formal definitions of the above measures. However, since there is a wide variability in the fuzzy set theory literature in terms of definitions, the results presented here should be construed as applicable only in cases where the definitions used elsewhere are either equivalent or imply ours. On the other hand, it has been proved in [Bellman and Giertz, 1973, Hamacher, 1976] that, under very reasonable assumptions, the only truth functional connectives for fuzzy sets that are possible are the usual max$\Leftrightarrow$min (or equivalently sup$\Leftrightarrow$inf which are used here). Thus our definitions correspond to the most conventional case which is (most probably) most widely used.

The rest of this chapter is organized along the general scheme of some preliminary definitions followed by a number of theorems and a discussion of their implications to systems, architectures, and applications. In the next section, basic definitions are introduced. These set up the context under which our theorems are presented and consist the foundations of probability, possibility, and fuzzy set theory. In section 2.2, we present a number of theorems that relate those theories and compare them from the viewpoint of *relative expressiveness* i.e. the ability of one to simulate the others. For infinite domains, it is proved that all measures have the same expressiveness and as such can be considered to be equivalent in terms of representational power. However, for finite domains, probabilities are proved to be more expressive than both possibilities and fuzzy sets but they have higher computational demands. Possibilities and fuzzy sets can actually simulate probabilities but, in that case, their space requirements are exponential when compared to those of probabilities.

The increased complexity of probabilities gives rise to a trade-off which is the subject of section 2.3. This is a trade-off of complexity and representational power versus efficiency.

Experimental systems and approaches as well as different applications are discussed.

2.1 Mathematical Measures, Systems, and System Classes

A formal definition of probability, possibility, and fuzzy set theory is given below. We begin with the definition of a *sigma-algebra*[1] and its *base*:

Definition 1 *A sigma-algebra $\mathcal{B}$ on a set Ω is a set of subsets of Ω satisfying the following properties:*

$$\begin{array}{ll} (a) & \Omega \in \mathcal{B} \\ (b) & \forall A \; (if \; A \in \mathcal{B} \; then \; \bar{A} \in \mathcal{B}), \quad \bar{A} \textit{ stands for the complement of } A \\ (c) & \textit{if } A_1, A_2, \ldots \textit{ is any sequence of sets in } \mathcal{B} \textit{ then } \bigcup_i A_i \in \mathcal{B} \end{array}$$

Definition 2 *A set α is a base of a sigma-algebra $\mathcal{B}$ iff the closure of α under set union and set complementation (wrt the universal set Ω of $\mathcal{B}$) is equal to $\mathcal{B}$ (written as $c(\alpha) = \mathcal{B}$) and no proper subset of α has this property.*

Now, we define define a *probability system* and a *probability measure*[2] as follows:

Definition 3 *A probability system is a triple, $(\Omega, \mathcal{B}, P)$, where Ω is an arbitrary set (the set of all possible outcomes), $\mathcal{B}$ is a sigma-algebra on Ω (the set of the events of interest) and P is a real valued function defined for each $A \in \mathcal{B}$ such that:*

$$\begin{array}{ll} (a) & \forall A \in \mathcal{B} \quad 0 \leq P(A) \leq 1 \\ (b) & P(\Omega) = 1 \\ (c) & \textit{if } A_1, A_2, \ldots \textit{ is any sequence of pairwise disjoint sets in } \mathcal{B} \textit{ then} \\ & \quad P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i) \end{array}$$

A function P that satisfies the three conditions above is called a *probability measure.* The third property is called *countable additivity.* For a discussion of sigma-algebras and probability measures see [Papoulis, 1991].

Similarly, we define a *possibility system*, and a *possibility measure* ([Dubois and Prade, 1987]):

[1]A sigma-algebra is also called *event class*, or *sigma-field.*
[2]Originally defined by A.N.Kolmogorov [Kolmogorov, 1950].

Definition 4 *A possibility system is a triple,* $(\Omega, \mathcal{B}, \Pi)$*, where* Ω *is an arbitrary set (the set of all possible outcomes),* $\mathcal{B}$ *is a sigma-algebra on* Ω *(the set of the events of interest) and* Π *is a real valued function defined for each* $A \in \mathcal{B}$ *such that:*

$$\begin{array}{ll} (a) & \Pi(\emptyset) = 0 \\ (b) & \Pi(\Omega) = 1 \\ (c) & \forall I \quad \textit{if } \{A_i \ / \ i \in I\} \textit{ is a set of sets in } \mathcal{B} \textit{ then} \\ & \quad \Pi(\bigcup_{i \in I} A_i) = \sup_{i \in I} \Pi(A_i) \end{array}$$

Now the function Π is a possibility measure.

Fuzzy sets were defined in 1965 by L.A.Zadeh [Zadeh, 1965]. The basic idea in *fuzzy set theory* is the notion of a fuzzy set which is the mathematical concept of set that allows partial membership for elements in its domain. Thus, an element x may partially belong to a fuzzy set A. Each fuzzy set is characterized by its membership function $f_A(x)$ that states, for any given x in the domain of A, "in what degree, x belongs to A".[3]

A formal definition of a *fuzzy set system* is as follows:

Definition 5 *A fuzzy set system is a triple,* (β, D, f)*, where* β *is a set of all fuzzy sets of interest,* D *is the domain of the fuzzy sets in* β*, and* f *is a real valued function defined for each* $A \in \beta$ *and each* $x \in D$ *such that:*

$$\begin{array}{ll} (a) & \forall x \in D \ \ f_{\emptyset}(x) = 0, \quad \emptyset \textit{ is the empty set in } \beta \\ (b) & \forall x \in D \ \ f_{\Omega}(x) = 1, \quad \Omega \textit{ is the universal set in } \beta \\ (c) & \forall I \quad \textit{if } \{A_i \ / \ i \in I\} \textit{ is a set of fuzzy sets in } \beta \textit{ then} \\ & \quad \forall x \in D \quad f_{\bigcup_{i \in I} A_i}(x) = \sup_{i \in I} f_{A_i}(x) \\ & \quad \forall x \in D \quad f_{\bigcap_{i \in I} A_i}(x) = \inf_{i \in I} f_{A_i}(x) \\ (d) & \forall A \in \beta \ \ \forall x \in D \quad f_{\bar{A}}(x) = 1 \Leftrightarrow f_A(x) \end{array}$$

[3]In addition, Zadeh defined fuzzy set subset-hood based on membership functions:

$$\forall A, B \ (A \subseteq B \ \Leftrightarrow \ f_A \leq f_B)$$

where $f_A \leq f_B$ means that $\forall x \ (f_A(x) \leq f_B(x))$. However, it should be noted that the above definition of fuzzy set subsethood is not universally accepted within the fuzzy set community and is often against the meaning of "if A then B".

Now observe that parts (c), (d) in the above definition should not be seen as requirements on the function f but rather as definitions of fuzzy set union, intersection and complementation. For example, the union of a sequence of fuzzy sets $A_1, A_2, \ldots$ is defined to be a fuzzy set whose membership function is equal to $\sup_{i=1}^{\infty} f_{A_i}$.

Furthermore, we can define the closure $c(\beta)$ of β recursively as follows

$$\begin{array}{l} \forall A \in \beta \quad A \in c(\beta) \\ \forall I \quad \text{if } \{A_i \ / \ i \in I\} \text{ is a set of fuzzy sets in } c(\beta) \text{ then} \\ \qquad \bigcup_{i \in I} A_i \in c(\beta) \end{array}$$

Intuitively, $c(\beta)$ is the fuzzy analog of sigma-algebras as defined for ordinary sets. Pushing this analogy further, if β has the property that for no proper subset β' of β would be $c(\beta') = c(\beta)$ then β will be called a *base* of $c(\beta)$. Alternatively, we could have defined fuzzy sigma-algebras to be closures of sets of fuzzy sets such as β above. A more detailed discussion of possibilities and fuzzy sets appears in [Drakopoulos, 1994a].

Furthermore, we define system classes to be classes of different systems. Hence, we define:

Definition 6 *A probability system class* $\mathbf{P}_\Omega$ *is the class of all probability systems* $(\Omega, 2^\Omega, P)$. *A possibility system class* $\mathbf{\Pi}_\Omega$ *is the class of all possibility systems* $(\Omega, 2^\Omega, \Pi)$. *A* fuzzy set system class $\mathbf{F}_{\beta,D}$ *is the class of all fuzzy set systems* (β, D, f).

A final issue regards the *extensionality* or *intensionality* of a measure. Formally, a measure F is called *extensional* iff for any $A_1, A_2, \ldots$ the value of $F(\bigcup_{i=1}^{\infty} A_i)$ can be computed from the values of $F(A_1), F(A_2), \ldots$ without any direct reference to $A_1, A_2, \ldots$. Otherwise, F would be called *intensional.* For example, probability measures are intensional for $P(\bigcup_{i=1}^{\infty} A_i)$ depends not only on $P(A_1), P(A_2), \ldots$ but also on whether the sets $A_1, A_2, \ldots$ are disjoint or not. On the other hand, possibility measures are extensional for $\Pi(\bigcup_{i=1}^{\infty} A_i) = \sup_{i=1}^{\infty} \Pi(A_i)$. Similarly, fuzzy sets membership is an extensional measure. The above definition can be generalized to extensional and intensional systems [Pearl, 1990]. The actual trade-off has been claimed to be between computational efficiency and semantic clarity or content [Pearl, 1990]. In the extreme case, one can define an extensional measure that is trivial to compute but carries little or no information. On the other hand, one can define an intensional measure that carries a lot of information but is difficult to compute. However, this is mostly an empirical observation and no mathematical proofs have been given in the literature to establish it. Nevertheless, in this thesis, we present theorems that show that for the particular intensional and extensional measures studied (i.e. probabilities, possibilities, and fuzzy sets) there is a trade-off between computational efficiency and representational power, when the domains of the measures are finite. For infinite domains, possibilities and fuzzy sets have an advantage in terms of computational complexity while no measure has an advantage over the others in terms of representational power as defined in section 2.2.

However, it should be noted that, in addition to the classical sup$\Leftrightarrow$inf pair, several algebraic structures on the interval $[0,1]$ have been defined and studied in the literature and consequently have been used to define fuzzy set connectives (see [Alsina *et al.*, 1983, Alsina and Trillas, 1983, Hamacher, 1976, Yager, 1980]). Those alternative definitions relax some of the requirements on the connectives (mainly distributivity) so that the definitions are consistent.

On the other hand, as proved in [Bellman and Giertz, 1973, Hamacher, 1976], under very reasonable assumptions (mainly distributivity and monotonicity) the classical sup$\Leftrightarrow$inf pair is not only natural but also the only one possible. Thus, our definition here, which is based on the sup$\Leftrightarrow$inf connectives and is equivalent to Zadeh's original definition ([Zadeh, 1965]), corresponds to the most natural and most widespread definition used in the fuzzy set research community. Nevertheless, it is worth noting that our theorems are intended for the classical connectives for fuzzy sets and do not necessarily apply for other connectives.

2.2 System Relationships

An important question is whether probability and possibility measures or systems (fuzzy or not) are actually different. A number of theorems presented below prove that they have some strong similarities as well as differences. To study them formally, we need to define the notion of *relative expressiveness* of one system with respect to another. In the following, we write $(\Omega, \mathcal{B}, M)$ to indicate a system where Ω is an arbitrary set, $\mathcal{B}$ is a sigma-algebra on Ω, and M is a real valued function defined for each $A \in \mathcal{B}$. Furthermore, we write 2^A to indicate the power set of A, for any set A.

Now, we define:

Definition 7 *A system* $\mathbf{S} = (\Omega, \mathcal{B}, M)$ *is at most as expressive as a system* $\mathbf{S}' = (\Omega', \mathcal{B}', M')$

(written as $\mathbf{S} \leq \mathbf{S}'$*) iff*

$$\forall A \in \mathcal{B} \;\exists A' \in \mathcal{B}' \quad M(A) = M'(A').$$

The intuition behind this definition is that if $\mathbf{S} \leq \mathbf{S}'$ then the system $\mathbf{S}'$ can simulate $\mathbf{S}$ by using $M'(A')$ whenever $\mathbf{S}$ uses $M(A)$. Of course, this raises the question about the existence of a deterministic algorithm or method that could derive A' given A, M, and the system class to which M' belongs. Our definition above requires only the establishment of the existence of a set A'. However, in [Drakopoulos, 1994a], constructive proofs of the theorems presented here are given that determine how to construct not only A' but also Ω' and M' given $A, M\Omega$, and the system class of M'.

Now, similarly as before, we define:

Definition 8 *A system class* $\mathcal{C}_\Omega$ *is at most as expressive as the system class* $\mathcal{C}'_{\Omega'}$ *(written as* $\mathcal{C}_\Omega \leq \mathcal{C}'_{\Omega'}$*) iff*

$$\forall M \quad (\Omega, 2^\Omega, M) \in \mathcal{C}_\Omega \Rightarrow \exists M' \left(\begin{array}{ll} (\Omega', 2^{\Omega'}, M') \in \mathcal{C}'_{\Omega'} & \\ & and \\ (\Omega, 2^\Omega, M) \leq (\Omega', 2^{\Omega'}, M') & \end{array} \right)$$

Therefore, $\mathcal{C}_\Omega \leq \mathcal{C}'_{\Omega'}$ iff each system in $\mathcal{C}_\Omega$ can be simulated by a system in $\mathcal{C}'_{\Omega'}$.

Similarly, for fuzzy set systems we define:

Definition 9 *A system* $\mathbf{S} = (\Omega, 2^\Omega, M)$ *is at most as expressive as a fuzzy set system* $\mathbf{S}' = (\beta, D, f)$ *(written as* $\mathbf{S} \leq \mathbf{S}'$*) iff*

$$\forall A \in 2^\Omega \;\exists A' \in c(\beta) \;\exists x \in D \quad M(A) = f_{A'}(x).$$

Furthermore, $\mathbf{S}' \leq \mathbf{S}$ *iff*

$$\forall A' \in c(\beta) \;\forall x \in D \;\exists A \in 2^\Omega \quad M(A) = f_{A'}(x).$$

In the case $\mathbf{S} \leq \mathbf{S}'$, the fuzzy set system $\mathbf{S}'$ can simulate the system $\mathbf{S}$ while in the case $\mathbf{S}' \leq \mathbf{S}$ the system $\mathbf{S}$ can simulate the fuzzy set system $\mathbf{S}'$. Now, we define:

Definition 10 *A system class* $\mathcal{C}_\Omega$ *is at most as expressive as the system class* $\mathbf{F}_{\beta,D}$ *(written as* $\mathcal{C}_\Omega \leq \mathbf{F}_{\beta,D}$*) iff*

$$\forall M \quad (\Omega, 2^\Omega, M) \in \mathcal{C}_\Omega \Rightarrow \exists f \left(\begin{array}{ll} (\beta, D, f) \in \mathbf{F}_{\beta,D} & \\ & and \\ (\Omega, 2^\Omega, M) \leq (\beta, D, f) & \end{array} \right)$$

Similarly, $\mathbf{F}_{\beta,D} \leq \boldsymbol{\mathcal{C}}_\Omega$ *iff*

$$\forall f \quad (\beta, D, f) \in \mathbf{F}_{\beta,D} \Rightarrow \exists M \left(\begin{array}{l} (\Omega, 2^\Omega, M) \in \boldsymbol{\mathcal{C}}_\Omega \\ (\beta, D, f) \leq (\Omega, 2^\Omega, M) \end{array} \quad and \right)$$

In the following, we denote the cardinal of a set A as $card(A)$ and the cardinal of the set $\mathcal{N}$ of natural numbers as $\aleph_0$ (see [Stoll, 1963] for more details on cardinal numbers).

Now, we can present a few second order logic theorems that relate probability, possibility, and fuzzy set systems. Their proofs have been given in [Drakopoulos, 1994a].

Theorem 1 *If* Ω *is finite then* $\mathbf{\Pi}_\Omega \leq \mathbf{P}_\Omega$

Theorem 2 $\forall \Omega \quad \mathbf{P}_\Omega \leq \mathbf{\Pi}_{2^\Omega}$

Theorem 3 *If* Ω' *is finite and* $card(\Omega) + 1 < card(2^{\Omega'})$ *then not*$(\mathbf{P}_{\Omega'} \leq \mathbf{\Pi}_\Omega)$

Theorem 4 *If* Ω *is infinite then* $\mathbf{P}_\Omega = \mathbf{\Pi}_\Omega$*, (i.e.* $\mathbf{\Pi}_\Omega \leq \mathbf{P}_\Omega$ *and* $\mathbf{P}_\Omega \leq \mathbf{\Pi}_\Omega$ *).*
Furthermore, if β *is infinite or* D *is uncountably infinite then* $\mathbf{P}_\Omega = \mathbf{F}_{\beta,D}$ *(i.e.* $\mathbf{P}_\Omega \leq \mathbf{F}_{\beta,D}$ *and* $\mathbf{F}_{\beta,D} \leq \mathbf{P}_\Omega$ *).*

Theorem 5 $\forall \Omega, \beta \quad \mathbf{P}_\Omega \leq \mathbf{F}_{\beta,2^\Omega}$ *(assuming* $card(\beta) \geq 3$*).*
$\forall \Omega, D \quad \mathbf{\Pi}_\Omega \leq \mathbf{F}_{\beta,D}$ *(assuming* D *is non-empty and* $card(\beta) \geq card(\Omega)$ *).*

Theorem 6 *If* $card(\Omega) \geq card(\beta) \cdot card(D)$ *and* $card(\beta) < \aleph_0$ *then* $\mathbf{F}_{\beta,D} \leq \mathbf{P}_\Omega$

Theorem 7 *If* Ω, β, D *are finite and both* $card(\beta)$ *and* $card(D)$ *are polynomially related to* $card(\Omega)$ *then not*$(\mathbf{P}_\Omega \leq \mathbf{F}_{\beta,D})$

The previous few theorems show that probabilities are more expressive (or equivalently have more representational power) than both possibilities and fuzzy sets, for finite domains Ω. However, as theorem 4 shows, their differences in expressiveness disappear when the domain is infinite. Nevertheless, it should be noted that the case of finite domains is of particular interest in computer science where only a finite subset of the rational numbers is representable in our finite machines. Furthermore, the case of finite domains does not cause an "overflow" of the range of the above measures allowing for interesting relationships among them to be formed. It is those relationships that reveal the differences between the

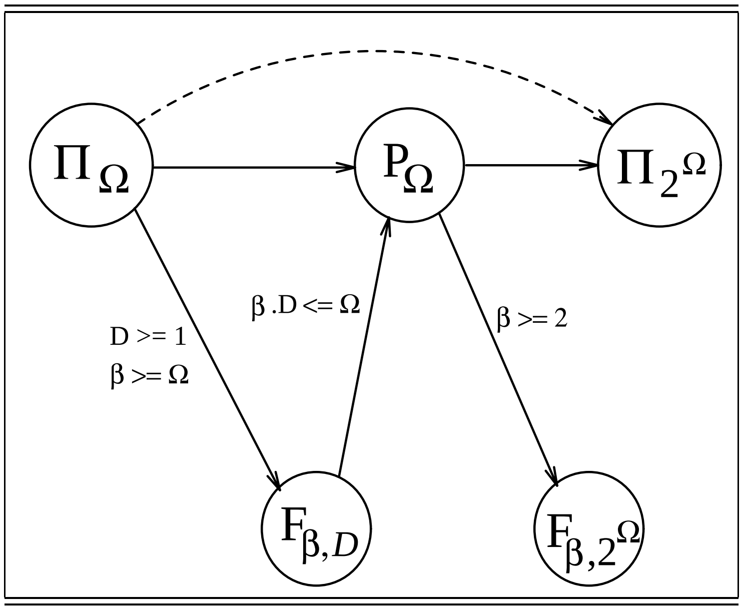


Figure 2.1: Expressiveness relations among probabilities, possibilities, and fuzzy sets.

two mathematical operators ($\sum$ and sup) that define the fundamental operations in each system class examined in this study.

Figure 2.1 represents our results graphically. The nodes in the graph represent system classes and the links represent "at most as expressive as" relations. The labels on some of the arcs indicate special conditions under which the corresponding "$\leq$" relations hold. The meanings of the labels are as follows:

Label	*Meaning*
$\beta >= 3$	$card(\beta) \geq 3$
$D >= 1$	$card(D) \geq 1$,
$\beta >= \Omega$	$card(\beta) \geq card(\Omega)$
$\beta.D <= \Omega$	$card(\beta) \cdot card(D) \leq card(\Omega)$

We assume that the sets Ω, β, D are finite. The dashed line is a link that has not been proved explicitly in any of the previous theorems but can be derived easily from the following:

$$card(\Omega) \leq card(\Omega') \Rightarrow \mathcal{C}_\Omega \leq \mathcal{C}_{\Omega'}$$

2.3 Discussion

Theorems 1 and 3 of the previous section showed that, for finite domains, possibilities are less expressive than probabilities. Another way to express that is that probabilities can simulate possibilities without any extra space requirements while the opposite is not true, in general.[4] Theorem 2 reveals that when extra space requirements can be greatly tolerated then probabilities can be simulated by possibilities, too. However, as theorems 1 and 3 showed, for any given finite space, probabilities have strictly greater representational power than possibilities. Thus, probabilities, when compared to possibilities, can represent more mappings from any given finite domain to the interval $[0,1]$ and as such can be regarded as being "denser" than possibilities in terms of representable mappings.

Unfortunately, the space requirements to simulate probabilities by possibilities are excessive since the size of 2^{Ω} is exponentially larger than the size of Ω. Therefore, in practice, we may regard that probabilities cannot be simulated by possibilities when the domain is finite.

Theorems 5, 6, and 7 reveal that a similar relation exists between probabilities and fuzzy sets, in finite spaces. Probabilties can simualte fuzzy sets without any extra space requirements: they both require $O(card(\beta) \cdot card(D))$ space. However, fuzzy serts cannot simulate probabilities without exponentially greater requirements (theorem 7). The second part of theorem 5 shows that fuzzy sets can simulate possibilities without extra space requirements. This comes at no surprise for fuzzy sets have been defined in a possibilistic way.

It must be stressed, however, that those relationships hold for finite domains. As theorem 4 shows, when domains become infinite, all measures studied here are capable of covering the whole $[0,1]$ interval and as such are representationally equivalent to each other (i.e. any measure representable in one of them is representable in the other without any extension or extra requirements on the domain). Of course, this can be seen as an "overflowing" effect, in the sense that the above measures overflow (i.e. cover) their range when their domains become appropriately large. Furthermore, studying them in such an "overflowing" state would not give us insight on their intrinsic representational capabilities. Thus, it is the

[4]For example, consider the case of a domain Ω of n elements. Let $\Omega = \{\omega_1, \ldots, \omega_n\}$ and the probability distribution P be defined so that $P(\{\omega_i\}) = m\frac{1}{2^i}$, $i = 1, 2, \ldots, n$, where m is a constant chosen so that the probabilities above sum up to one (i.e. $m = \frac{2^n}{2^n - 1}$). Then, as it was shown in [Drakopoulos, 1994a], no possibility measure in a domain of less than 2^n elements can simulate the above probability distribution.

author's view that finite domains are more appropriate for comparing the corresponding system classes in term of the mappings they can represent.

The flip side of the representational advantage of probabilities, in finite domains, is their efficiency disadvantage. Possibilistic measures such as possibilities and fuzzy sets are easier to compute than probabilities since the former are extensional while the later are intensional measures. This theoretical advantage of possibilities is very noticeable in practice where one can easily compute the possibility value of a compound logical expression or relation among events or entities by simply applying the definition. On the other hand, using probabilities, such a task is either very difficult or impossible. For example, evaluating a Bayesian Network [Charniak, 1991, Pearl, 1988], is computationally an NP-hard problem ([Cooper, 1987]). Although there are simple and efficient algorithms for the restricted class of singly connected networks ([Neapolitan, 1990]), in general, exact algorithms are very complicated and often very inefficient. This led to approximate evaluations of the conditional probabilities in a Bayesian network ([Charniak, 1991, Pearl, 1990]), a solution which introduces an error which in turn reduces the information probabilities carry. Furthermore, the structure of the network implies independence assumptions which do not always capture all the possible situations and as such act as a new source of error and information loss. Therefore, it appears that probabilities often carry too much information imposing approximate or inefficient solutions.

Probabilistic approaches are not limited to Bayesian networks. Probabilistic logic [Nilsson, 1986] is another example. In general, Bayesian statistics have been used as a statistical theory of evidence, uncertainty, and inference ([Genesereth and Nilsson, 1987, Lukasiewicz, 1970]). Furthermore, a number of probabilistic techniques have been applied for pattern recognition and classification. These include (but are not limited to) Bayes classifier and learning, Parzen windows, hidden Markov models, k-nearest neighbor, and stochastic approximation methods. A good overview of them appers in [Duda and Hart, 1973].

Possibilistic measures, have a computational advantage over probabilities at the cost of reduced representational power when domains are finite. However, for infinite domains, possibilistic measures maintain their computational advantage at no apparent cost for their representational power is then equal to that of probabilities.

Applications of possibilities are mainly within the spectrum of fuzzy set applications. Fuzzy sets and logic have been used in representation and approximate reasoning [Yager *et al.*, 1987], pattern recognition [Dubois *et al.*, 1993, Pal and Majumder, 1986], operations research [Dubois *et al.*, 1993], and modeling uncertainty and control [Dubois *et al.*, 1993]. Applications include decision support systems, expert systems, natural language processing ([Yager *et al.*, 1987]), database management, linear programming, robotics, vision ([Dubois *et al.*, 1993]), clustering, classification, image analysis ([Dubois *et al.*, 1993, Pal and Majumder, 1986]), and speech recognition ([Pal and Majumder, 1986]).

In addition to the measures studied here, other measure systems have been used to model uncertainty [Shortliffe, 1976, Dempster, 1968, Shafer, 1976]. However, those measures have not been used in pattern recognition problems and are not studied here. For a discussion of their relationship to probabilistic and possibilistic measures see [Drakopoulos, 1994a, Klir, 1989, Klir and Parviz, 1992].

The results presented in this chapter establish a computational complexity versus representational power trade-off between probabilistic and possibilistic measures. Although further studies seem to be required in order to completely resolve this trade-off in each domain, some results in the next section stress some of the advantages of possibilistic measures (such as fuzzy sets) which are finally used in our hybrid pattern recognition system.

Chapter 3

Sigmoidal Theory

Eureka! (I've got it!)

Archimedes 287–212 B.C.

With a name like yours, you might be any shape, almost.

Lewis Carroll 1832–1898

In this chapter, we formally define and analyze a family of functions which we call *sigmoidal functions* for their shape resembles the Greek letter sigma (ς)[1]. The resulting theory, called *sigmoidal theory*, generates an infinite number of classes of sigmoidal functions (called *sigmoidal classes*). Currently, only few of those functions are known or used in applications. In addition, sigmoidal theory specifies an algorithm to construct sigmoidal classes and establishes that those classes form a partition of the sigmoidal family. Using sigmoidal functions belonging to a class, we can construct general functions of arbitrary complexity that are piecewise sigmoidal functions. Those are called *multi-sigmoidal functions.*

Sigmoidal functions and sigmoids in particular are in widespread use in neural networks [Hinton, 1989, Rumelhart *et al.*, 1986a]. In this chapter, we show that sigmoidal functions are very appropriate for representing fuzzy set membership, too. In fact, it was the study of fuzzy sets and pattern recognition problems that led to the formalization of sigmoidal

[1]We chose the name sigmoidal functions instead of sigmoids in order to distinguish our functions from those that have been called sigmoids in the literature. In general, all sigmoids are sigmoidal functions but the opposite is not true.

functions (see [Drakopoulos, 1991, Drakopoulos and Hayes-Roth, 1994] for our earliest approach).

In the next section, we formally introduce sigmoidal functions and state a number of theorems that constitute sigmoidal theory. In section 3.2, we specify an algorithm for generating affine sigmoidal classes and discuss why affine classes are preferable than general classes. That algorithm is a straightforward application of sigmoidal theory. In section 3.3, we describe multi-sigmoidal functions, which have been used in *multi-sigmoidal neural networks* [Drakopoulos, 1995]. Those functions are shown to be closely related to fuzzy set membership functions. Finally, in section 3.4, we discuss various applications of sigmoidal functions in fuzzy sets, pattern recognition, and neural networks.

3.1 Sigmoidal Theory

In the following we assume that $\mathcal{R}$ is the set of real numbers.

Definition 11 *A function $f : \mathcal{R} \rightarrow \mathcal{R}$ is a symmetric sigmoidal function (or simply a sigmoidal function) iff*

$$\begin{array}{ll} (a) & f \textit{ is monotonic} \\ (b) & \inf_{\mathcal{R}} f = L, \quad \sup_{\mathcal{R}} f = U \\ (c) & \exists m \in \mathcal{R} \;\; \forall x \in \mathcal{R} \quad f(m \Leftrightarrow x) + f(m + x) = L + U \end{array}$$

In that case, we write $f \in sigm(L, U, m)$.

Therefore, a real valued function over $\mathcal{R}$ is a symmetric sigmoidal function if and only if it is monotonic, bounded, and symmetric around a point ($= (m, \frac{L+U}{2})$). A typical example of a sigmoidal function appears in figure 3.1. We can furthermore define a function f to be a *unit sigmoidal function* iff $f \in sigm(0, 1, m)$. If, in addition, $m = 0$ and f is increasing then f is called *basic unit sigmoidal function.*

Now define

Definition 12 *A function $\alpha : D \rightarrow R$ is called an* index function *from D to R iff α is strictly monotonic in D and $\alpha(D) = R$.*

and

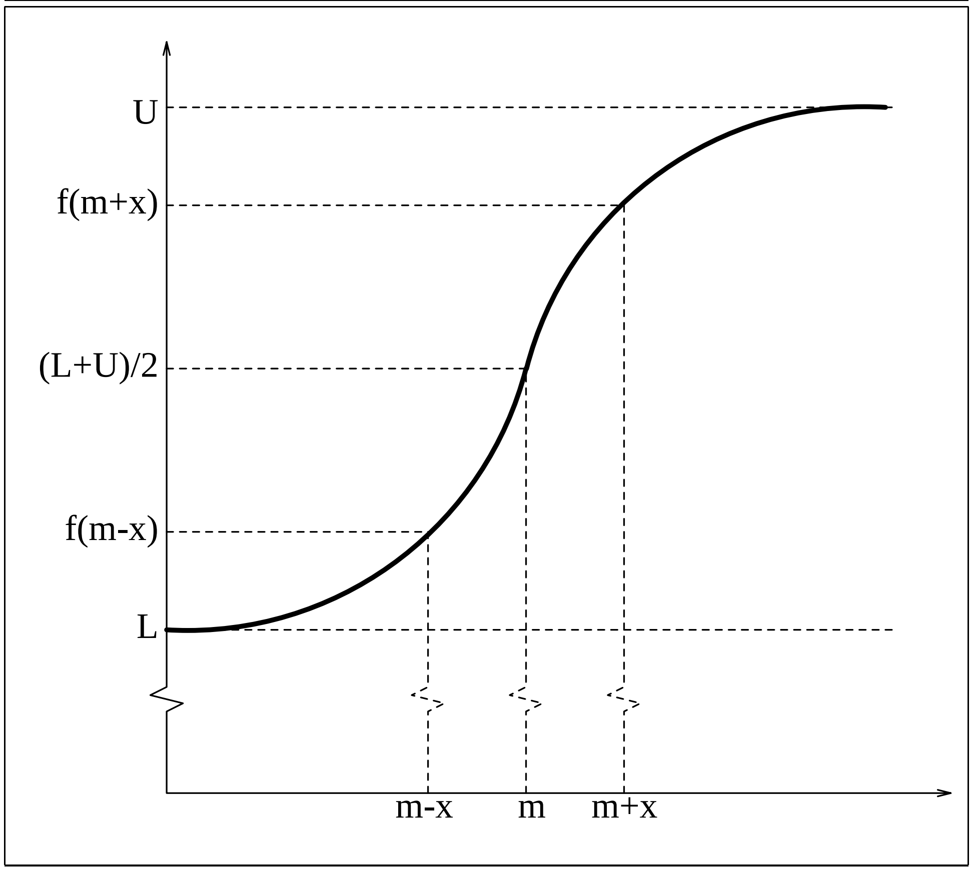


Figure 3.1: A typical sigmoidal function.

Definition 13 *The class C_f of a function $f : D \to R$ is the set of all functions that can be expressed as a composition of f with an index function from D to D i.e.*

$$C_f = \{ f \circ \alpha \ / \ \alpha \ \textit{is an index function from} \ D \ \textit{to} \ D \}$$

If, furthermore, the index function α is restricted to be an affine function the resulting class A_f is called the affine class of f.

Notice the the class of a symmetric sigmoidal functions contains functions that are S-shaped but not symmetric around a point. For example, if f is a symmetric sigmoidal function and α is an index defined as follows

$$\alpha(x) = \left\{ \begin{array}{ll} 4x & \text{If } x < 0 \\ x & \text{If } x \geq 0 \end{array} \right\}$$

then $f \circ \alpha$ does not have a center of symmetry (see figure 3.2). We call those functions *asymmetric sigmoidal functions.*

In the following, we write $f\uparrow$ to indicate that the function f is increasing and $f\downarrow$ to indicate that f is decreasing. We can now present the theorems of sigmoidal theory. Proofs of those theorems are given in [Drakopoulos, 1994b].

Theorem 8 (Sigmoidal Reduction Theorem). *Every sigmoidal function can be expressed as an affine transformation of a unit sigmoidal function or as the affine transformation of the composition of a basic unit sigmoidal function and a translation.*

The above theorem simply states that it is always possible to to transform a sigmoidal function (by an affine transformation) so that its range is within the interval $[0, 1]$. Furthermore, any sigmoidal function can always be translated so that its center of symmetry is the point $(0, 0.5)$ and then be transformed (by an affine transformation) so that it would be an increasing function in $[0, 1]$. Now, those transformations show that sigmoidal, unit sigmoidal, and basic unit sigmoidal functions are all equivalent under affine transformations and compositions. Furthermore, a similar theorem holds for asymmetric sigmoidal functions. If f is such a function then there exists an index function β such that $f \circ \beta$ is a symmetric sigmoidal function. Thus, the above theorem applies.

Sigmoidal theory is stated in terms of sigmoidal functions and not simply in terms of basic unit sigmoidal functions despite the transformational equivalence of those functions.

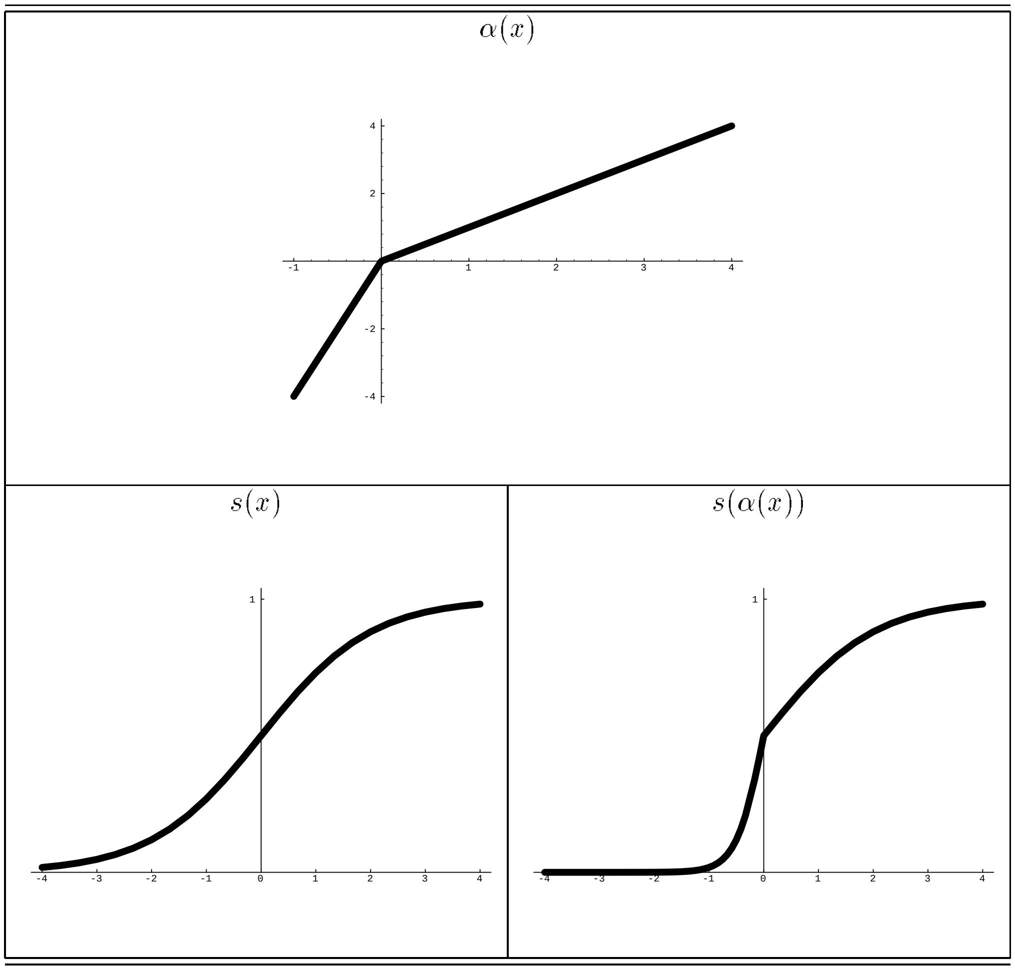


Figure 3.2: Example of generation of an asymmetric sigmoidal function.

Furthermore, sigmoidal theory focuses on functions that are almost everywhere differentiable (i.e. on functions whose derivative is not defined only over a subset of $\mathcal{R}$ that is of zero length). For example, consider a function that is differentiable everywhere except in a set of points that is at most countably infinite. Then that function is almost everywhere differentiable. Hence, all functions of practical interest are almost everywhere differentiable.

Theorem 9 (Sigmoidal Derivative Theorem). *If f is almost everywhere differentiable then*

$$f \in sigm(L, U, m) \Leftrightarrow \begin{cases} f' \textit{ does not change sign} \\ \left| \int_{-\infty}^{+\infty} f'(x)\, dx \right| = U \Leftrightarrow L \\ \forall x \in \mathcal{R} \quad f'(m \Leftrightarrow x) = f'(m + x) \end{cases}$$

We call the function f' a *sigmoidal generator.*

Corollary 1 *Every even probability density function is a basic unit sigmoidal generator.*

Theorem 10 (Sigmoidal Generation Theorem). *If f is almost everywhere differentiable then*

$$f \in sigm(L, U, m) \Leftrightarrow \exists g \quad \forall x \in \mathcal{R} \left\{ \begin{array}{lcl} g(x) & \geq & 0 \\ f(x) & = & \epsilon \int_{-\infty}^{x} g(t)\, dt + A \end{array} \right\}$$

where $(\epsilon, A) = \left\{ \begin{array}{ll} (+1,\ L) & \textit{if } f\uparrow \\ (\Leftrightarrow 1,\ U) & \textit{if } f\downarrow \end{array} \right\}.$

Lemma 1

$$\begin{array}{l} \forall f_1 \in sigm(L, U, m_1) \\ \quad \forall f_2 \in sigm(L, U, m_2) \\ \qquad \begin{array}{rcl} C_{f_1} \cap C_{f_2} \neq \emptyset & \Leftrightarrow & \exists \alpha_1, \alpha_2, \ \ \forall x \quad f_1(\alpha_1(x)) = f_2(\alpha_2(x)) \\ & \Leftrightarrow & \exists \alpha_1, \alpha_2, \ \ \forall x \quad \epsilon_1 \alpha_1'(x) g_1(\alpha_1(x)) = \epsilon_2 \alpha_2'(x) g_2(\alpha_2(x)) \\ & \Leftrightarrow & C_{f_1} = C_{f_2} \end{array} \end{array}$$

where α_1, α_2 are index functions from $\mathcal{R}$ to $\mathcal{R}$, and $\epsilon_1, g_1, \epsilon_2, g_2$ result by applying the sigmoidal generation theorem to f_1, f_2, respectively.

Theorem 11 (Sigmoidal Partitioning Theorem). *If f_1, f_2 are almost everywhere differentiable then*

$$\begin{aligned} &\forall f_1 \in sigm(L, U, m_1) \\ &\quad \forall f_2 \in sigm(L, U, m_2) \\ &\qquad C_{f_1} \cap C_{f_2} \neq \emptyset \quad \Leftrightarrow \quad C_{f_1} = C_{f_2} \end{aligned}$$

Corollary 2 *All strictly monotonic sigmoidal functions (that are almost everywhere differentiable) belong to a single sigmoidal class.*

The above corollary states that if we do not restrict ourselves in choosing the index functions then sigmoidal functions tend to merge into large classes. For the same reason, it is difficult to test whether two sigmoidal functions belong to the same class or not. An appropriate constraint to impose on the choice of the index functions in order to overcome the drawbacks just mentioned, is to require index functions to be affine. This results in affine sigmoidal classes and is discussed in more detail in the next section as well as in [Drakopoulos and Hayes-Roth, 1994].

We present now the last theorem of the sigmoidal theory:

Theorem 12 (Sigmoidal Bubble Theorem). *If s is a basic unit sigmoidal function then*

$$\forall s_1, s_2 \in C_s \quad \exists \alpha_1, \alpha_2 \quad \forall x_1, x_2 \in \mathcal{R}$$
$$\left[\begin{array}{lcl} \sup(s_1(x_1), s_2(x_2)) & = & s(\sup(\alpha_1(x_1), \alpha_2(x_2))) \\ \inf(s_1(x_1), s_2(x_2)) & = & s(\inf(\alpha_1(x_1), \alpha_2(x_2))) \\ 1 \Leftrightarrow s_1(x_1) & = & s(\Leftrightarrow \alpha_1(x_1)) \end{array} \right]$$

where α_1, α_2 indicate index functions from $\mathcal{R}$ to $\mathcal{R}$.

The importance of the above theorem is explained in section 3.4,

3.2 Sigmoidal Class Generation Algorithm

The sigmoidal generation and partitioning theorems of section 3.1 provide a way to generate sigmoidal classes given a set of index functions $\mathcal{A}$. The algorithm is as follows:

1. Choose an even non-negative function g such that

$$\int_{-\infty}^{+\infty} g(t)\, dt = U \Leftrightarrow L$$

2. Define $s(x) = \epsilon \int_{-\infty}^{x} g(t)\, dt + A$, where ϵ, A are as defined in the sigmoidal generation theorem.

3. Define a new sigmoidal class $C_s = \{s \circ \alpha \ / \ \alpha \in \mathcal{A}\}$

Now, if we choose $\mathcal{A}$ to be the set of affine indices i.e. indices of the form $ax \Leftrightarrow c$, where $a \neq 0$, then the above algorithm produces the following affine classes:

$$A_s = \{f \ / \ \exists a, c \quad a \neq 0 \quad \text{and} \quad \forall x \quad f(x) = s(ax \Leftrightarrow c)\}$$

The choice of affine indices is motivated not only by their simplicity and low computational cost but also by their clarity in defining sigmoidal classes. To see the later, observe that, by lemma 1, two sigmoidal classes C_{f_1}, C_{f_2} would intersect each other and as such would be identical iff

$$\exists a_1, c_1, a_2, c_2 \in \mathcal{R} \quad \forall x \in \mathcal{R} \quad \epsilon_1 a_1 g_1(a_1 x \Leftrightarrow c_1) = \epsilon_2 a_2 g_2(a_2 x \Leftrightarrow c_2)$$

where $a_1 \neq 0$ and $a_2 \neq 0$. Therefore, two affine sigmoidal classes are disjoint if and only if their generators are not affinely related. Since the later condition is easy to test, affine sigmoidal classes are easily discernible. Finally, affine indices result in sigmoidal classes that do not contain asymmetric sigmoidal functions[2].

Figure 3.3 depicts a number of basic unit sigmoidal functions and their generators. As it can be easily verified, the generators are not affinely related and as such the underlying affine sigmoidal classes are all disjoint.

Finally, each sigmoidal function that belongs to a particular affine class can now be represented by a pair of numbers (a, c). This reduces the storage requirements of sigmoidal functions to a level comparable to that of linear segments and makes them very attractive as representation functions.

[2]This is due to the fact that if $s \in sigm(L, U, m)$ and $f(x) = s(ax \Leftrightarrow c)$ then $f \in sigm(L, U, \frac{m+c}{a})$

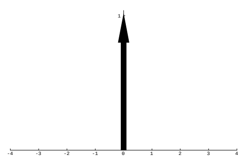
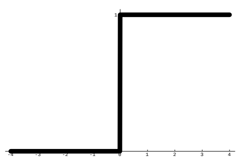
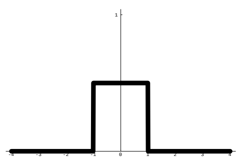
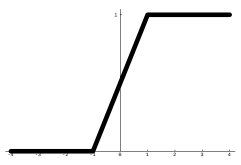
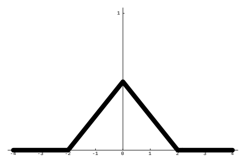
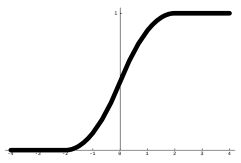
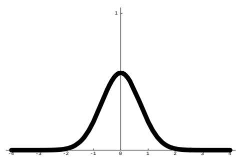
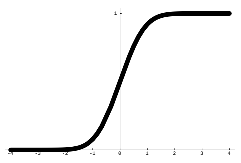
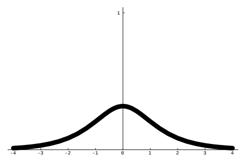
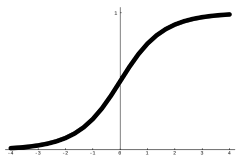
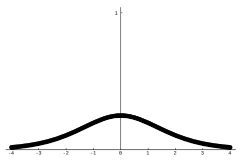
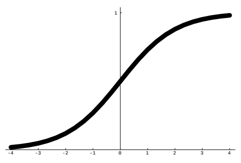
Generating Function	Sigmoidal Function	Generating Function	Sigmoidal Function
$\delta(x)$ 	$step(x)$ 	$box(x)$ 	$ramp(x)$ 
$\triangle(x)$ 	$quadratic(x)$ 	$\frac{1}{\sqrt{\pi}}e^{-x^2}$ 	$\frac{1}{\sqrt{\pi}}\int_{-\infty}^{x} e^{-t^2}\,dt$ 
$\frac{2}{\pi(e^x+e^{-x})}$ 	$\frac{2}{\pi}\arctan(e^x)$ 	$\frac{1}{e^x+2+e^{-x}}$ 	$\frac{1}{1+e^{-x}}$ 

Figure 3.3: Basic unit sigmoidal functions and their generators.

3.3 Multi-Sigmoidal Functions

Multi-sigmoidal functions are composed of many sigmoidal functions so that their monotonicity can change at many points. Assuming that S is a basic unit sigmoidal function and $S_{a,c}$ denotes a member of the affine class A_S of S (e.g. $S_{a,c}(x) = S(ax \Leftrightarrow c)$), a formal definition of multi-sigmoidal functions is as follows:

Definition 14 *A function f is called a multi-sigmoidal function (written as $ms(f)$) iff*

$$
\begin{array}{lll}
\exists k, a_1, c_1, \ldots a_k, c_k & & \\
\quad (a) \quad \forall i \in \{2, \ldots, k\} & a_{i-1}a_i < 0 \\
\quad (b) \quad \forall x \in \mathcal{R} & f(x) = \sum_{i=1}^{k} S_{a_i,c_i}(x)[b_{i-1} < x \leq b_i]
\end{array}
$$

where

$$
\begin{aligned}
[\phi] &= \left\{ \begin{array}{ll} 1 & \textit{if } \phi \textit{ is true} \\ 0 & \textit{if } \phi \textit{ is false} \end{array} \right\} \\
b_i &= \frac{c_{i+1} \Leftrightarrow c_i}{a_{i+1} \Leftrightarrow a_i}, \quad i = 1, 2, \ldots, k \Leftrightarrow 1 \\
b_0 &= \Leftrightarrow \infty, \qquad b_k \;=\; +\infty
\end{aligned}
$$

In that case, we write

$$f = (S_{a_1,c_1}, b_1, S_{a_2,c_2}, b_2, \ldots, b_{k-1}, S_{a_k,c_k})$$

The constants a_i, c_i are called *affine coefficients* while the points b_i are called *barrier locations*, for $i = 1, \ldots, k$. Intuitively, affine coefficients determine the positioning and steepness of the sigmoidal functions while barrier locations indicate where the sigmoidal functions intersect and thus where the monotonicity of the overall multi-sigmoidal function changes.

Condition (a) in definition 14 is required so that successive sigmoidal functions have different monotonicity while condition (b) creates a piece-wise sigmoidal shape. Barrier locations are defined in terms of affine coefficients so that the function f is continuous at $b_1, b_2, \ldots, b_{k-1}$. Furthermore, if S is strictly monotonic then the above definition of barrier locations is not only a sufficient but also a necessary condition for continuity of the function f at those locations. Figure 3.4 shows a typical multi-sigmoidal function.

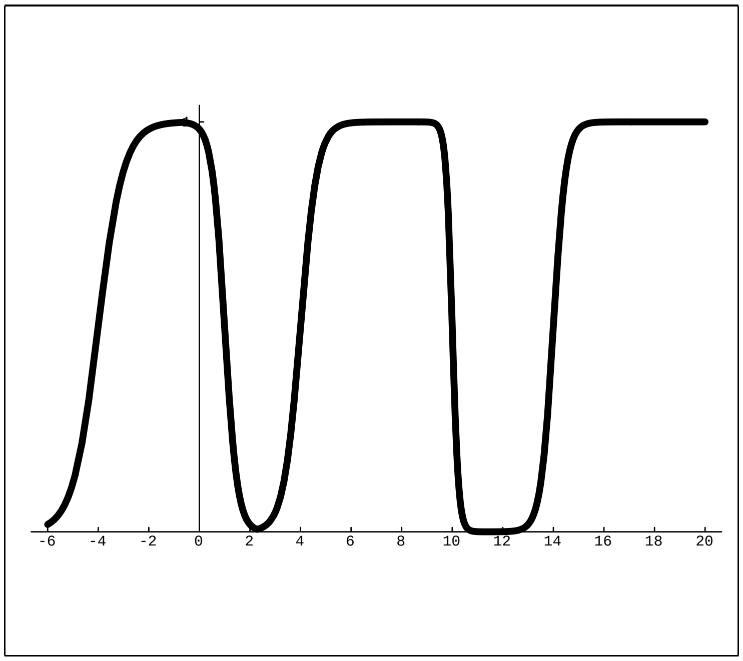


Figure 3.4: A typical multi-sigmoidal function.

Multi-sigmoidal functions have been used successfully in multi-sigmoidal neural networks as non-monotonic unit activation functions [Drakopoulos, 1995]. Their flexible shape and their ability to undergo local shape transformations (within two barrier locations) without affecting the shape of distant sigmoidal segments of the curve make multi-sigmoidal functions very powerful functional representation devices. Multi-sigmoidal neural networks are described in more detail in the next chapter.

There is a strong relationship between multi-sigmoidal functions and fuzzy set membership functions. If f is a multi-sigmoidal function (as above) then we can define a new function g that approximates f as follows:

$$g(x) = \left\{ \begin{array}{ll} g_D(x) & \text{if } a_1 > 0 \text{ and } k \text{ is even} \\ \sup\{g_D(x), S_{a_k,c_k}(x)\} & \text{if } a_1 > 0 \text{ and } k \text{ is odd} \\ g_C(x) & \text{if } a_1 < 0 \text{ and } k \text{ is even} \\ \inf\{g_C(x), S_{a_k,c_k}(x)\} & \text{if } a_1 < 0 \text{ and } k \text{ is odd} \end{array} \right\}$$

where

$$\begin{aligned} g_C(x) &= \sup_{i=1}^{\lfloor k/2 \rfloor} \{\inf(S_{a_{2i-1},c_{2i-1}}(x), S_{a_{2i},c_{2i}}(x))\} \\ g_D(x) &= \inf_{i=1}^{\lfloor k/2 \rfloor} \{\sup(S_{a_{2i-1},c_{2i-1}}(x), S_{a_{2i},c_{2i}}(x))\} \end{aligned}$$

Furthermore, if S_{a_i,c_i} is the membership function of a fuzzy set A_i then g is the membership function of the fuzzy set G defined by

$$G = \left\{ \begin{array}{ll} G_D & \text{if } a_1 > 0 \text{ and } k \text{ is even} \\ G_D \cup A_k & \text{if } a_1 > 0 \text{ and } k \text{ is odd} \\ G_C & \text{if } a_1 < 0 \text{ and } k \text{ is even} \\ G_C \cap A_k & \text{if } a_1 < 0 \text{ and } k \text{ is odd} \end{array} \right\}$$

where G_D and G_C are the fuzzy sets with membership functions g_D and g_C, respectively. Thus

$$\begin{aligned} G_D &= \cup_{i=1}^{\lfloor k/2 \rfloor} (A_{2i-1} \cap A_{2i}) \\ G_C &= \cap_{i=1}^{\lfloor k/2 \rfloor} (A_{2i-1} \cup A_{2i}) \end{aligned}$$

As a result, multi-sigmoidal functions can be seen as arbitrary fuzzy set membership functions defined in terms of sigmoidal functions. Intuitively, G_D is in disjunctive normal form while G_C is in conjunctive normal form in terms of their constituent fuzzy sets, $A_i, \quad i = 1, 2, \ldots, k$. Depending on whether its leftmost sigmoidal component is increasing or decreasing (i.e. $a_1 > 0$ or $a_1 < 0$), a multi-sigmoidal function can be seen as a sequence of "hills" or a sequence of "valleys", respectively. Then the i-th "hill" or "valley" can be approximated by the membership function of the fuzzy set $A_{2i-1} \cap A_{2i}$ or $A_{2i-1} \cup A_{2i}$, respectively. Finally, the overall function emerges as the disjunction of the "hills" or the conjunction of the "valleys" resulting in disjunctive or conjunctive normal forms.[3]

The only difference between the above definition of multi-sigmoidal functions and their original one is that the new one has the undesirable property that the value of the function g at a point x may depend not only on the sigmoidal functions surrounding x but also on

[3]The case where the number of sigmoidal components k is odd, is treated separately for it results in a "hill" or "valley" than has only one "side".

sigmoidal functions that are distant to x. This situation is depicted in figure 3.5 where one of the constituent sigmoidal functions of the multi-sigmoidal function f affects the value of f only within a closed interval between 3 and 14. However, in the fuzzy set theoretic approximation g of f the area of influence of that same sigmoidal function is extended all the way to $\Leftrightarrow\infty$ suppressing most of the first (from left to right) sigmoidal function. It is now obvious that local shape modifications are not guaranteed to be local. This drawback led to the adoption of the original definition that uses barrier locations to achieve local modifiability.

3.4 Application of Sigmoidal Functions

Fuzzy sets and pattern recognition are areas where sigmoidal functions find immediate application.

Regarding fuzzy sets, sigmoidal functions are very useful in modeling fuzzy membership due to the sigmoidal bubble theorem. That theorem showed that one can always interchange sigmoidal functions with sup, inf, and complementation operations. As a result, if all the constituent subexpressions of a fuzzy set theoretic expression are evaluated through sigmoidal functions of a single class, only a single sigmoidal computation is required to evaluate the overall expression no matter how complicated the expression is. For example, assume that s is a basic unit sigmoidal function and C_s be its sigmoidal class. Furthermore, assume that f_i is the membership function of the fuzzy set A_i and that $f_i \in C_s$ i.e.

$$f_i = s \circ \alpha_i$$

for some index function α_i. Then if D is a new fuzzy set defined as

$$D = A_1 \cup (A_2 \cap \bar{A}_3) \cup (A_3 \cap \bar{A}_4)$$

where $\bar{A}$ indicates the complement of the fuzzy set A, then, due to sigmoidal bubble theorem, it would be

$$\begin{aligned} f_D &= \sup\{f_{A_1},\ \inf(f_{A_2}, 1\Leftrightarrow f_{A_3}),\ \inf(f_{A_3}, 1\Leftrightarrow f_{A_4})\} \\ &= s(\sup\{\alpha_1,\ \inf(\alpha_2, \Leftrightarrow\alpha_3),\ \inf(\alpha_3, \Leftrightarrow\alpha_4)\}) \end{aligned}$$

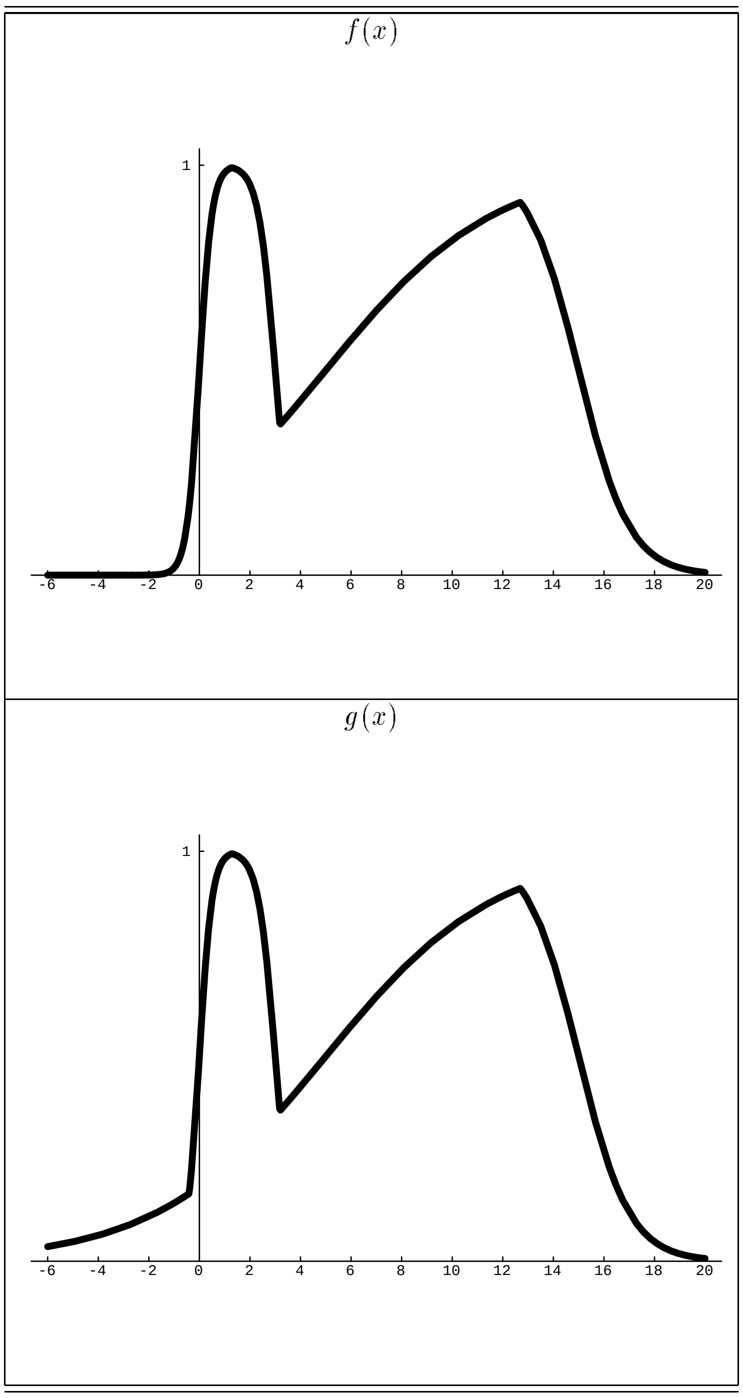


Figure 3.5: A multi-sigmoidal function and its fuzzy set approximation.

i.e. the membership function of the new set can be computed by evaluating s only once. This is true irrespectively of the length or complexity of the expression that defines the new set. Furthermore, if the new set D is used in another expression that defines another fuzzy set then the same transformations would apply and only one sigmoidal computation would be required to compute the membership function of the new set. Finally, since we always need to compute only a single sigmoidal function s, we can tabulate its values to further speed up the computations.

Sigmoidal functions, like bubbles in water, move toward the "surface" of a fuzzy set theoretic expression and as such need only be computed once, per expression. Furthermore, if all sigmoidal functions belong to the same class then they would form a big "bubble" that corresponds to a single basic sigmoidal function like s above. This analogy of computations with sigmoidal functions and "bubbles" is responsible for the name of theorem 12.

The approach of representing fuzzy set membership with sigmoidal functions has been taken in the tFPR system which is described in chapter 5 as well as in [Drakopoulos and Hayes-Roth, 1994, Drakopoulos, 1991].

A number of functions have been used so far to model fuzzy set membership ([Yager *et al.*, 1987], [Kaufmann, 1975], ch.III, section 29, [Gupta *et al.*, 1975]). Often low order polynomials (mainly, linear segments, segments of parabolas, and splines) are used for they are very simple, flexible, and efficient to use. We have showed that sigmoidal functions have similar properties and, in particular, result in very efficient computations. Furthermore, some preliminary experiments done with the tFPR system showed that sigmoidal functions, in general, outperform low order polynomials both in terms of accuracy and time or space efficiency. It is likely that the use of sigmoidal functions as fuzzy set membership functions would grow after their formalization, here.

Another application area of sigmoidal functions is neural networks. Here, sigmoids are in widespread use as neuron activation functions [Hinton, 1989, Rumelhart *et al.*, 1986a]. The two sigmoids most commonly used are the logistic function and $tanh(x)$. A good discussion and some experimental results regarding the trade off between sigmoid and radial basis functions as neuron activation functions appear in [Weigend *et al.*, 1990, Moody and Darken, 1989, Lapedes and Farber, 1987] . Further comparisons of sigmoids with other unit activation functions appear in [Moody and Yarvin, 1992, DasGupta and Schnitger, 1993, Drakopoulos, 1995],

We believe that detailed studies are required in order to determine which sigmoidal functions are the best for a set of different applications and to determine their most effective organization in each unit. The internal functional richness of the sigmoidal family and its close ties with the paradigms of neural networks and fuzzy sets make the sigmoidal family not only an important source of representational elements but also a potential link between the two paradigms that could allow their mutual interpretation.

Chapter 4

Multi-Sigmoidal Neural Networks

Since two distinct finite subsets of pattern vectors can always be nonredundantly partitioned by a set of parallel hyperplanes, ...

Learning Machines
Nils J. Nilsson 1933–

Multi-sigmoidal units, which are a generalization of ordinary sigmoidal units used in neural networks, are discussed in this chapter. *Multi-sigmoidal neural networks* (i.e. neural networks with multi-sigmoidal units) inherit all function representation and approximation capabilities of ordinary sigmoidal neural networks. In addition, as proved here, if X is a finite and discrete subset of real numbers then any function $f : X^n \rightarrow \{0, \ldots, C \Leftrightarrow 1\}^\lambda$ can be represented by a two-layer network (i.e. no hidden units) that has $\lambda \lceil \log_2 C \rceil$ multi-sigmoidal output units. The above result indicates that there is a trade-off in capturing interactions among inputs and representing or approximating functions either via hidden units or via non-monotonic unit activation functions.

Multi-layer multi-sigmoidal neural networks combine the above two mechanisms i.e. hidden units and non-monotonic activation functions. Those networks consist of a new neural network architecture based on dynamically created non-monotonic activation functions that are modeled by a set of sigmoidal functions. A modification of the back-propagation algorithm is presented in this chapter that is capable of learning both the weights and the unit activation functions themselves. A number of classification problems are used to evaluate that algorithm in terms of convergence rates, solution quality, and generalization accuracy.

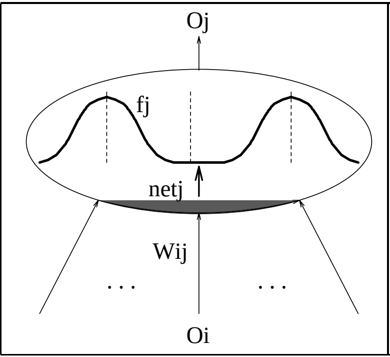
4.1 Introduction

Perceptrons have been severely criticized for their inability to represent simple boolean functions (such as exclusive-OR) [Minsky and Papert, 1988]. As it has been shown in the literature [Rumelhart *et al.*, 1986a], the above limitation is mainly due to the fact that perceptrons are two-layer networks (i.e. they have no hidden layers). Three-layer neural networks with arbitrary activation functions (especially in the upper layer) can represent any continuous function [Kolmogorov, 1957, Sprecher, 1965, Hecht-Nielsen, 1987]. On the other hand, given a particular activation function ψ, three-layer neural networks can uniformly approximate any continuous function, if and only if ψ is non-polynomial [Leshno *et al.*, 1993, Hornik, 1993]. Furthermore, if ψ is a squashing function then three-layer neural networks can uniformly approximate any Borel measurable function [Hornik *et al.*, 1989]. Justified by the later results, simple monotonic activation function (such as sigmoidal functions) have been used in many neural network applications [Rumelhart *et al.*, 1986a, Hinton, 1989, Sejnowski and Rosenberg, 1987].

Here, it is shown that some of the computational limitations of two-layer networks with monotonic activation functions are due to the monotonicity of their activation functions. In our study, units, which are called *multi-sigmoidal units*, use multi-sigmoidal functions as unit activation functions.

In the next section, it is proved that if X is a discrete and finite subset of real numbers then any function $f : X^n \rightarrow \{0, 1\}$ can be represented by a single multi-sigmoidal unit. As a consequence, any function $f : X^n \rightarrow \{0, \ldots, C \Leftrightarrow 1\}^\lambda$ can be represented by a two-layer network (i.e. no hidden units) that has $\lambda \lceil \log_2 C \rceil$ multi-sigmoidal output units. Unfortunately, a similar statement cannot be made about two-layer neural networks with monotonic activation functions. Those networks can only represent or approximate linearly separable functions [Minsky and Papert, 1988, Nilsson, 1965]. Thus, the above results indicate that there is a trade-off in capturing interactions among inputs and representing or approximating functions using either hidden units or non-monotonic unit activation functions.

Non-monotonic activation functions are not uncommon in neural network literature [Lippmann, 1989, Poggio and Girosi, 1989, Moody and Yarvin, 1992, Dawson and Schopflocher, 1992]. Empirical comparisons of sigmoids and radial basis functions appear



$$\begin{aligned} o_j &= f_j(net_j) \\ f_j &= (S_{a_{j1},c_{j1}}, b_{j1}, \ldots, b_{j,k-1}, S_{a_{jk},c_{jk}}), \\ &\quad \text{for some } k \in \mathcal{N} \\ net_j &= \textstyle\sum_i o_i w_{ij} \end{aligned}$$

Figure 4.1: A typical MS unit. net_j is the net input to node j, o_j is its output, and f_j is its activation function.

in [Weigend *et al.*, 1990, Moody and Darken, 1989, Lapedes and Farber, 1987] while empirical comparisons of sigmoids, polynomials, rational functions, and flexible Fourier series appear in [Moody and Yarvin, 1992]. All those studies provide some empirical evidence that non-monotonic unit activation functions could aid representation and approximation of functions by neural networks.

4.2 Multi-Sigmoidal Units and Neural Networks

Multi-sigmoidal units (MS units) are units with multi-sigmoidal unit activation functions while multi-sigmoidal neural networks (MSNNs) are neural networks whose units are MS units. A typical MS unit and its functionality are shown in figure 4.1. In this chapter we will focus in cases where S is a unit sigmoidal function.

Obviously, if we choose the sigmoidal function S to be the step function (see figure 3.3 for a definition of the step function) then a corresponding MS unit with k instances of S would implement k parallel hyperplanes. Figure 4.2 shows a graph of the output of an MS unit with two inputs and $k = 4$ sigmoidal functions. In the first case, S is chosen to be the step function which, in turn, results in regions in input space having boundaries that are defined by a number of parallel hyperplanes. However, in the second case, S is chosen to be the logistic function. In that case, regions have "smooth" boundaries which can be considered as parallel "hyper-bands". The steepness of S as well as the magnitudes of the weights determine the width of those hyper-bands. Alternatively, the step function can be

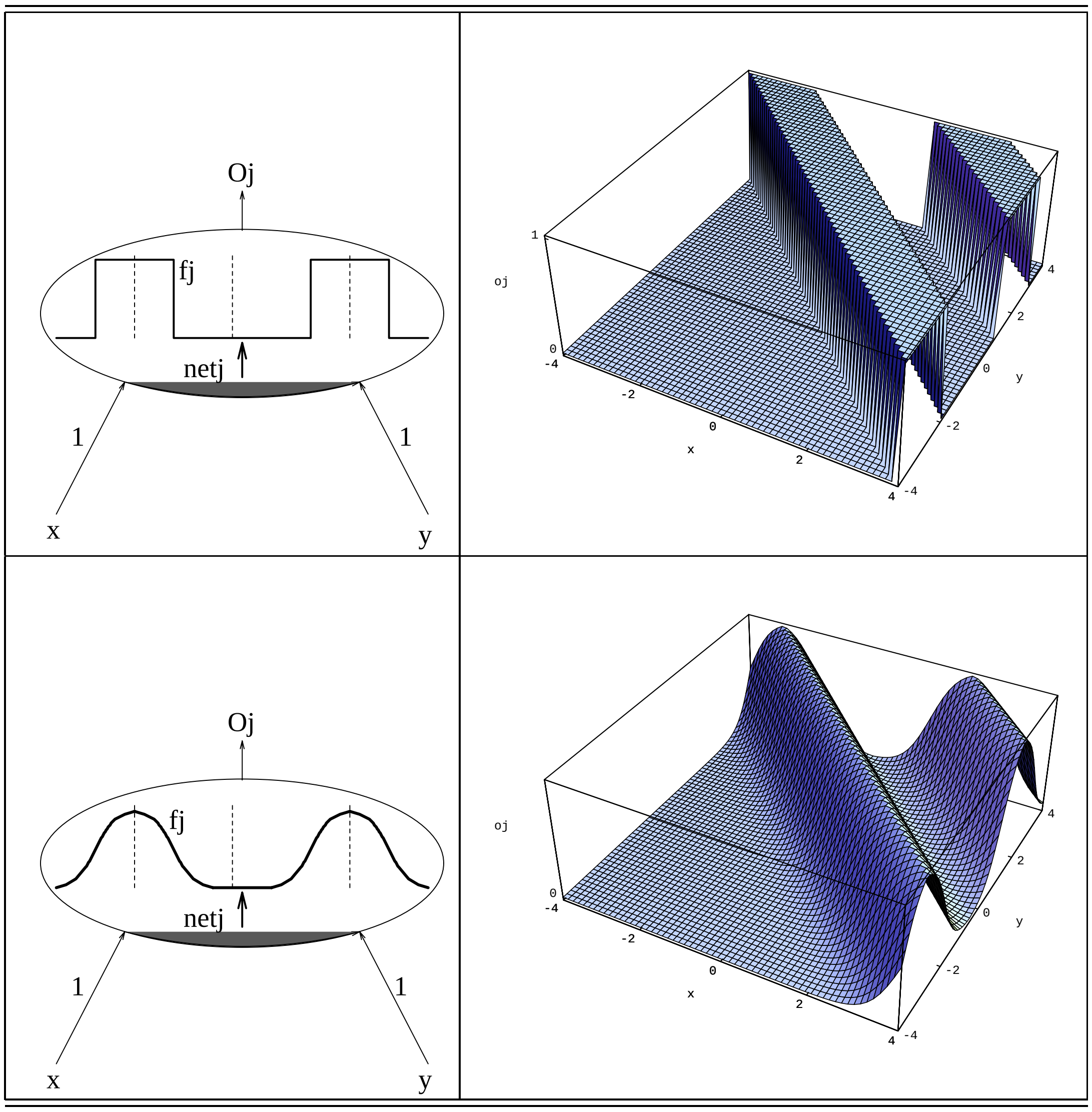


Figure 4.2: Multi-sigmoidal units for the step and logistic function with $(a_1, a_2, a_3, a_4) = (3, ⇔3, 3, ⇔3)$ and $(c_1, c_2, c_3, c_4) = (0, ⇔6, 15, ⇔21)$. The graphs show the output o_j over x, y.

considered as a special case of a sigmoidal function with infinite steepness that results in hyper-bands or zero width (i.e. hyperplanes).

The definition of MS units motivates us in extending the definition of linear separability [Nilsson, 1965] to *p-linear separability* (p stands for parallel):

Definition 15 *Two subsets $\mathcal{Y}_0, \mathcal{Y}_1$ of $\mathcal{R}^d$ ($d \in \mathcal{N}$) are said to be p-linearly separable iff there exists a threshold $\theta \in [0, 0.5)$, a unit sigmoidal function S, and an MS unit that uses S to implement a (discriminant) function $g : \mathcal{R}^d \to [0, 1]$ such that*

$$\begin{aligned} \forall y \in \mathcal{Y}_0 \qquad & g(y) \leq \theta \\ \forall y \in \mathcal{Y}_1 \qquad & g(y) \geq 1 \Leftrightarrow \theta \end{aligned}$$

Intuitively, if we can separate $\mathcal{Y}_0, \mathcal{Y}_1$ by a set of parallel hyperplanes (or hyper-bands) then $\mathcal{Y}_0, \mathcal{Y}_1$ would be p-linearly separable.

The importance of p-linear separability lies in the fact that it can be implemented by a single MS unit as well as the fact that $\mathcal{Y}_0, \mathcal{Y}_1$ can always be separated by at most $N \Leftrightarrow 1$ parallel hyperplanes provided that they are disjoint and the number N of points in $\mathcal{Y}_0 \cup \mathcal{Y}_1$ is finite. Formally, we can state and prove the following theorem:

Theorem 13 *Let $X \subset \mathcal{R}$ be a finite and discrete set and S be a unit sigmoidal function with range R_S (i.e. $(0, 1) \subseteq R_S \subseteq [0, 1]$). Then*

$$\begin{aligned} & \forall n \geq 1 \quad \forall f : X^n \to \{0, 1\} \quad \exists W \in \mathcal{R}^n \quad \forall \epsilon > 0 \\ & \quad \exists k, a_1, \ldots, a_k, c_1, \ldots, c_k, b_1, \ldots, b_{k-1} \quad \forall u \in X^n \quad |f(u) \Leftrightarrow g(W \cdot u)| \leq \epsilon \end{aligned}$$

where $g = (S_{a_1,c_1}, b_1, S_{a_2,c_2}, b_2, \ldots, b_{k-1}, S_{a_k,c_k})$. Furthermore, if $R_S = [0, 1]$ then ϵ could be chosen to be zero (i.e. $f = g$) in the above inequality.

Proof
Let $m = \min\{|x \Leftrightarrow y| \ / \ x, y \in X, x \neq y\}$, $M = \max\{|x \Leftrightarrow y| \ / \ x, y \in X\}$, $B = \frac{M}{m} + 1$, and $W^t = [B^0, B^1, \ldots, B^{n-1}]$. We shall now prove that the function $h(u) = W \cdot u$, $u \in X^n$ is $1 \Leftrightarrow 1$ in X^n. To this purpose, let $u = [u_0, \ldots, u_{n-1}], u' = [u'_0, \ldots, u'_{n-1}]$, and $u, u' \in X^n$. We shall first prove that

$$\forall k \in \{0, 1, \ldots, n \Leftrightarrow 1\} \qquad u_k < u'_k \quad \to \quad \sum_{i=0}^{k} B^i (u_i \Leftrightarrow u'_i) < 0 \tag{4.1}$$

Indeed, $\sum_{i=0}^{k} B^i(u_i \Leftrightarrow u_i') \leq \Leftrightarrow B^k m + \sum_{i=0}^{k-1} B^i M = \Leftrightarrow B^k m + (B^k \Leftrightarrow 1)m < 0$.
Now, if $u \neq u'$ then $j \overset{\triangle}{=} \max\{i \ / \ u_i \neq u_i', \ 0 \leq i \leq n \Leftrightarrow 1\}$ is well defined and

$$h(u) \Leftrightarrow h(u') = \sum_{i=0}^{j} B^i(u_i \Leftrightarrow u_i') \neq 0$$

For the last inequality simply observe that if $u_j < u_j'$ then by (4.1) the above sum is negative. Similarly if $u_j > u_j'$ then the sum has to be positive. This proves that h is $1 \Leftrightarrow 1$ in X^n. As a result, we can write X^n as $\{u_1, u_2, \ldots, u_N\}$ so that $h(u_i) < h(u_{i+1})$ $(i = 1, 2, \ldots, N \Leftrightarrow 1)$, where N is the finite cardinal of X^n.

Now, we can construct g. To this purpose, let $u_{i_1}, u_{i_2}, \ldots, u_{i_k}$ denote the elements of X^n where it is $f(u_{i_j}) \neq f(u_{i_j+1})$ $(j = 1, 2, \ldots, k \Leftrightarrow 1)$. Then define

$$g = (S_{a_1,c_1}, b_1, S_{a_2,c_2}, b_2, \ldots, b_{k-1}, S_{a_k,c_k})$$

where

$$|f(u_{i_j}) \Leftrightarrow S_{a_j,c_j}(h(u_{i_j}))| = |f(u_{i_j+1}) \Leftrightarrow S_{a_j,c_j}(h(u_{i_j+1}))| = \epsilon$$

Obviously, since $\{f(u_{i_j}), f(u_{i_j+1})\} = \{0, 1\}$, ϵ could be chosen to be zero in the above equations, if $R_S = [0, 1]$. In either case, those equations can define solution sets for a_j, c_j $(j = 1, 2, \ldots, k)$. Any element in those solution sets will be an acceptable assignment of values to the affine coefficients a_j, c_j $(j = 1, 2, \ldots, k)$. The barrier locations are now defined as $b_j = \frac{c_{j+1} - c_j}{a_{j+1} - a_j}$, $j = 1, 2, \ldots, k \Leftrightarrow 1$. This completely defines the function $g(W \cdot u)$, $u \in X^n$ as shown in figure 4.3 (the dashed lines represents the polygonal extension of the discrete function $f(u)$ and the solid curve represents the function $g(h(u)) = g(W \cdot u)$).

Now, $\forall u \in \{u_{i_1}, u_{i_2}, \ldots, u_{i_k}\}$ it would be $u = u_{i_j}$, for some $j \in 1, \ldots, k$ and so it would be $|f(u) \Leftrightarrow g(W \cdot u)| = |f(u_{i_j}) \Leftrightarrow S_{a_j,c_j}(h(u_{i_j}))| = \epsilon$. On the other hand, $\forall u \in X^n \Leftrightarrow \{u_{i_1}, u_{i_2}, \ldots, u_{i_k}\}$ it would be true that $\exists j \in \{0, 1, \ldots k + 1\}$ $\ h(u_{i_j}) < h(u) < h(u_{i_{j+1}})$, where we define $u_{i_0} = u_1$ and $u_{i_{k+1}} = u_N$ to cover boundary cases.

Now, $f(u)$ is constant (equal to $f(u_{i_j+1})$), for $u \in \{u_{i_j+1}, \ldots, u_{i_{j+1}}\}$. Furthermore, there are two possible cases, as shown in figure 4.3, depending on whether $f(u_{i_j+1})$ would be equal to 1 or 0. In either case, however, the functions $S_{a_j,c_j}, S_{a_{j+1},c_{j+1}}$ approximate $f(u)$ in the interval $[h(u_{i_j+1}), h(u_{i_{j+1}})]$ closer than they do at points $h(u_{i_j+1}), h(u_{i_{j+1}})$. Therefore,

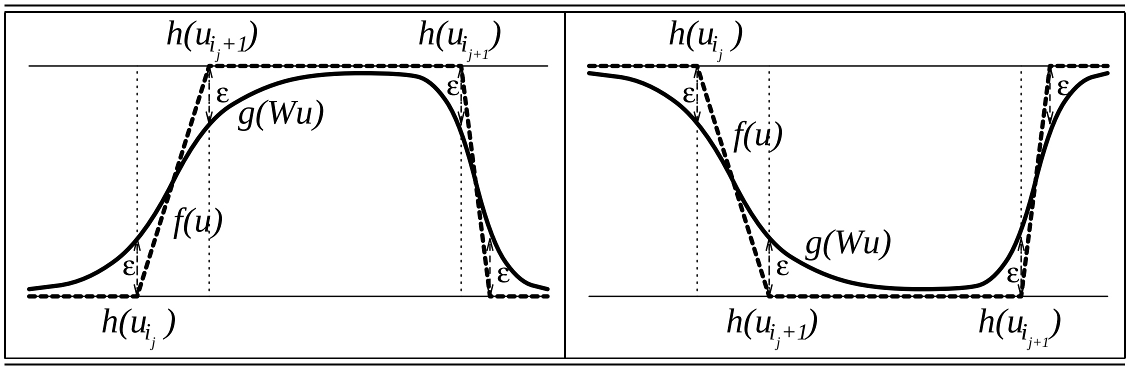


Figure 4.3: Approximation of $f(u)$ by $g(W \cdot u)$.

it is

$$|f(u) \Leftrightarrow g(W \cdot u)| \leq |f(u_{i_j}) \Leftrightarrow S_{a_j,c_j}(h(u_{i_j}))| = |f(u_{i_{j+1}}) \Leftrightarrow S_{a_{j+1},c_{j+1}}(h(u_{i_{j+1}}))| = \epsilon$$

As a result, the required inequality $|f(u) \Leftrightarrow g(W \cdot u)| \leq \epsilon$ holds for all $u \in X^n$. Q.E.D.

The above theorem states that one multi-sigmoidal unit is necessary in order to represent (or approximate) any function $f : X^n \to \{0, 1\}$. As a direct consequence, λ multi-sigmoidal units are necessary to represent any function $f : X^n \to \{0, 1\}^\lambda$. Furthermore, since $\lceil \log_2 C \rceil$ bits are necessary to represent any number in $\{0, 1, \ldots, C \Leftrightarrow 1\}$ we have the following theorem:

Theorem 14 *If $X \subset \mathcal{R}$ is discrete and finite then any function $f : X^n \to \{0, 1, \ldots, C \Leftrightarrow 1\}^\lambda$ can be represented by a two-layer multi-sigmoidal network having $\lambda \lceil \log_2 C \rceil$ multi-sigmoidal units.*

On the other hand, given that a single MS unit cannot represent or uniformly approximate a function f such as

$$f(x, y) = [xy > 0]$$

the use of multi-layer (multi-sigmoidal) neural networks becomes an issue. In general, there is a trade-off between parallel and arbitrary hyperplanes. Since the number of adjustable parameters defines the dimensionality of the search space in a learning problem, it is important to maintain this number small or even to minimize it. However, in order to represent n parallel hyperplanes of a d-dimensional space $d + n$ parameters are required ($d + 1$ parameters to represent one hyperplane plus $n \Leftrightarrow 1$ parameters to represent the signed distances

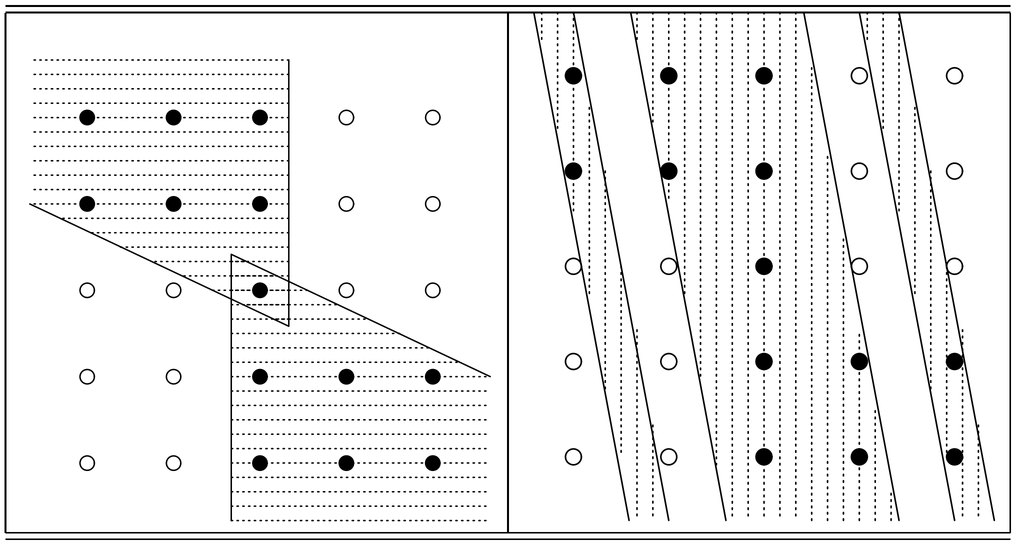


Figure 4.4: Two sets of points in 2-D space separated by arbitrary and parallel hyperplanes.

of this hyperplane from the rest). On the other hand, in order to represent m arbitrary hyperplanes of a d-dimensional space $m(d+1)$ parameters are required ($d+1$ parameters for each hyperplane). In all applications, it would be $m \leq n$. Yet, it is the ratio $\frac{n+d}{m(d+1)}$ that should be used to measure the utility and effectiveness of parallel hyperplanes (the smaller the ratio the more appropriate parallel hyperplanes would be for the problem at hand). The threshold is equal to 1.

As a concrete example, consider the case shown in figure 4.4 where $n = 6, m = 4, d = 2$, and $\frac{n+d}{m(d+1)} = \frac{8}{12}$. Parallel hyperplanes resulted in fewer free parameters on this problem despite the low dimensionality of the space ($d = 2$) and the choice of this example which is a discrete and finite instance of a problem that is not amenable to solution by a single set of parallel hyperplanes. Indeed as the number of samples would increase so would do the number of parallel hyperplanes that are required to separate the two classes. However, for this problem, we can always separate the two classes by four arbitrary hyperplanes. Yet, a typical neural network must have four layers in order to represent an exact solution to this problem. On the other hand, a three-layer MSNN can represent the same solution. In short, parallel hyperplanes can be very useful but not in all situations or networks.

However, if a function is defined over a discrete and finite set then a two-layer MSNN can represent that function. The same is true for ordinary three-layer (but not for two-layer) sigmoidal neural networks. Thus MSNNs seem to be more appropriate for discrete and finite problems than ordinary neural networks.

On the other hand, all the above discussion is valid only for classification problems or for functions that constant over finite sets of regions. In that case, we can use sets of hyperplanes (parallel or not) to approximate the boundaries of those regions. However, for all other problems and functions the concept of regions and their boundaries does not apply and so does the concept of parallel hyperplanes. In that case, as pointed out earlier in this chapter, the use of MS units provides non-monotonicity in unit activation functions. The remaining questions regard the utility of this feature and the existence of an effective learning algorithm under its presence.

To answer the first question, one can simply observe that non-monotonicity allows for efficient representation of any discrete relationships that may be present in the problem under consideration. Yet, there is a more fundamental justification of non-monotonicity. A simple inspection of the construction used in Kolmogorov's superposition mapping theorem (see [Kolmogorov, 1957, Sprecher, 1965]) reveals that upper layer unit activation functions must be non-monotonic in most cases. Adding to this the fact that for problems of n inputs only $2n + 1$ hidden units are required in the construction of that theorem, the trade-off between hidden units and non-monotonicity in unit activation functions becomes apparent in its most general form. It is that trade-off that justifies the use of non-monotonic activation functions. The choice of MS functions to model non-monotonicity is motivated by the study of sigmoidal functions in chapter 3 that demonstrated the effectiveness of sigmoidal functions in capturing non-linearity in some function spaces.

A final issue is the number of sigmoidal functions used in an MS unit. A direct application of the construction given in the proof of theorem 13 would not always result in the minimum number of sigmoidal functions. Consider the following example. Let, $f(x,y,z) = (x \wedge y) \vee (x \wedge \bar{z})$. Now, it is $X = \{0,1\}$. Thus (see proof of theorem 13), it would be $M = m = 1$, $B = 2$, and so it would be $h(x,y,z) = x + 2y + 4z$. This results to a multi-sigmoidal function g as shown in figure 4.5. However, if we define $h(x,y,z) = 4x + 2y + z$ we get the multi-sigmoidal function g' that has fewer sigmoidal functions than g. In addition, for the n-parity problem, it can be proved by induction that a weight vector as the one constructed in theorem 13 will require $\lfloor \frac{2^{n+1}-1}{3} \rfloor$ sigmoidal functions. On the other hand, setting all weights equal to a constant (say 1) will result in a multi-sigmoidal unit that can

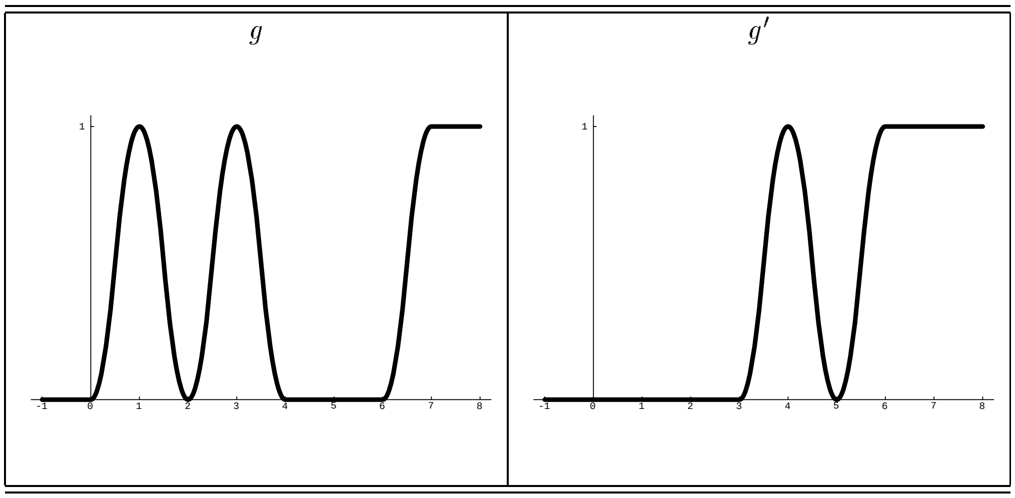


Figure 4.5: Multi-sigmoidal functions for $f(x, y, z) = (x \wedge y) \vee (x \wedge \bar{z})$.

represent n-parity using only n sigmoidal components.

In general, for any given problem, the weight vector W may affect the number of sigmoidal components of g. Finding a weight vector that would minimize the number of sigmoidal functions of an MS unit is an open problem and should be a subject of further research.

However, two-layer MSNNs cannot represent or (uniformly) approximate arbitrary functions by arbitrary precision $\epsilon > 0$. For example, a function as simple as $f(x, y) = [xy > 0]$ or the two-spiral problem cannot be modeled by a single multi-sigmoidal unit. On the other hand, multi-sigmoidal units are generalizations of ordinary sigmoidal units since every sigmoidal unit is a multi-sigmoidal unit with only one sigmoidal component. Thus, the functions that are representable by ordinary sigmoidal neural networks are representable by MSNNs, too. As a consequence, three-layer MSNNs can uniformly approximate any (Borel) measurable function [Hornik *et al.*, 1989].

Unfortunately, there has not been discovered a theorem such as theorem 13 that would provide weight vectors and multi-sigmoidal functions for multi-layer MSNNs. However, a heuristic and greedy algorithm (steepest descent) that trains multi-layer MSNNs and dynamically adapts the sigmoidal components on each unit is the multi-sigmoidal back propagation algorithm that is discussed in section 4.4.

4.3 Back-Propagation and Self-Configuring Neural Networks

The previous section left us with the problem of training multi-layer MSNNs. Common multi-layer feed-forward neural networks have been trained with the back-propagation algorithm (BP) [Rumelhart *et al.*, 1986a]. Although, BP has been very successful in producing good pattern classifiers, it has some notable drawbacks. First, it requires many training epochs to converge (see [Lippmann, 1989] [Mooney *et al.*, 1990]). Second, though BP adapts dynamically the weights in the network, the topology and connectivity of the network is static and predetermined. This may result in networks that are too large or too dense and so do not generalize well or to networks that are too small to learn the intended task.

Finally, BP does not adapt the activation function of the units and so it requires all interactions of the inputs that are significant to the classifier to be captured by the hidden units. Furthermore, in feed-forward neural networks, there is no direct communication between the hidden units in each hidden layer and so problems like the *moving target problem* appear [Fahlman and Lebiere, 1990].

It is remarkable that BP has been so much a successful learning algorithm despite all those problems. This is due, to some extent, to the fact that most of the above problems act as sources of inefficiency rather than as factors of solution quality degradation. As a result, BP produces good classifiers but is very slow when compared to other related algorithms (see [Mooney *et al.*, 1990] for a comparison of BP, ID3, and perceptron learning).

Different approaches have been taken so far to speed up BP and deal with the problems mentioned above. Among the most successful weight update algorithms is *Quickprop* [Fahlman, 1988] that assumes quadratic error surfaces in order to update weights more aggressively.

Algorithms that change the topology and/or connectivity of the networks are often termed *self-configuring* or *ontogenic* neural network learning algorithms. Ontogenic algorithms that mainly change the connections include Optimal Brain Damage [LeCun *et al.*, 1990], Optimal Brain Surgeon [Hassibi and Stork, 1993], and Occam's Razor [Thodberg, 1991]. On the other hand, ontogenic algorithms that update the topology of the network include Cascade-Correlation and Recurrent Cascade-Correlation [Fahlman and Lebiere, 1990, Fahlman, 1991], dynamic node creation (DNC) [Ash, 1989], node splitting [Wynne-Jones, 1992], meiosis networks [Hanson, 1990], and second order methods like skeletonization

[Mozer and Smolensky, 1989].

All the above ontogenic algorithms address some of the problems mentioned earlier and result in faster learning convergence and smaller networks which should generalize better according to Occam's Razor. However, all these methods change a network by either pruning or growing connections and/or nodes as needed. None of them changes the internal complexity of a unit itself.

Multi-sigmoidal neural networks, on the other hand, contain adjustable unit activation functions. Their learning algorithm –a modification of BP– is called MSBP. It dynamically creates or removes sigmoidal functions on each unit in order to build complicated non-monotonic activation functions. Hence, the local shape, monotonicity, and overall complexity of a unit activation function is dynamically controlled and changes radically during training and in different ways across different problems.

Static, non-monotonic activation functions have been used in value unit networks ([Dawson and Schopflocher, 1992]) and Radial Basis Function (RBF) networks [Poggio and Girosi, 1989]. A good discussion and some experimental results regarding the trade off between sigmoid and radial basis functions as neuron activation functions appear in [Weigend *et al.*, 1990, Moody and Darken, 1989, Lapedes and Farber, 1987]. An analytic comparisons of different static activation function appear in [DasGupta and Schnitger, 1993], while an empirical comparisons of different activation functions appear in [Moody and Yarvin, 1992]. Although some of the functions examined in [Moody and Yarvin, 1992] are parameterized by learnable parameters their overall complexity is static.

It is worth noting that multi-sigmoidal networks resemble more than traditional sigmoidal neural networks do the function mappings used to prove Kolmogorov's superposition mapping theorem [Kolmogorov, 1957, Sprecher, 1965]. Seeing those mappings as networks, one can notice that units do not have simple, static, and monotonic activation functions but rather complicated non-monotonic activation functions which are dynamically determined through an iterative procedure that is proved to converge exponentially fast to a solution. That result may provide a partial explanation of the claim that neural networks with non-monotonic and dynamic activation functions (such as multi-sigmoidal networks) should converge quickly to a solution.

Some speed-ups resulted from using non-monotonic activation functions, as reported in [Dawson and Schopflocher, 1992, Moody and Yarvin, 1992], are a first indication that interactions in the inputs can be captured by non-monotonicity in unit activation functions.

However, the use of a static Gaussian function in both value unit and RBF networks and the use of functions of static complexity in the experiments of [Moody and Yarvin, 1992] do not allow for very large speed-ups and, more importantly, do not completely describe the underlying trade-off between hidden units and non-monotonicity in unit activation functions.

In the next section, an adaptation of BP to train MSNNs is presented. The algorithm is called MSBP.

4.4 MSNNs and MSBP

Assuming a given error function E one can compute its derivative wrt each parameter in a neural network and derive a gradient descent learning algorithm in a way similar to that in [Rumelhart *et al.*, 1986a]. In the following formulae, we should have indexed each variable by the current training epoch t and the current input pattern p i.e. we should written E_{pt} to indicate the value of the error function on pattern p at the t-th epoch. However, in order to make the formulae easier to read, we dropped both of these indexes for they apply to all involved symbols. Now, if we define

$$\delta_{wj} = \Leftrightarrow\frac{\partial E}{\partial net_j}, \qquad \delta_{ajl} = \Leftrightarrow\frac{\partial E}{\partial a_{jl}}, \qquad \delta_{cjl} = \Leftrightarrow\frac{\partial E}{\partial c_{jl}}$$

where l is the sigmoid that is currently active on unit j (e.g. $b_{j,l-1} < net_j \leq b_{j,l}$) then we get the following update rules:

$$\Delta w_{ij} = \eta_w o_i \delta_{wj}, \qquad \Delta a_{jl} = \eta_a \delta_{ajl}, \qquad \Delta c_{jl} = \eta_c \delta_{cjl}$$

where η_w, η_a, and η_c are three different learning rates. We can compute deltas using the following recurrent equations:

$$\begin{aligned}
\delta_{wj} &= a_{jl}\gamma_j S'(a_{jl}net_j \Leftrightarrow c_{jl}) \\
\delta_{ajl} &= net_j\gamma_j S'(a_{jl}net_j \Leftrightarrow c_{jl}) \\
\delta_{cjl} &= \Leftrightarrow\gamma_j S'(a_{jl}net_j \Leftrightarrow c_{jl})
\end{aligned}$$

where

$$\gamma_j = \Leftrightarrow \frac{\partial E}{\partial o_j} = (\Leftrightarrow \frac{\partial E}{\partial o_j})[j : OU] + (\sum_k w_{jk}\delta_{wk})[j : IU]$$

and $j : OU$, $j : IU$ are true whenever j is an output or intermediate unit respectively.

The above algorithm will learn all parameters in a gradient descent fashion. Its learning of the affine coefficients (and subsequently of the barrier locations) would cause a local modification of the shape of the activation function but it would not change the number of the existing sigmoidal functions. Therefore, a policy is needed for creating new and deleting old sigmoidal functions. To this purpose, dx is defined to be the minimal distance of two successive barrier locations. If they come closer than dx one of them has to be removed (e.g. if $b_i + dx > b_{i+1}$ remove either b_i or b_{i+1}). In addition, if a barrier or a sequence of leftmost or rightmost sigmoidal functions have not been active in the last R_E epochs they are removed. R_E is an input parameter to the algorithm.

In order to add new sigmoidal functions, the error function is monitored. A criterion similar to that used in DNC to add new nodes [Ash, 1989] is used here in order to add new sigmoidal functions. The only difference is that a smoothed version of the error function is used instead of error itself. Assuming that E_t denotes the total error at the end of the t-th epoch, define

$$\begin{aligned} FE_t &= \lambda E_t + (1 \Leftrightarrow \lambda) FE_{t-1}, \quad t > t_0 \\ FE_{t_0} &= E_{t_0} \end{aligned}$$

where λ is a constant such that $0 < \lambda \leq 1$. The algorithm monitors this new smoothed error. It uses a threshold Δ_T called *trigger slope*. Whenever it detects a reduction of the slope of the function FE_t beyond the trigger slope at proportionally high values, new sigmoidal functions are added. The mathematical description of this condition is the following:

$$\frac{|FE_t \Leftrightarrow FE_{t-w}|}{FE_{t_0}} < \Delta_T \quad and \quad t \Leftrightarrow w \geq t_0$$

where w is the width of a window over which the current slope is determined.

The policy adopted in adding new sigmoidal functions requires that only a single barrier (or a single sigmoidal function) would be added on each selected unit each time the triggering criterion is activated. To this purpose, the values of γ_j or $\Leftrightarrow gamma_j$ (depending on whether $a_{jl} > 0$ or $a_{jl} < 0$) are accumulated at the corresponding active sigmoidal functions on each pass. The sigmoidal function with the maximum accumulated value that exceeds some predetermined threshold Δ_γ is then chosen and the unit is selected.

A single sigmoidal function or a barrier is added in the middle of the chosen sigmoidal function depending on whether that sigmoidal function was extreme (e.g. leftmost or rightmost) or not.

The above policy for adding sigmoidal functions can be considered as a form of gradient descent in a functional space defined by all possible sigmoidal configurations of an MS unit. More specifically, recall that adding new sigmoidal functions in an MS unit j would result in a change in the output o_j of j in a region X of its domain where the new sigmoidal function would be active. Thus, in a gradient descent fashion, such a change would be justified only if, over region X, $\frac{\partial E}{\partial o_j}$ would be negative or positive and o_j would be close to 0 or 1, respectively. In those cases, we should be able to reduce the error by increasing or decreasing o_j over X by adding a new sigmoidal function that would be active over X. Given that $\gamma_j = \Leftrightarrow \frac{\partial E}{\partial o_j}$ the choice of a sigmoidal function that maximizes the accumulated value of γ_j is again compatible with a gradient descent view that prefers the direction of highest descent.

MSBP, as described above, has been implemented and tested on some classification problems. Its performance is documented in the next section.

4.5 Experimental Results

The next subsection discusses convergence rates and solution quality for MSBP. Since solution quality is hard to estimate for problems where an optimal solution is not known, simple classification tasks with obvious optimal solutions are used here. On the other hand, more difficult and less artificial problems are used in section 4.5.2 to measure generalization performance of MSBP.

4.5.1 Convergence Rates and Solution Quality of MSBP

The MSBP algorithm described in the previous section requires a number of input parameters. Namely, learning rates (η_w, η_a, η_c), and adaptation thresholds $(dx, R_E, \Delta_T, w, \Delta_\gamma, \lambda)$. Finding their optimal settings for each problem is a laborious task. However, the algorithm performs very well for a wide range of settings. Hence, the solution adopted was to choose constant values for almost all the above parameters. In all the test cases reported below, it is $\eta_w = \eta_a = \eta_c = \eta$, $dx = 0.1$, $R_E = 10$, $\Delta_T = 0.05$, $\Delta_\gamma = 0.03$, and $\lambda = 0.25$). The only

Network	Units						
Name	I.	H.	O.	d	#DOF	min #DOF	$s_{\min}$
par$N$-bp	N	N	1	3	$(N+1)^2$	$(N+1)^2$	-
par$N$-ms	N	-	1	2	$N+2s$	$3N$	N
add$N$-M-bp	N	M	1	$M+2$	$O(3N(M+1)+\frac{M^2}{2})$	$9N$	-
add$N$-ms	N	-	1	2	$N^2+3N+2(N+1)s$	N^2+9N	$\frac{3N}{N+1}$
sym$N$-bp	N	2	1	3	$2N+5$	$2N+5$	-
sym$N$-ms	N	-	1	2	$N+2s$	$N+4$	2

Table 4.1: Network specifications

parameters varied are η and w.

The problems tested are parity, binary addition, and symmetry (see [Rumelhart *et al.*, 1986a] for a definition of those problems). Table 4.1 reports the structure of the networks as well as the number of degrees of freedom (or free parameters) in a general as well as the smallest known solution. The extensions "-bp" and "-ms" are used to indicate feed forward BP or MS neural networks, respectively. par$N$ stands for parity with N input bits (par2 is the famous XOR problem), add$N$ stands for binary addition of two N digit numbers, and sym$N$ stands for the symmetry problem with N inputs. I., H., and O. stand for input, hidden and output units. d is the depth of the network and #DOF is the total number of degrees of freedom in the network. For MSNNs, s indicates the average number of sigmoidal functions over all non-input units. $s_{\min}$ is the minimum value of s over all MSNN solutions of the corresponding classification problem.

Table 4.2 reports the best performance of BP and MSBP on the above problems. Each problem was tested 20 times and averages were taken. Epochs were counted in the way suggested in [Fahlman, 1988] i.e. restarts were allowed and the epochs spent in them were accumulated. "MAX" indicates the maximum number of epochs before restart. "fail" is the ratio of trials in which failure to converge within MAX epochs occurred. Finally, "Cross." is a quantity used to compare performance. It is based on the concept of *connection crossing* which appears to be a credible performance measure (see [Fahlman and Lebiere, 1990] for more details on this issue). In our measurements, a connection or BP unit crossing counts for one while an MS unit crossing counts for two crossings. This later assumption has been based on preliminary measurements of CPU time on random MS and BP units.

Interestingly, the number of epochs for MSNNs in the case of parity grew similarly to an

Network Name	η	w	MAX	Epochs	fail	s	Cross.	#DOF
par2-bp	3.8	-	3000	716.37	0.05	-	25789.26	9
par2-ms	1.9	1	100	22.90	0.00	2.95	366.40	7.9
par3-bp	0.3	-	12000	3101.89	0.05	-	397042.53	16
par3-ms	1.0	1	300	40.00	0.00	4.65	1600.00	12.3
par4-bp	3.7	-	45000	18919.11	0.10	-	7567644.89	25
par4-ms	0.8	1	1500	53.50	0.00	5.50	5136.00	15
par5-bp	1.3	-	150000	113293.60	0.25	-	130514272.73	36
par5-ms	0.6	1	4000	80.65	0.00	6.85	18065.64	18.7
par6-bp	0.2	-	500000	473492.81	0.25	-	185370484.62	49
par6-ms	0.3	3	15000	106.50	0.00	7.15	54528.00	20.3
add2-bp	3.9	-	8000	2103.88	0.15	-	605918.11	18
add2-3-bp	0.3	-	8000	2757.21	0.05	-	1588153.22	36
add2-4-bp	1.8	-	8000	546.80	0.00	-	411193.60	47
add2-5-bp	3.9	-	8000	393.40	0.00	-	371369.60	59
add2-ms	0.4	7	1000	150.10	0.00	4.30	38425.60	35.8
add3-4-bp	0.1	-	8000	13811.75	0.60	-	54805027.14	62
add3-5-bp	0.3	-	8000	3922.27	0.25	-	19328925.60	77
add3-6-bp	0.4	-	8000	856.60	0.00	-	5098483.00	93
add3-ms	0.2	11	1000	295.45	0.00	5.04	491628.80	58.32
sym6-bp	6.7	-	4000	618.83	0.10	-	673290.70	17
sym6-ms	1.0	26	1000	159.00	0.00	3.15	81408.00	12.3
sym8-bp	6.8	-	4000	4388.80	0.50	-	23594188.19	21
sym8-ms	1.1	58	1000	271.80	0.00	3.05	695808.00	13.1

Table 4.2: Performance of BP and MSBP

arithmetic progression $(22.90, 40.00, 53.50, 80.65, 106.50)$ with an average increment or 20.9 epochs. This extremely fast convergence for parity can be attributed to the fact that parallel hyperplanes are ideal devices for problems such as parity.

An important issue regarding MSNNs is the quality of the learned solution and the effect of the adaptation thresholds on that solution. Of course a measure of solution quality is generalization i.e. the ability of the learned solution to predict the values of a function on instances that are not in the training set. This is the subject of section 4.5.2. However, the value of s can provide some information regarding solution quality, too. In particular, in cases where $s_{\min}$ is known the ratio $\frac{s}{s_{\min}}$ can give a quite accurate picture of solution quality. Whenever that ratio is much larger than 1 overfitting occurs. If it is much less than 1 the under-fitting occurs. Finally, if it close or equal to 1 then the solution, with high probability, is of high quality.

To measure the ratio $\frac{s}{s_{\min}}$, each problem was tested 20 times and averages were taken as before. Many different values of learning rate η and window width w were used and the solutions were ordered according to the value of the above ratio. In figure 4.6, the ratio $\frac{s}{s_{\min}}$ is plotted over the number of epochs required to converge to a solution. Note that in 6 out of 9 cases the optimal solution (i.e. $\frac{s}{s_{\min}} = 1$) was reached and that in most cases the plotted ratio was less than two. The worst case is add2 which started with a ratio of 2.15 and converged to 1.46.

4.5.2 Generalization Performance of MSBP

Two classification problems that are more realistic than those used in the previous section were used in order to evaluate generalization capabilities of MSBP and compare MSBP with BP. The first problem regards classification of sonar signals using a neural network. The task is to train a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. The sonar data set consists of a training set of 104 samples and a test set of equal size (see [Gorman and Sejnowski, 1988], for more details).

The second problem regards classification of patients based on thyroid data. The problem is to determine whether a patient referred to the clinic is hypothyroid, normal, or hyperthyroid. The thyroid data set is relatively large consisting of 3772 training examples and 3428 test examples; a total of 7200 examples (see [Schiffman *et al.*, 1993] for more details).

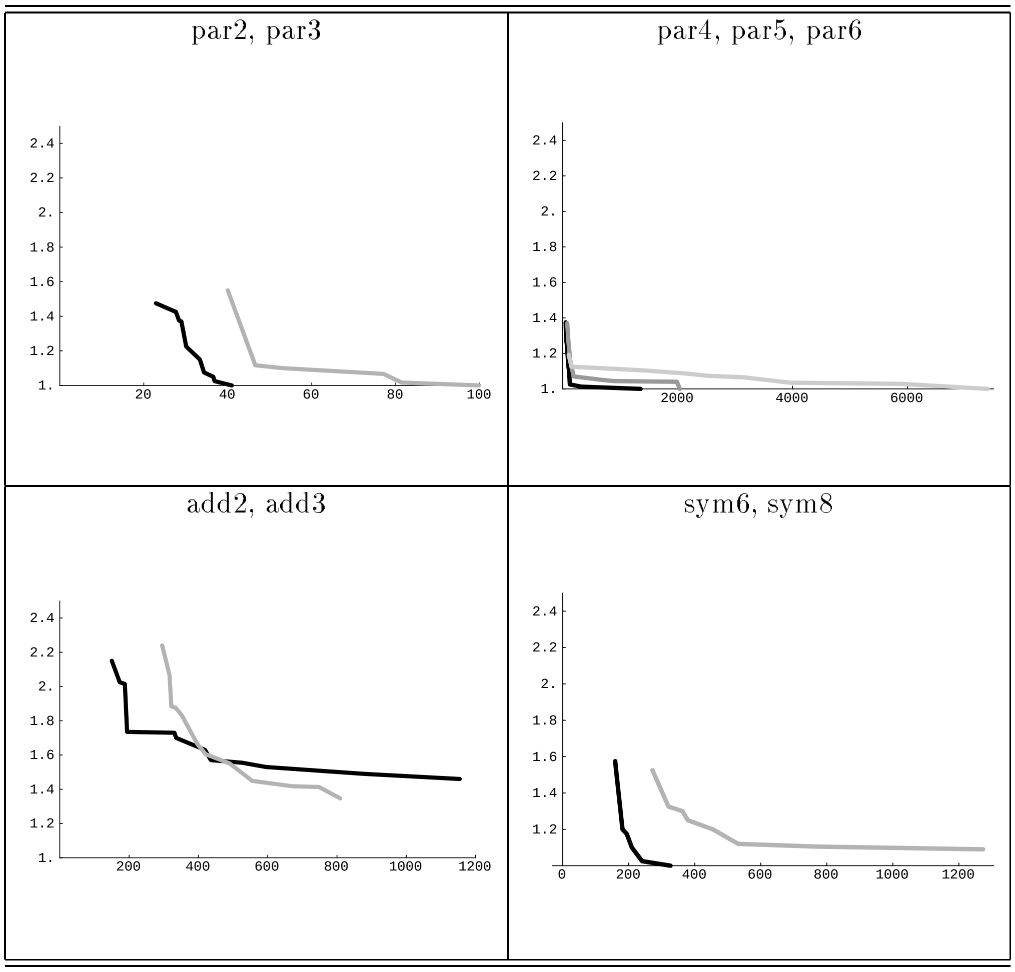


Figure 4.6: Plot of ratio $\frac{s}{s_{\min}}$ over number of epochs required to converge to a solution for the classification problems par2, par3, par4, par5, par6, add2, add3, sym6, and sym8.

Sonar data set				
Learning Algorithm	Hidden Units	#DOF	Generalization error	STD
BP	0	122.00	0.274	0.046
	2	128.00	0.158	0.054
	4	254.00	0.131	0.027
	8	506.00	0.112	0.021
	16	1010.00	0.114	0.017
MSBP	0	125.72	0.217	0.052
	2	133.92	0.141	0.056
	4	261.20	0.123	0.032
	8	516.00	0.109	0.028
	16	1028.00	0.111	0.020

Thyroid data set				
Learning Algorithm	Hidden Units	#DOF	Generalization error	STD
BP	4	103.00	0.044	0.011
	8	203.00	0.040	0.009
	16	403.00	0.046	0.006
MSBP	4	117.98	0.034	0.010
	8	218.40	0.040	0.012
	16	422.00	0.041	0.008

Table 4.3: Generalization error of BP and MSBP for sonar and thyroid data sets

For the sonar classification problem, one two-layer and four three-layer networks (with 2,4,8, and 16 hidden units respectively) were used. The number of epochs was fixed to 300 (as in [Gorman and Sejnowski, 1988]) for both BP and MSBP. Also most of the adaptation thresholds for MSBP were fixed ($\eta_w = \eta_a = \eta_c = \eta$, $dx = 0.05$, $R_E = 50$, $\Delta_T = 0.05$, $\Delta_\gamma = 0.5$, and $\lambda = 0.25$). The learning rate η for BP and MSBP as well as window width w for MSBP were determined through 10-fold cross validation on the training set. An estimation of generalization error was obtained by training the networks 10 times on the training set and computing the average of the classification error on the test set over all runs that converged to a relatively small classification error on the training set. Classification error was defined as the ratio of misclassified examples in a (training or test) set. Table 4.3 shows generalization error for the five networks used in our experiments. Figure 4.7 shows graphs of generalization error over number of hidden units and number of degrees of freedom (i.e.

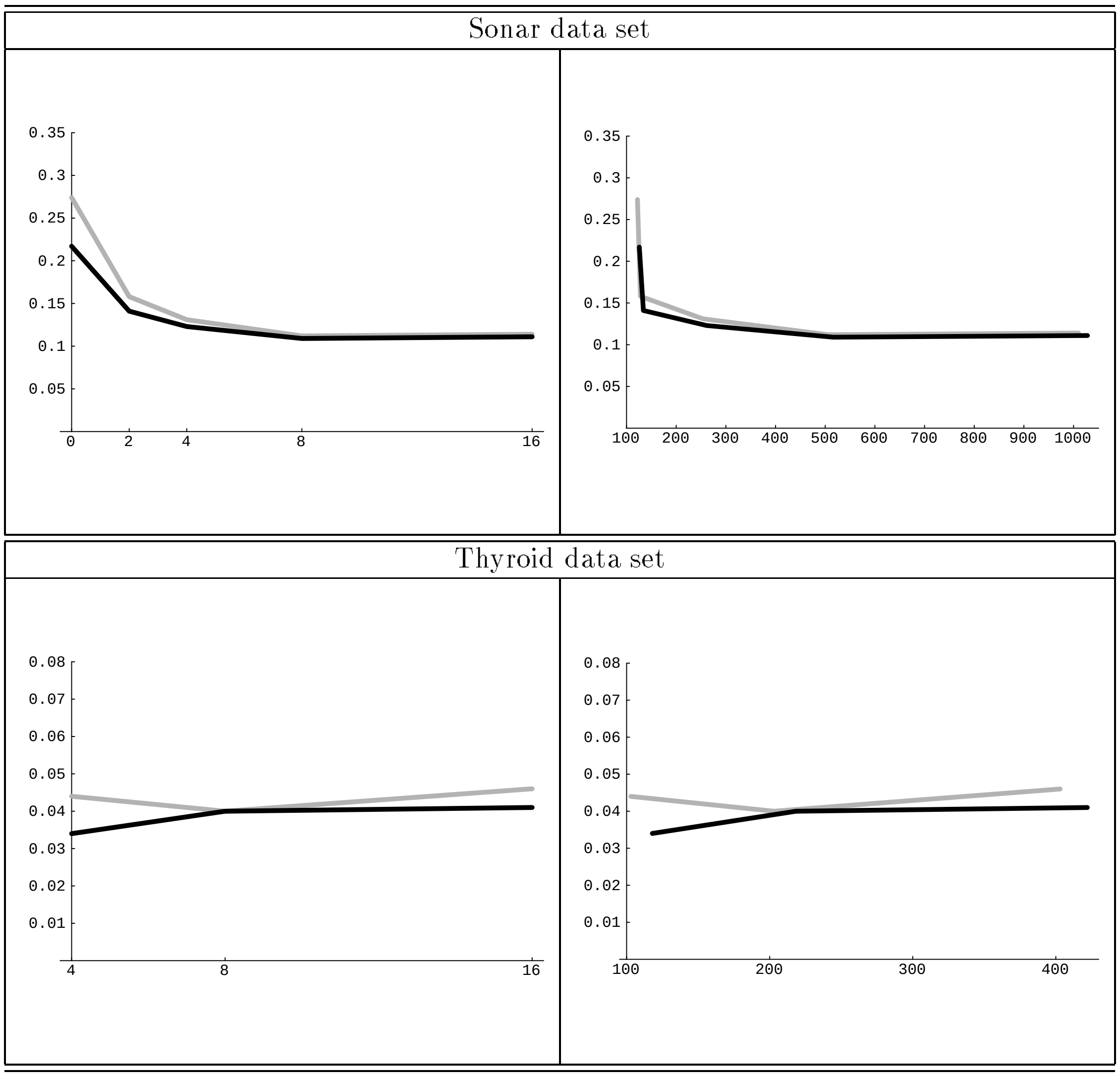


Figure 4.7: Generalization results for sonar and thyroid data sets. The graphs on the left show generalization error over number of hidden units while the graphs on the right show generalization error over number of degrees of freedom. The gray curves correspond to BP and the black ones to MSBP.

adjustable parameters) in the networks.

For the thyroid classification problem, three three-layer networks (with 4,8, and 16 hidden units respectively) were used. The number of epochs was fixed to 5000 (as in [Schiffman *et al.*, 1993]) for both BP and MSBP. Also most of the adaptation thresholds for MSBP were fixed ($\eta_w = \eta_a = \eta_c = \eta$, $dx = 0.05$, $R_E = 50$, $\Delta_T = 0.05$, $\Delta_\gamma = 5.0$, and $\lambda = 0.25$). Four different learning rates ($\eta = 0.01, 0.05, 0.1, 0.5$) and five different window widths ($w = 1000, 2000, 3000, 4000, 500$) were used. An estimation of generalization error was obtained by training the networks 4 times on the training set and computing the average of the classification error on the test set over all runs that converged to a relatively small classification error on the training set. The learning rates and window widths that resulted in the lowest classification error on the training set were selected. Classification error for those settings is reported in table 4.3. Again, classification error was defined as the ratio of misclassified examples in a (training or test) set. Figure 4.7 shows graphs of generalization error over number of hidden units and number of degrees of freedom (i.e. adjustable parameters) in the networks.

Manipulation and maintenance (i.e. addition or removal) of sigmoidal functions in each unit depends on the adaptation thresholds. However, the most important adaptation threshold turned out to be the window width w that determines the length of the window that is used to monitor the smoothed version of the error function (see section 4.4 for more details). Apparently, for a given value w of the window width and a bound M on the number of training epochs at most $\lfloor \frac{M}{w} \rfloor$ extra sigmoidal functions can be added in any MS unit.

In order to experimentally determine the effect of window width on generalization error, different window widths were used and experiments as those described above were performed. Generalization error was estimated for each window width. Figure 4.8 shows graphs of generalization error over window width for the sonar and thyroid data sets. For the sonar, three networks of 0, 2, and 4 hidden units were used while window widths that were integer multiples of 30 epochs were used. For the thyroid data set, three networks of 4, 8, and 16 hidden units were used while window widths that were integer multiples of 1000 epochs were used. As is apparent from those graphs, there is usually a set of optimal window width values that is dependent on the size of the network.

A general comment is that MSBP can improve generalization when there are not enough hidden units in the network by employing non-monotonic activation functions which are

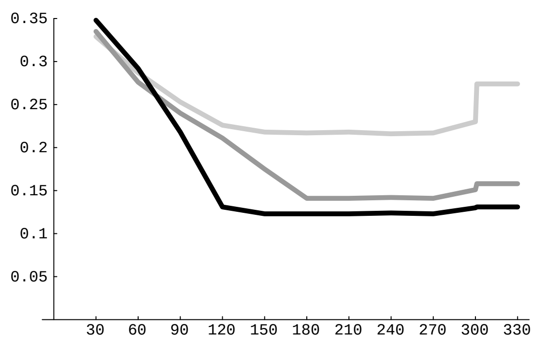
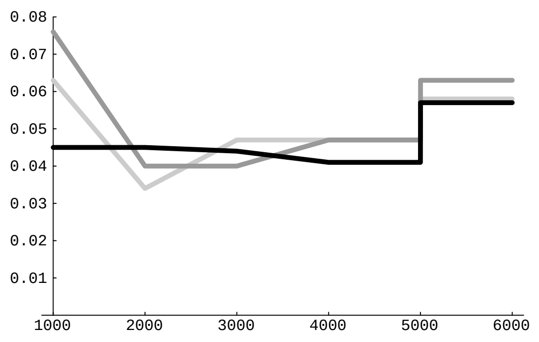
Sonar data set	Thyroid data set
	

Figure 4.8: Graphs of generalization error over window width (w). The three curves for sonar correspond to networks of 0, 2, and 4 hidden units while the three curves for thyroid correspond to networks of 4, 8, and 16 hidden units respectively. The darker the curve the more the hidden units.

interpreted as parallel hyperplanes in classification problems. Of course when the number of hidden units was large enough for BP to (approximately) capture the complexity of the underlying problem, MSBP was not able to provide an improvement by employing non-monotonicity (or parallel hyperplanes). In most of those cases, MSBP did not use any parallel hyperplanes at all.

Of course, it should be noted that the above is not a statement about MSNNs or non-monotonic activation functions but rather a statement about the limitations of MSBP. It is conceivable that another learning algorithm or a different policy in adding and removing sigmoidal components may result in a more effective use of non-monotonicity (or parallel hyperplanes) even in cases where the number of hidden units is relatively large. The above experiments provide only a brief empirical evaluation of MSBP. Further studies and maybe development of new learning algorithms or policies are required in order to understand in more detail the trade-off between functional representations that are based on units with monotonic and non-monotonic activation functions.

4.6 Future Extensions and Conclusion

As shown in the previous section, MSBP is effective in training MSNNs to obtain solutions that are either nearly optimal or generalize well. Of course, many more experiments need to be performed in order to determine accurately the power and maybe the limitations of those networks and their learning algorithm. Their current performance only underlines their importance. Yet, the MSBP algorithm can be extended and improved in many ways. For example, more aggressive weight schemes like Quick-prop can be employed, ontogenic methods may be used to create new MS units, and more efficient policies in adding and deleting sigmoidal functions can be found. Furthermore, it is possible to extend the algorithm so that it would dynamically set the threshold values for its input parameters by monitoring the error function. It is conceivable that the above extensions may further improve performance and generalization.

The introduction and formalization of MSNNs here revealed the existence of a trade-off in capturing dependencies and functionality through hidden-units or through non-monotonic activation functions. The interaction of those two mechanisms is co-operative rather than antagonistic for either mechanism does not exclude the other. They are, in some sense, "orthogonal". Furthermore, it is possible that optimal networks may have many layers consisting of a combination of MS and simple sigmoidal units. Detailed studies are required in order to get more accurate knowledge of this kind of interaction. The results reported here show clearly that such knowledge can help us build smaller and more efficient networks and improve our understanding of functional representations.

Chapter 5

tFPR: A Hybrid Pattern Recognition System

I must Create a System, ...

Jerusalem, pl.10, l.20
William Blake 1757–1827

The theoretical results and functional formalisms developed and presented in the previous chapters allow for the definition and specification of a hybrid pattern recognition system and architecture such as tFPR. tFPR is a hybrid fuzzy, structural, and neural pattern recognition system that uses fuzzy sets to represent multi-variate pattern classes that can be either static or dynamic depending on time or some other parameter space. The membership functions of the fuzzy sets that represent pattern classes are modeled by sigmoidal functions. The choice of sigmoidal functions was motivated by their ability to represent efficiently and concisely different multi-variate pattern classes. Given a set of input data and a pattern class specification, tFPR evaluates the pattern class specification for the input data by computing the grade of membership of the data in the fuzzy set that corresponds to the current pattern class. The result of this evaluation is an estimate of the degree to which the data match the pattern class specification. The input data may be a number of time-dependent signals whose past values may influence the evaluation of the pattern class. In that case, structural pattern recognition methods, in addition to fuzzy ones, are employed in order to match curves of arbitrary dimensionality (rather than points) in the input domain. Two different structural (segmentation) methods are presented. The first method is a very efficient greedy algorithm that is linear to the number of input data points while the second method is an algorithm that is optimal under some conditions but is slower than the first method. Although efficiency is a very important consideration in tFPR, the main issues are knowledge acquisition and knowledge representation (in terms of pattern class descriptions). tFPR has been implemented in the BB1 blackboard architecture. It is currently being applied in a system for medical monitoring. Potential applications in other domains are under investigation.

5.1 Introduction

The pattern recognition methods used in tFPR are fuzzy for they use fuzzy sets to represent pattern classes. However, some of them are neural pattern recognition methods for they use multi-sigmoidal neural networks to model the membership function of the corresponding fuzzy sets. Finally, the pattern recognition methods used in tFPR for parameter-dependent pattern classes are structural for they decompose a non-static parameter-dependent pattern class description into a number of primitive pattern elements or *segments* and compute the evaluation of the overall pattern class on the current input by evaluating the segments and combining their evaluations into a single evaluation.

In our approach, a pattern class evaluation is an assessment of the degree to which the available input data match a pattern class of interest. The input data correspond to a set of values of a number of variables which are called *input features* or *pattern measures*. A pattern class evaluation may use the values of the corresponding pattern measures at some point in time (usually the current time) or may require the examination of curves or surfaces of the proper dimensionality that represent the behavior of pattern measures through time or some other *parameter space*. Furthermore, arbitrary combinations of such multi-dimensional objects with points or other objects of different dimensionality are also allowed.

In our system, the above computations are based on fuzzy membership functions. A method for generating those functions is, thus, necessary. Although a wide range of membership functions have been used in the literature ([Yager *et al.*, 1987], [Kaufmann, 1975, ch.III, section 29], [Gupta *et al.*, 1975]) we choose to use a particular class of functions derived from the sigmoidal family of functions (see chapter 3). Arguments for the suitability of sigmoidal functions in modeling fuzzy set membership have been given in [Drakopoulos, 1994b] as well as in chapter 3. A brief presentation of those arguments as well as the basic model used in tFPR and its class of sigmoidal functions are given in the next section. In section 5.3, an extension to the basic model to include time or any parameter-space dependent signals and pattern classes is presented. Algorithms that can support this model are presented in section 5.4. In section 5.5, a further extension of the method to include a wide range of operators as well as its particular implementation (tFPR) in the BB1 blackboard architecture is presented. In section 5.6, example applications are presented, and in section 5.7 specifications of a graphical interface for defining pattern classes are given. Finally, in section 5.8, future extensions and enhancements of the system are discussed.

5.2 The Underlying Model

Figure 5.1 depicts the basic idea behind the architecture of our system. The basic building block of our system and essential ingredient of its methods is the *pattern object*, i.e. an object that describes a pattern class of interest. Having a collection of such pattern objects stored in a *pattern base*, we can build a *pattern manager* that evaluates a pattern object on the current input whenever input related to that pattern object is presented to the system. Each such evaluation results in a continuous valued signal in [0, 1] that is the value of the membership function of the fuzzy set that describes the pattern class. Thus, we have to define a model for pattern objects and, in particular, for the membership functions of the corresponding fuzzy sets.

In the most general case, a pattern object would be associated with many pattern measures that have different and maybe non-numeric ranges. A series of transformations may be required to transform the pattern measure values into a domain which would be the domain of a fuzzy set. Let as call this the *x-domain*. Then an application of the membership function of the fuzzy set would result in the desired membership value in the interval [0, 1]. Let as call this interval the *y-domain*

Now, in order to define formally pattern objects assume the following:

p	is a pattern class
M_p	is the set of pattern measures associated with p
$R[M_p]$	is the Cartesian product of the ranges of the measures in M_p
$\mathcal{R}$	is the set of real numbers

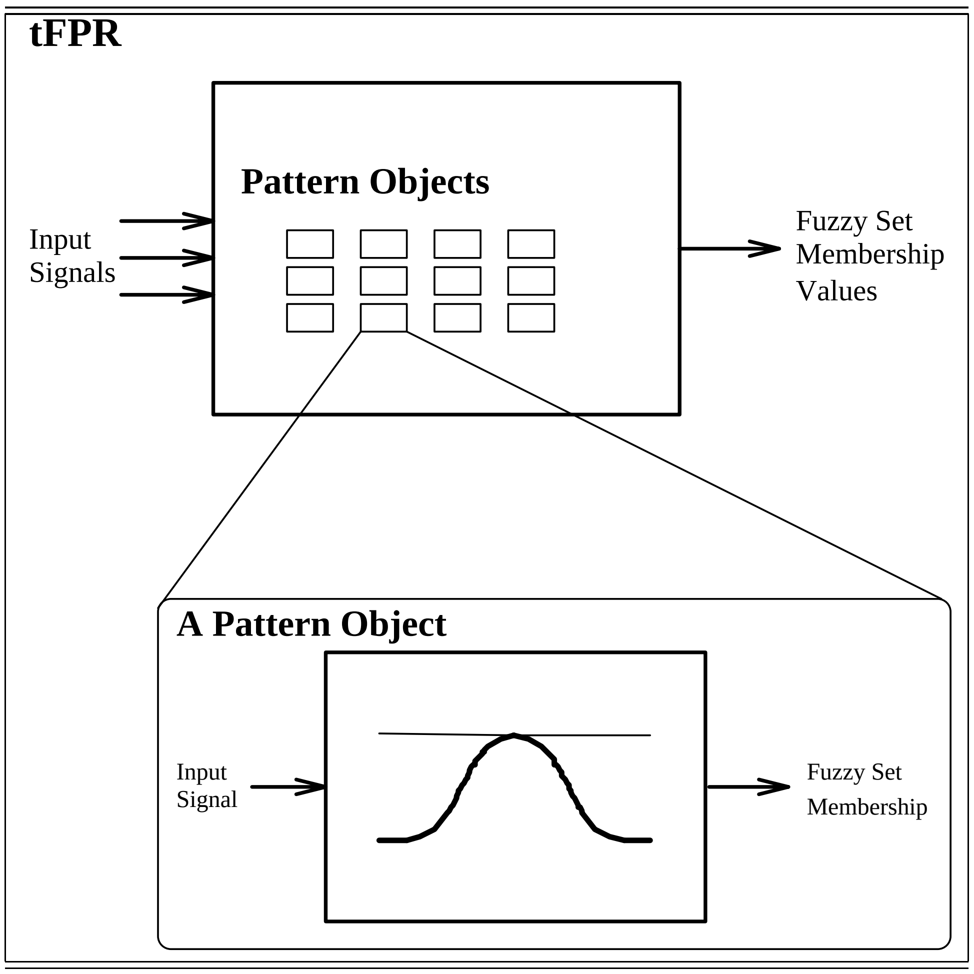


Figure 5.1: Basic structure of tFPR

Then, a pattern object can be defined as follows:

Definition 16 *A pattern object corresponding to a pattern class p is a tuple*

$$(M_p, A_p, v_p, n_p, a_p, d_p, C_p)$$

where

M_p	*is the set of pattern measures associated with p*
a_p, d_p	*are integers*
A_p	*is a function from $R[M_p]$ to $\mathcal{R}^{a_p}$*
v_p	*is a function from $\mathcal{R}^{a_p}$ to $\mathcal{R}^{d_p}$*
n_p	*is a function from $\mathcal{R}^{d_p}$ to $[0, 1]$*
C_p	*is a set of conditions*

C_p is a set of conditions that determine the applicability of pattern class p and can be used to enforce or inhibit an evaluation of the corresponding pattern object at some point in time. There is not anything special about the number of mapping functions used in a pattern object. If one would replace A_p, v_p, and n_p by a function f_p that is the composition of those three functions he should get a functionally equivalent pattern class definition that would involve only a single mapping function. Our dividing the mapping function process into three parts was primarily motivated by the difference in functionality of the tasks these three functions perform but is not otherwise required. Furthermore, putting the whole mapping into a single function would result in complicated and hard-to-understand functions. In short, the most general solution would allow the user to define an arbitrary sequence of functions to be applied to the input data and the most restrictive one would allow only one such function. Our choice stands in the middle.

The transformations A_p and v_p are arbitrary. As a concrete example, consider the case where the input is a non-stationary signal $x(t)$, such as the sound of a bird, that is stored in database of input data with a symbolic name such as `bird_sound_3`. Furthermore, assume that the pattern class under consideration is defined in terms of a number of conditions upon the peak energy values of the second and third harmonic of $x(t)$. In that case, M_p should contain as its sole element the symbolic name `bird_sound_3` to identify the source of the input signal for its pattern object. A_p would compute the spectrogram of $x(t)$ over some time window W. Since the spectrogram is a function of energy over time and frequency, the output of A_p should be triplets of the form (*time, frequency, energy*). Thus, A_p should map the set $\{x(t) \ / \ t \in W\}$ to $\mathcal{R}^3$ and so a_p must be equal to 3. Subsequently, for each frequency present in the output of A_p, v_p would produce its peak energy value over time. Thus, v_p would be a function from $\mathcal{R}^3$ to $\mathcal{R}^2$ and so it must be $d_p = 2$. Finally, n_p would be a configuration of sigmoidal functions as complicated as a multi-sigmoidal network. Its function would be to test the values v_p assigns to the second and third harmonic in order to determine to what degree they satisfy a given set of conditions which define the current pattern class.

In tFPR, the specification of membership functions n_p that map from $\mathcal{R}^{d_p}$ to [0, 1] is always based on sigmoidal functions. The choice of sigmoidal functions was motivated by the *sigmoidal bubble theorem* (see chapter 3 for more details). To illustrate the issue more clearly, we present here a modified version of the theorem that makes direct references to fuzzy sets. To this purpose, let S indicate a *basic unit sigmoidal function* and A_S indicate the affine sigmoidal class of S i.e.

$$A_S = \{S_{a,c} \ / \ a, c \in \mathcal{R}, \quad a \neq 0\}$$

where $\forall x \ \ S_{a,c}(x) = S(ax \Leftrightarrow c)$. Now observe that if f_{B_i} is the membership function of the fuzzy set B_i and $f_{B_i} \in A_S$ (i.e. $f_{B_i} = S_{a_i,c_i}$, for some $a_i, c_i \in \mathcal{R}$) then

$$\begin{aligned}
f_{B_i \cup B_j}(x) &= \sup\{f_{B_i}(x), f_{B_j}(x)\} &&= S(\sup\{a_i x \Leftrightarrow c_i, \ a_j x \Leftrightarrow c_j\}) \\
f_{B_i \cap B_j}(x) &= \inf\{f_{B_i}(x), f_{B_j}(x)\} &&= S(\inf\{a_i x \Leftrightarrow c_i, \ a_j x \Leftrightarrow c_j\}) \\
f_{\bar{B}_i}(x) &= 1 \Leftrightarrow f_{B_i}(x) &&= S(\Leftrightarrow a_i x + c_i)
\end{aligned}$$

Therefore, we can always interchange sigmoidal functions with sup, inf, and complementation operations. As a result, if all the constituent subexpressions of a fuzzy set theoretic expression are evaluated through sigmoidal functions of a single class then only a single sigmoidal computation is required to evaluate the overall expression no matter how complicated the expression is!

Furthermore, each sigmoidal function that belongs to a particular affine class can now be represented by a pair of numbers (a, c). This reduces the storage requirements of sigmoidal functions to a level comparable to that of linear segments and makes them very attractive as representation functions. As reported in [Drakopoulos, 1994b], some preliminary experimentation with the tFPR system showed that sigmoidal functions outperformed linear segments and lower order polynomials.

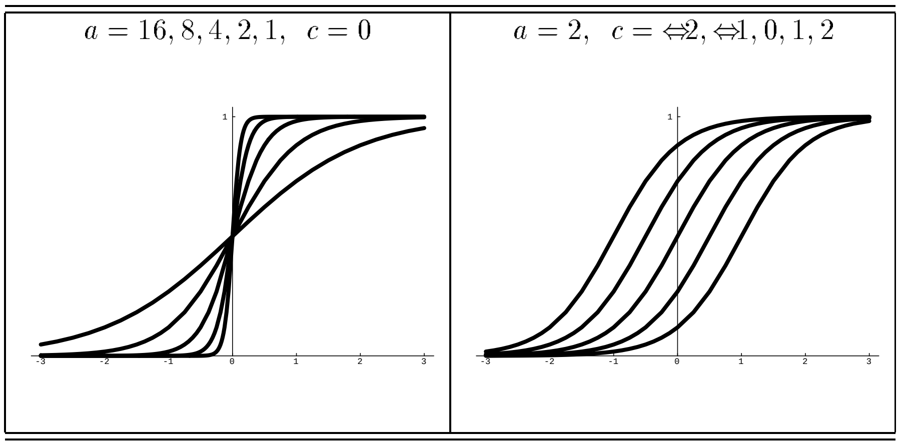


Figure 5.2: Sigmoidal functions for various values of a, c.

Finally, it has been shown in ([Drakopoulos, 1994b]) that sigmoidal functions are the only functions that posses the above properties.

Figure 5.2 shows the functions $S_{a,c}$ for various values of a, c and figure 5.3 shows fuzzy set operations on sigmoidal functions.

Now, sigmoidal functions can be used to approximate arbitrary membership functions. As an example of membership function approximation, consider the situation depicted in figure 5.4 where the membership function $y(x)$ is defined in the following way :

$$y(x) = \left\{ \begin{array}{ll} 0 & \text{if } x < x_1 \\ \frac{x - x_1}{x_2 - x_1} & \text{if } x_1 \leq x < x_2 \\ 1 & \text{if } x_2 \leq x < x_3 \\ \frac{x_4 - x}{x_4 - x_3} & \text{if } x_3 \leq x < x_4 \\ 0 & \text{if } x_4 \leq x \end{array} \right\}$$

Now, let S be the sigmoidal function that passes through the points $(x_1,\ \epsilon)$ and $(x_2,\ 1 \Leftrightarrow \epsilon)$ and S' be the sigmoidal that passes through $(x_3,\ 1 \Leftrightarrow \epsilon)$ and $(x_4,\ \epsilon)$, where ϵ is a small positive constant. Then the combination $\min(S, S')$ is a good approximation of y as shown in figure 5.4.

Now, let us call *set operations* the traditional fuzzy set operations (e.g. fuzzy set union, intersection, and complementation). We have shown that sigmoidal functions combine very nicely with set operations. However, it is not possible to approximate any arbitrary function

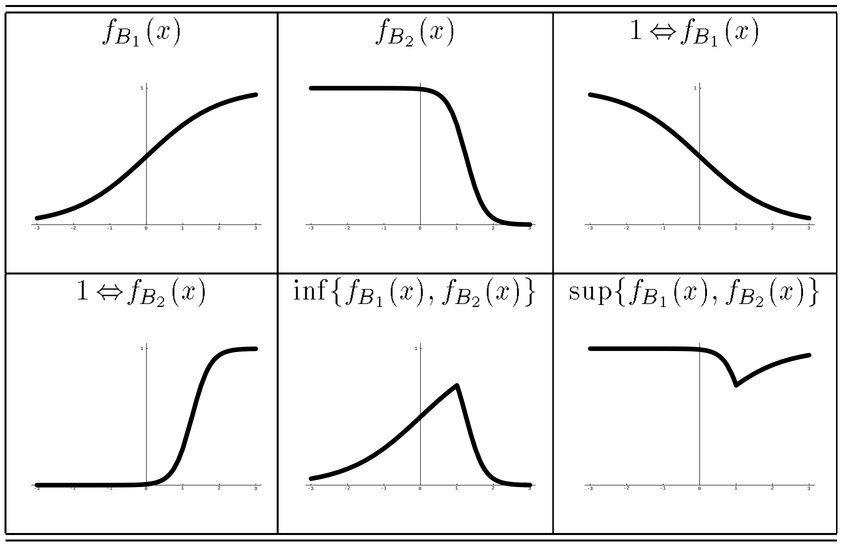


Figure 5.3: Fuzzy set operations on sigmoidal functions.

using only set operations on sigmoidal functions. To solve this problem, the domain of the fuzzy sets (x-domain) was divided into appropriate intervals and set operations were applied within each interval to approximate a given function. The extra conditions that define the set of intervals of a pattern class are called *interval conditions*. The resulting functions are piece-wise sigmoidal functions. Using an appropriately large number of intervals, a sigmoidal approximation of an arbitrary function (similar to its polygonal approximation) can be found.

The use of interval conditions extends the representational spectrum of set operations to include new functions. Furthermore, since only set operations are performed within each interval, only a single sigmoidal computation per pattern class is required.

However, the emphasis in tFPR is on knowledge acquisition and knowledge representation. An extended set of tFPR operators would conform to that principle. Furthermore, even in cases where new tFPR operators would require more than one sigmoidal computation per pattern class, efficiency would not be sacrificed or compromised in any way. On the contrary, sometimes it would be nearly intractable to approximate a given function using only set operations and interval conditions for the number of intervals may grow enormously high. In those cases, extra tFPR operators (such as summations or weighted summations) become not only useful but also resourceful.

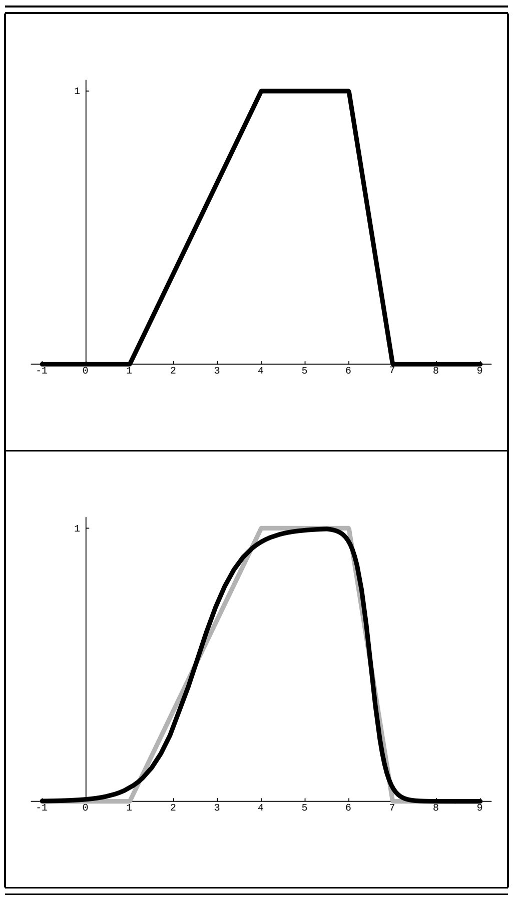


Figure 5.4: A pattern class and its sigmoidal approximation.

In short, an extended tFPR operator set not only would provide an agent with higher flexibility in defining pattern classes and reduce the overall knowledge acquisition effort but also should result in more effective pattern class specifications. The set of operators currently used in tFPR is described in [Drakopoulos and Hayes-Roth, 1994, Drakopoulos, 1994c] and is briefly presented in section 5.5.

In the next section, we present an extension of the basic model presented above that would deal with dynamic pattern classes that depend on time or any other parameter space.

5.3 The Extended Model

An extension of the basic model allows pattern measures to be arbitrary functions instead of arbitrary values in a domain. This extension requires the system to deal with multi-dimensional curves (e.g. parametric functions) instead of points (e.g. vectors). To this purpose, pattern classes and objects are defined over some parameter space. The extension to parameter-space-dependent pattern objects is very useful in cases where, as an example, the definition of a pattern class requires access not only to current, but also to past values of pattern measures. In that case, the paramenter space is time and pattern measures are time-dependent functions.

In the model below, we use a single one-dimensional parameter space not only because this simplifies our presentation but also because the extension of our model to other multi-dimensional parameter spaces is straightforward.

In tFPR, curves and surfaces are broken into *segments*, transforming the problem of representing arbitrary shapes into a problem of segmentation and segment specification. Each segment consists of a basic ideal curve —we call it the *ridge* of the segment— that can be described by a small amount of information in terms of conditions upon the literal values of the curve and the values of its k-order derivatives with respect to the parameter space (for $k = 1, 2, \ldots$). A number of sigmoidal functions are used to evaluate the degree of similarity between a portion of a curve in the input data and a segment's ridge by matching the values and the derivatives of the curve against those of the ridge. Those functions produce an evaluation (y-value) for each segment. All the evaluations are eventually combined to produce an overall evaluation of the whole curve. This way, both description (representation) and recognition (matching) of arbitrary curves can be accomplished using exactly the same primitives and formalism. In tFPR, we use only two primitives for describing segments and a single one for combining them.

Uniformity in the representation and evaluation of functions has not been our primary motive in choosing to segment curves. In fact, our choice was initially based on a very basic observation. When people try to describe a trajectory, a curve, or a surface, most of the time, they describe it as a collection of segments over a set of regions in which some very basic properties of the curve or the surface (such as curvature, rate of growth/decline, range, etc.) are, more or less, within some specifiable range. Thus, segmentation seems to be the natural approach to the problem.

In tFPR, in addition to segmentation functions, we use some basic functions to extract some features of the actual values of a segment (such as maximal or minimal value, duration, etc.) whenever the pattern object evaluation depends on those features, too. The subsequent evaluations of those features are used in the overall evaluation of the pattern object and often help in determining the segments themselves. For example, consider a pattern class that requires its input to stay at zero for some time then start increasing with a bounded range of rates up to a value M_3, then stay constant for a while, and then fall back to zero (again with bounded rate) and stay there. The corresponding pattern object could consist of 5 segments :

segment 1:	constant at 0 (with some tolerance T_1)
segment 2:	going from 0 to M_3 with rate bounded by r_{2l} and r_{2h}
segment 3:	constant at M_3 (with tolerance T_3)
segment 4:	going from M_3 to 0 with rate bounded by r_{4l} and r_{4h}
segment 5:	constant at 0 (with tolerance T_5)

We can also add another kind of constraint that would relate properties of different segments. For example, we may require the duration of segment 3 to be greater than the duration of each of segments 2 and 4 but less than 10 minutes. In addition, we may require the maximum value of the input in segment 1 to be less than the corresponding maximum value in segment 5.

In the next section we present algorithms (*segmentation algorithms*) that combine segments together in order to match (evaluate) arbitrary curves.

5.4 Segmentation Algorithms

Our efforts to build tFPR led to the development of a number of segmentation algorithms. Each of then has its own characteristics, precision, and noise tolerance as well as complexity and performance. Here, we present two of those algorithms which we consider to be the most important ones. The first one is a greedy non-optimal algorithm that is linear with the input data set size. That algorithm is of practical interest due to the apparently high requirements for speed in a system that deals with a large number of pattern objects. The second algorithm is polynomial yet optimal when the cost function is additive and there are no constraints among features of different segments similar to those described in the previous section. The above requirements as well as the higher time complexity of the second algorithm severely limit its practicality and applicability in any system that contains a large number of pattern objects. We attribute to it only theoretical importance for it shows that the segmentation problem is polynomial in cases where the algorithm applies. Both of these algorithms scan the input in reverse temporal order (right to left) and dynamically collect new data as needed to match the pattern object. The implicit assumption is that the last (i.e. rightmost) segment of the corresponding pattern object ends at a given point in time which is used as the starting point for a right-to-left trip.

The choice of right to left evaluation was initially motivated by the fact that the most recent data are most probably the most important ones. In addition, the opposite (i.e. left to right) evaluation would require some state bookkeeping in order to have the same computational complexity as the right to left evaluation. This state information can grow linearly with the input data set size, a fact that forbids its use on systems with large input data sets. Furthermore, that state, even in cases where it would not grow prohibitively large, would degrade the actual run-time performance of the pattern evaluation functions for it enlarges the active data set of the running code (see [Peterson and Silberschatz, 1985, section 6.8]).

The greedy algorithm developed is called the *RL-algorithm* and is very simple. Given a pattern object consisting of P segments $(S_1, \ldots, S_P)$, their weights $(w_1, \ldots, w_P)$, a number of input signals $(d_1, \ldots, d_I)$ and the current time t (ending point), it computes a sequence of times $(T_0, \ldots, T_P)$ that define the segments (segment i lasts from T_{i-1} to T_i) as well as an evaluation of that segmentation. It works from present to past, keeping track of the number of sampled data-points n_i and the current evaluation, for each segment S_i. Initially, the last point belongs to the last segment S_P which is the current segment. At each step, the algorithm decides whether the next point (previous in time) should belong to the current segment (S_i) or to the next one (S_{i-1}). The criterion used to make that decision is the local maximization of the evaluation function. The algorithm also computes the earliest and latest end (L_i and R_i, respectively) of each segment in order to compute durations and other segment features that may be required in order to evaluate constraints among segments during the segmentation process.

Termination occurs whenever the algorithm has matched all the segments or when the matching process is at an intermediate segment and the evaluations of the remaining segments to be matched cannot change significantly the matching process outcome either by perfect match or mismatch. Eventually, for some p $(1 \leq p \leq P)$, it would be

$$\forall i \in \{p, p+1, \ldots, P\} \quad \left\{ \begin{array}{ccc} L_i & = & T_{i-1} \\ R_i & = & T_i \end{array} \right\}$$

where S_p is the segment where the matching process was terminated. These times are also used to compute durations and can help in constraint evaluation during the process of segmentation. This algorithm runs in $O(n)$ time (where n is the number of data points used) since it spends $O(1)$ time at each point to compare two evaluations of two competing (e.g. neighboring) segments and choose the greater one. It is not optimal in the sense that the evaluation it returns may be affected by the presence of noise or by neighboring segments whose definitions overlap.

The second algorithm is optimal in cases where there are no special constraints among features of segments and the segmentation cost function is additive. This algorithm essentially is a modification of Dijkstra's shortest path algorithm ([Papadimitriou and Steiglitz, 1982]) applied to a graph of nP nodes. Each node in that graph is associated with a pair of numbers (i, j) (for $i = 1, \ldots, P$, and $j = 1, \ldots, n$) and is connected to the nodes of the graph that correspond to pairs $(i,\ j \Leftrightarrow 1)$ and $(i \Leftrightarrow 1,\ j \Leftrightarrow 1)$ (figure 5.5). Intuitively, going from (i, j) to $(i,\ j \Leftrightarrow 1)$ means that data D collected at time t_{j-1} belong to segment S_i. On the other hand, going from (i, j) to $(i \Leftrightarrow 1,\ j \Leftrightarrow 1)$ means that the matching of segment S_i ends and the matching of segment S_{i-1} starts at t_{j-1} (i.e. $T_{i-1} = t_{j-1}$). In short, matching consequent data to the same segment corresponds to horizontal transitions while switching segments corresponds to diagonal transitions in the graph of figure 5.5.

Now, if we associate costs with the graph's links to reflect changes in the evaluation

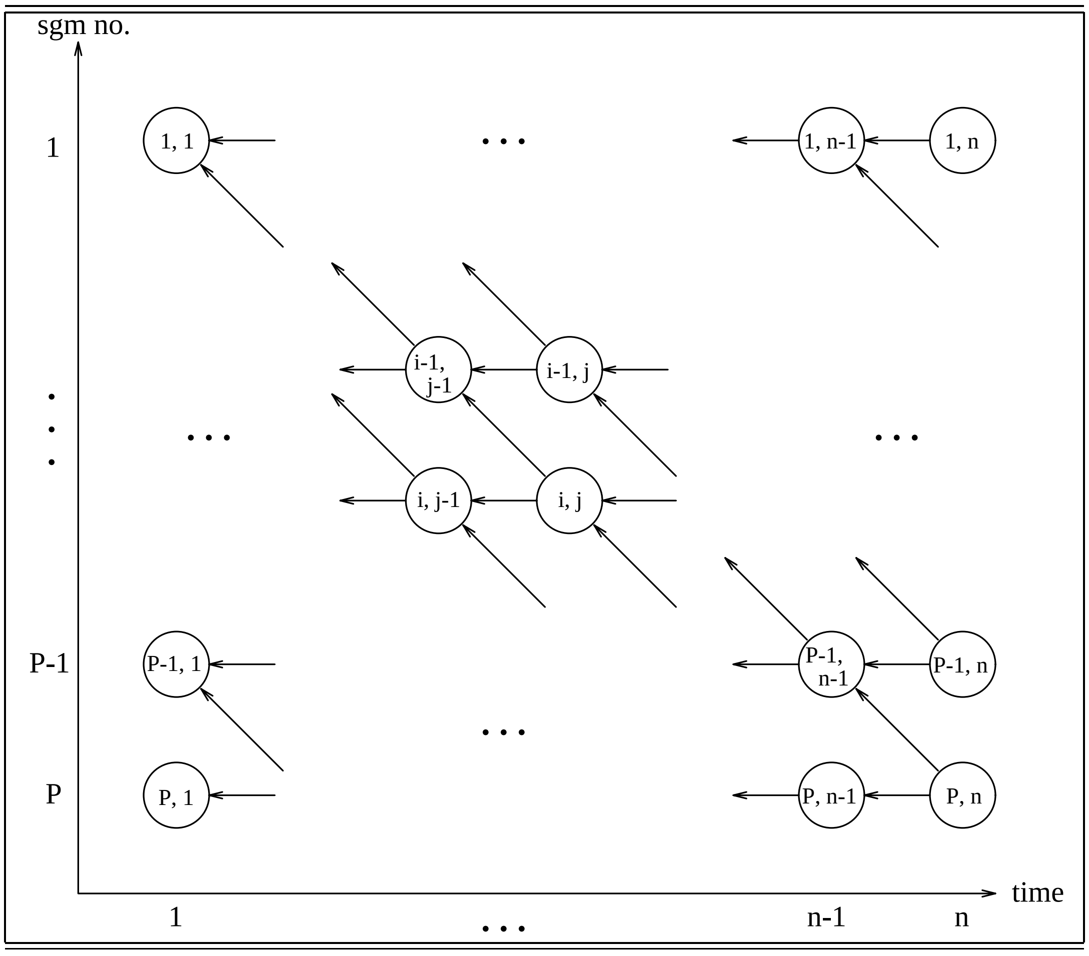


Figure 5.5: Search graph for the optimal algorithm.

of a pattern object as we follow paths in this graph, we can solve the input segmentation problem by finding a maximum cost path that goes from (P, t) to $(1, t')$, for some $t' < t$. Obviously a path denotes a particular segmentation of the data while its cost measures the degree in which this segmentation matches the current pattern object.

We can further simplify this optimal algorithm by observing that each node in the graph has exactly two immediate successors. Assuming that $MC(i, j)$ indicates the maximum cost path to node (i, j) we can solve the maximum cost path problem by traversing the graph from right to left (i.e from $j = n$ down to $j = 1$) and solving the following recurrent equation:

$$MC(i \Leftrightarrow 1,\ j \Leftrightarrow 1) = \max\{MC(i \Leftrightarrow 1,\ j),\ MC(i,\ j)\}\ +\ v(S_{i-1},\ t_{j-1})$$

with initial conditions :

$$MC(i,\ n) \quad = \quad v(S_i,\ t_n) \quad \forall i \in \{1, \ldots, P\}$$

where $v(S, t)$ denotes the evaluation of segment S at time t. The complexity of this algorithm is $O(n^2 P^2)$.

The optimal algorithm presented above has two major limitations. First, the cost function v must be additive along paths in order for the algorithm to apply. Unfortunately, this is not true in the current formulation of the problem where we take into account the number of sampled points in order to normalize each segment's evaluation and its contribution to the overall evaluation of the pattern object. Second, the algorithm cannot be extended to deal with constraints among different segments without sacrificing its optimality.

The above two limitations and the higher time complexity of the optimal algorithm render it inappropriate for applications where some strict time limitations apply. Yet, the algorithm is of theoretical importance for it proves that the segmentation problem has polynomial time complexity when the cost function is additive and there are no constraints among different segments.

5.5 Implementation

tFPR has been implemented as a software library that provides users with functions that create and evaluate pattern objects. In addition, it has been embedded in the BB1 blackboard architecture [Hayes-Roth, 1985], where it acts as a front-end signal interpretation module. For our own purposes, we have implemented the tFPR pattern manager as a *knowledge source* in BB1. However, the organization of pattern manager is architecturally independent. We describe it here in very general terms. It can be reimplemented easily in other software architectures. The pattern manager is triggered by satisfaction of its *trigger-condition* which is the addition of a new interval of data to a temporal database, which in BB1 is called the *timeline.* Triggering binds the pattern manager's local variables to particular attributes of the triggering data. These values indicate which measure was updated, and the time at which it occurred. Next, pattern objects from the blackboard database that are related to the triggering data are retrieved using the *context mechanism* of BB1 knowledge sources. Finally, the pattern manager creates a unique instance of its "action," which reflects the established bindings of its local variables and a unique binding for its context variables that correspond to pattern objects. Thus, each new interval of data added to the timeline can trigger one or more instances of the pattern manager's action, each one prepared to evaluate a particular relevant pattern object in a right-to-left fashion, starting with the new data interval. We refer to each such instance as an executable pattern matching action.

Any executable pattern matching action may or may not actually be executed. In BB1, this is determined by the larger application system's current *strategy.* A strategy might differentially weight pattern recognition versus other reasoning tasks. And it might differentially weight pattern recognition activities for different data measures or different pattern objects. For example, the *Guardian* application for monitoring medical patients dynamically adopts context-dependent strategies that differentially weight pattern matching actions versus actions involved in diagnosis, prediction, planning, and explanation. And it dynamically adjusts its pattern matching activities to favor the most important patient data and the most important pattern objects in the current context. Any non-trivial application system (in terms of number of observed data measures, rate of adding new data intervals to the timeline, and number of pattern objects in the knowledge base) will need to control its pattern recognition activities with some such mechanism for selective attention.

Executing a pattern matching action involves calling an evaluation function with the instantiated variable bindings. Then, this function enters a cycle in which iteratively collects timeline data for the related pattern measures and updates its current pattern evaluation based on the values of the collected data. Termination occurs whenever the process reaches a state where no significant changes can occur to the final outcome either by perfect match or mismatch of the remaining data and segments. In the abnormal case where there would be missing data values at some point in time, the process is terminated and its current evaluation is returned. Eventually, the returned evaluation would appear as a new *episode* on the timeline.

tFPR contains many operators that are used to represent and evaluate n_p functions. Those operators are described in detail in [Drakopoulos and Hayes-Roth, 1994, Drakopoulos, 1994c]. A summary of them appears in table 5.1. The syntactic definition of n_p functions appears in table 5.2. In that table, n_p-list denotes a non-null list of n_ps and w_p-list, b_p-list denote lists of real numbers used as weights or bounds, respectively. Square brackets ([ ]) are used to denote optional expressions. The pair (a c) denotes the sigmoidal function $S_{a,c}$. The *number* argument in *rate* is used to denote the degree of the derivative. Finally, C_p-list is a list of constraints each of which has the following format :

```
(op arg arg weight)
```

where

```
op   ∈  {   S-GT, S-GE, S-LT, S-LE, S-EQ, S-NE  }
arg  ∈  {   (S-DT segment),  (S-MAX segment measure),
            number,          (S-MIN segment measure)   }
```

where `S-GT` stands for greater than, `S-GE` for greater equal and so on. Also `(S-DT segment)` is used to refer to the duration of segment `segment` while `(S-MAX segment measure)` is used to refer to the maximum value of `measure` in segment `segment`.

The difference between an operator θ (say `or`) and $m\theta$ (e.g. `mor`) is that the former uses all of its measures to all of its arguments while the later distributes its measures across its arguments. To illustrate the above, let E be the function that would be used to evaluate an n_p function on some input data $x = [x_1, \ldots, x_k]$. Then $E(n_p, x)$ should return the membership grade of x in the fuzzy set associated with n_p. Now, E can be defined recursively as follows:

$$E(nil, x) \quad = \quad x$$

$$
\begin{aligned}
f_{a,c}(x) &= S(ax \Leftrightarrow c) \\
f_{AND}(z_1, \ldots, z_k) &= \min(z_1, \ldots, z_k) \\
f_{OR}(z_1, \ldots, z_k) &= \max(z_1, \ldots, z_k) \\
f_{NOT}(x) &= \Leftrightarrow x \\
f_{NOT}(y) &= 1 \Leftrightarrow y \\
f_{ONE}(x) &= 1 \\
f_{ZERO}(x) &= 0 \\
f_{WS}(y_1, \ldots, y_k, w_1, \ldots, w_k) &= \frac{\sum_{j=1}^{k} y_k w_k}{\sum_{j=1}^{k} w_j} \\
f_{AVG}(y_1, \ldots, y_k) &= \frac{1}{k}\sum_{j=1}^{k} y_j \\
f_{GRP}(z_1, \ldots, z_k) &= [z_1, \ldots, z_k] \\
f_{BND}(x,\ b_1, \ldots, b_{k-1},\ y_1, \ldots, y_k) &= \left\{ \begin{array}{ll} y_1 & \text{if } x < b_1 \\ y_2 & \text{if } b_1 \le x < b_2 \\ & \vdots \\ y_{k-1} & \text{if } b_{k-2} \le x < b_{k-1} \\ y_k & \text{if } b_{k-1} \le x \end{array} \right\} \\
f_{VALUE}(z_1(t), \ldots, z_k(t)) &= [z_1(t), \ldots, z_k(t)] \\
f_{RATE}(z(t)) &= \frac{z(t+dt) \Leftrightarrow z(t)}{dt} \\
f_{RATE,1}(z) &= f_{rate}(z) \\
f_{RATE,i}(z) &= f_{rate,i-1}(f_{rate}(z)) \\
f_{SGM}(n_p, t) &= \text{RL-algorithm}(n_p, t) \\
f_{MSNN}(n_p, t) &= \text{evaluate-MSNN}(n_p, t)
\end{aligned}
$$

Table 5.1: Operator evaluation functions in tFPR. x is used to denote x-domain values, y is used to denote y-domain values and, z is used to denote values in either domain. [ ] are used to denote lists.

```
$n_p$ :  (a c)
    | (and $n_p$-list)
    | (mand $n_p$-list)
    | (or $n_p$-list)
    | (mor $n_p$-list)
    | (not $n_p$)
    | (one )
    | (zero )
    | (ws $n_p$-list $w_p$-list)
    | (mws $n_p$-list $w_p$-list)
    | (avg $n_p$-list)
    | (mavg $n_p$-list)
    | (grp $n_p$-list)
    | (mgrp $n_p$-list)
    | (bnd $n_p$-list $b_p$-list)
    | (mbnd $n_p$-list $b_p$-list)
    | (value $n_p$)
    | (rate [number] $n_p$)
    | (sgm $n_p$-list $w_p$-list $c_p$-list)
    | (msnn ms-unit-list)
    ;
```

Table 5.2: Syntactic definition of n_p functions in tFPR.

$$\begin{aligned} E((\theta, [n_{p1}, \ldots, n_{pk}], \alpha), x) &= f_\theta(E(n_{p1}, x), \ldots, E(n_{pk}, x), \alpha) \\ E((m\theta, [n_{p1}, \ldots, n_{pk}], \alpha), [x_1, \ldots, x_k]) &= f_\theta(E(n_{p1}, x_1), \ldots, E(n_{pk}, x_k), \alpha) \end{aligned}$$

where

θ	denotes a tFPR operator
$[n_{p1}, \ldots, n_{pk}]$	is a list of n_p functions
nil	denotes the empty list
α	is a list of arguments used by the function f_θ.

Special attention should be given to operators SGM and $MSNN$. The first is used to evaluate time-dependent pattern objects and involves an invocation of the RL-algorithm. The second is used to evaluate pattern objects that are defined by multi-sigmoidal neural networks. Note that tFPR would only evaluate MSNNs without training or re-training them. It assumes that all MSNNs in its pattern base have already been trained to compute the membership functions of their pattern objects. Figure 5.6 shows a block diagram describing the architecture of tFPR.

Training of MSNNs occurs during their definition process by the MSNN component of tFPR. At that time, the user has to provide the system with training data and network specification parameters (such as number of layers, number of units, connectivity, learning rates, window width etc.) that would be used to create and train a multi-sigmoidal neural network which, in turn, would be the n_p function of the corresponding pattern object. Thus, pattern objects which are defined in terms of MSNNs have to be trained during their definition process. On the other hand, all other pattern objects would be explicitly defined and need not be trained. Those pattern objects are expected to be supplied to the system by a user or an expert through the tFPR user interface (see figure 5.6). In either case, all pattern objects stored in the pattern base remain static during evaluation. The only way to modify them is to redefine them either through the MSNN component or through the tFPR user interface.

As described above, tFPR has two pattern object definition modes. The first uses the MSNN component and learning. The second requires an expert to provide the system with a pattern object definition. The first mode allows for difficult or imprecise pattern classes which cannot be described by a simple expression to be represented into the system by MSNNs which are trained to learn the appropriate membership functions. On the other hand, the second mode allows for easy pattern classes that can be described by a short and concise expression to be encoded directly into the system by an expert using the tFPR

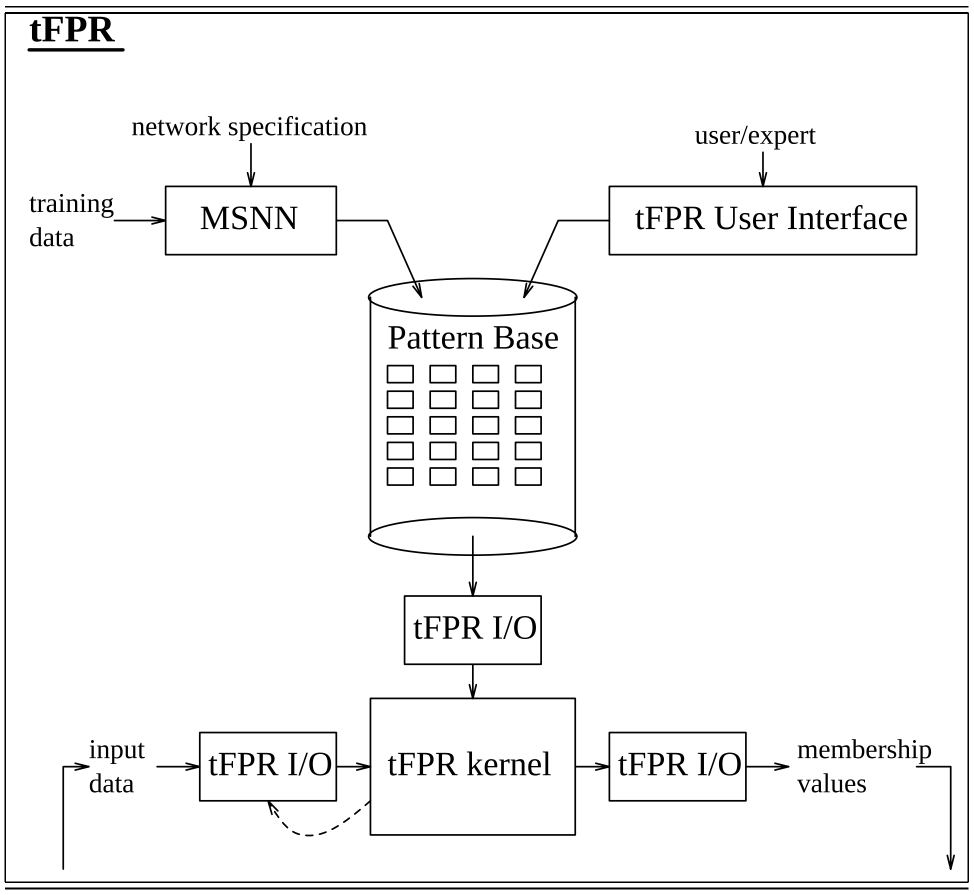


Figure 5.6: Architecture of tFPR

user interface. Thus, tFPR would be able to learn a pattern object definition whenever an expert is not available or cannot provide a precise definition.

It is certain that tFPR is a hybrid system for it uses fuzzy, neural, and structural pattern recognition methods. However, there is an additional reason for calling tFPR hybrid. tFPR is an extensional programming device in addition to being an intensional one whose programmability is dependent on fuzzy sets. We follow here the definitions in [Cottrell *et al.*, 1987] and call extensional programming the programming of complex relationships into a computing machine by showing it examples or instances of the relationships under consideration. In contrast, we call intensional programming the traditional programming where we directly write rules or specific algorithms into a computing machine without reference to any particular examples. As it was indicated above, there are good reasons for having both of those modes of programming in a system like tFPR. In general, we would like tFPR to be extensional so that the knowledge acquisition bottleneck ([Feigenbaum, 1977]), which in tFPR is instantiated as the definition of the pattern objects, can be avoided or reduced. On the other hand, we would like it to be intensionally programmable so that we can easily program into it an amount of (medical) knowledge that is readily available (in textbooks or elsewhere).

5.6 Illustrative Application

We are applying tFPR in a system called Guardian for monitoring intensive care patients [Hayes-Roth *et al.*, 1992]. Guardian performs several real-time reasoning tasks, including diagnosis, prediction, planning, and explanation of the patient's dynamic condition. All of these reasoning activities are based on the recognition of instances of important pattern classes in observed patient data.

Before we present some examples of tFPR pattern objects, let us define some auxiliary functions that could be used to create sigmoidal functions. Assuming an affine sigmoidal class A_S, where S is a basic unit sigmoidal function, define the following functions:

produce_sigmoidal(x_1, y_1, x_2, y_2). It creates the unique sigmoidal function in A_S that passes through the points (x_1, y_1) and (x_2, y_2).

produce_le(a, da) = produce_sigmoidal$(a,\ 1 \Leftrightarrow \epsilon,\ a + da,\ \epsilon)$. It creates a sigmoidal function that is nearly equal to 1 in the interval $(\Leftrightarrow\infty, a]$.

produce_ge(a, da) = produce_sigmoidal$(a \Leftrightarrow da,\ \epsilon,\ a,\ 1 \Leftrightarrow \epsilon)$ It creates a sigmoidal function that is nearly equal to 1 in the interval $[a, +\infty)$.

produce_gt(a, da) = produce_sigmoidal$(a,\ \epsilon,\ a + da,\ 1 \Leftrightarrow \epsilon)$ It creates a sigmoidal function that is nearly equal to 1 in the interval $(a, +\infty)$.

produce_ccint(a, b, da, db) = min(produce_ge(a, da), produce_le(b, db)) It creates a sigmoidal function that is nearly equal to 1 in the interval $[a, b]$.

produce_eq(a, da) = produce_ccint(a, a, da, da). It creates a sigmoidal function that is nearly equal to 1 only at point a.

ϵ, da, db are assumed to be small positive constants.

Table 5.3 illustrates the tFPR pattern objects for three conditions of interest to Guardian.

Hypoxia, on top of table 5.3, exhibits a pattern class that is described as a set of constraints upon its measures as well as a constraint between two of its measures. The latter is a linear constraint and is eliminated in the v_p function which replaces its third argument which is FiO_2 by $PaO_2 \Leftrightarrow 5 * FiO_2$. Finally the n_p function requires this new value to be less or equal to 0. Post bypass myocardial depression with adequate volume, in table 5.3, is a pattern class specification based on many measures and constraints which have been assigned relative importance with respect to the pattern evaluation. This example illustrates the difficulty in using only set operations to describe patterns depending on many conditions/measures. In this pattern, we used a weighted sum to express the vague interelationship of pattern measures regarding the evaluation of the pattern. Trying to put this pattern into a form of set operations (probably by using some interval conditions) not only would result in a much larger and more complicated definition but would also require an immensely greater effort for the pattern to be described at the first place. Finally, sudden increases in chest tube output are recognized by the pattern object at the bottom of table 5.3. That pattern object is a time-dependent pattern object consisting of two segments. The first segment requires the rate of chest tube output to be zero (e.g. chest tube output to be constant) while the second segment requires the rate of increment of chest tube output to be greater than 10/3600. In addition, the pattern objects imposes a constraint on the first segment requiring its duration to be less than 200 seconds. The relative importance of the two segments and the constraint is $1 : 3 : 1$.

Pattern class specification	**tFPR pattern object**
HYPOXIA: whenever $PaO_2 \leq 60\ mmHg$ or $O_2SAT \leq 90\%$ $(da = 7)$ or $PaO_2 \leq 5 * FiO_2$	($M_p = [PaO_2, O_2SAT, FiO_2]$, $A_p =$ identity function, $v_p(p, o, f) = [p,\ o,\ p \Leftrightarrow 5 * f]$, $n_p = [mOR,$ produce_le(60), produce_le(90, 7), produce_le(0)] $a_p = 3$, $d_p = 3$, $C_p = True$)
POST BYPASS MYOCARDIAL DEPRESSION WITH ADQ. VOLUME: whenever $CO \leq 3.4l/min$ $(da = 0.1)$ and $SVR > 1600$ $(da = 400)$ and $PAD > 18mmHg$ $(da = 2)$ and $CVP > 12mmHg$ $(da = 2)$ and $PCWP > 15mmHg$ $(da = 2)$ where the relative importance of the above measurements is 2:1:1:1:1.	($M_p = [CO, SVR, PAD,$ $CVP, PCWP]$, $A_p =$ identity function, $v_p =$ identity function, $n_p =$ $[mWS,$ produce_le(34, 0.1), 2, produce_gt(1600, 400), 1, produce_gt(18, 2), 1, produce_gt(12, 2), 1, produce_gt(15, 2), 1] $a_p = 5$, $d_p = 5$, $C_p = True$)
SUDDEN INCREASE OF CHEST TUBE OUTPUT: whenever a short period of constant chest tube output is followed by a sudden increase in chest tube output	($M_p =$ [PULMONARY.CT-output] $A_p =$ identity function, $v_p =$ identity function, $n_p =$ $[SGM,$ produce_RATE(produce_eq(0, 0.1)), 1, produce_RATE(produce_gt(10/3600)), 3, ;;duration of flat area < 200 (S_LT (S_DT 1) 200 1)] $a_p = 1$, $d_p = 1$, $C_p = True$)

Table 5.3: Three pattern class definitions and their corresponding pattern objects in tFPR.

Guardian currently has a knowledge base of about 100 pattern objects. We expect it to grow to a larger database especially after we would develop user-friendly or automatic tools for pattern acquisition.

5.7 Specifications of a Graphical Interface

The knowledge acquisition bottleneck ([Feigenbaum, 1977]) is manifested in tFPR as the pattern object specification process. This may be a major obstacle in the development of a useful system based on tFPR when the number of pattern objects would grow large. Of course the learning component (i.e. the MSNN component) can help alleviate this problem. However, a graphical interface that would facilitate the pattern object specification process would be very useful. We present here the specifications of such an interface.

Our solution requires that the user provide the system with n points $(x_1, y_1), \ldots, (x_n, y_n)$ that specify a pattern class. Then the system tries to find a minimum combination in terms of sigmoidal functions that would interpolate the pattern at the n given points with a given tolerance.

Though it is very hard to find an optimal algorithm, an approximate but fast solution is the following :

```
1. Specify the tolerance ($\theta$) in the approximation.
2. Sort the $n$ given points so that $x_1 < x_2 < \ldots < x_n$.
3. For $i = 1, 2, \ldots, n \Leftrightarrow 1$
       let $S_i$ be the sigmoidal function that passes through
               the $i$-th and $(i+1)$-th point.
       let $L_{S_i} = i$, $R_{S_i} = i + 1$.
4. While two successive sigmoidal functions $S_{a',c'}$ and $S_{a'',c''}$ can be merged
         (or equivalently when $|S_{a,c}(x_j) \Leftrightarrow y_j| < \theta, \quad \forall j \in L_{S_{a',c'}}, \ldots, R_{S_{a'',c''}}$,
                 where $a = (a' + a'')/2, \quad c = (c' + c'')/2$ )
         merge them (i.e replace them by $S_{a,c}$ and
                         let $L_{S_{a,c}} = L_{S_{a',c'}}, \quad R_{S_{a,c}} = R_{S_{a'',c''}}$ )
5. Return the current collection of sigmoidal functions.
```

The algorithm above can run (if implemented carefully) in $O(n^2)$. Its average time depends on the distribution of the number of sigmoidal mergings; the larger the number of mergings the slower the algorithm. However, for a uniform distribution, the average time is $O(nlogn)$. However, the worst case occurs when all the sigmoidal functions merge into a single one. In that case, we need $\Omega(n^2)$ sigmoidal computations.

In the case of time-dependent pattern classes, the user has to define the ridge of the segments in pattern objects, i.e. the ideal variation of the input to match the pattern object segments perfectly. This corresponds to the locus of points where the membership function of the fuzzy set that describes the pattern class in question is maximal. Then the user can proceed by defining other set of points where the pattern object evaluation is constant i.e. he can use isopotential curves to define his pattern objects. For example, the user may define the set of points where the pattern object evaluates to 0.5. After having defined at least two such sets the system can use the original algorithm to interpolate sigmoidal functions across the sets (with a given tolerance of approximation). Of course, the numerical nature of our computations require the set of points to be selected so that they correspond to values that are not very close to each other and not too close to either 0 or 1.

Finally, the set of interpolating sigmoidal functions would be used to form an n_p function. The construction of the rest of the elements of the pattern object for the current pattern is straightforward.

5.8 Extensions

There are two important extensions or enhancements of tFPR that are worth mentioning The first extension or enhancement of the system results from a basic observation made in section 5.5. Any non-trivial application must control and limit its pattern recognition activities with a mechanism for selective attention. Although such a mechanism was outlined in that section and is partially implemented in the Guardian system [Hayes-Roth *et al.*, 1992] alternative solutions should be studied as well. One such solution could be to create *control patterns* i.e. patterns that control the applicability of other patterns based on observations on the input data. These patterns serve no purpose but to control the matching process in order to make it more efficient. Alternatively, one can allow the C_p conditions in pattern objects to be learnable and based on experience. These extensions should result in a more efficient pattern recognition system.

The second extension of tFPR regards missing data In its current state, tFPR does not deal at all with that problem. It aborts the current pattern (by returning the current partial evaluation) as soon as it finds that some related data are missing. A better solution would be to include in the pattern object information regarding the behavior of the pattern in cases where information is missing. For example, in order to deal with missing data, the pattern matching process may choose to abort, use default values to anticipate the lack of some information, use expected values, assign some predefined UNKNOWN value to anything that is missing, or use a combination of these policies depending on the definition of the pattern object that is under evaluation.

Currently, there are no control patterns in tFPR and pattern evaluations with missing data are discarded.

5.9 Conclusion

tFPR is a hybrid pattern recognition system that has been founded on a number of new theoretical results as well as a number of practical considerations. Because of an existing trade-off between probabilistic and possibilistic measures, tFPR represents pattern classes by possibilistic measures such as fuzzy sets. Because of the sigmoidal bubble theorem, tFPR uses sigmoidal functions to approximate arbitrary fuzzy set membership. In order to deal with time-dependent pattern classes, tFPR follows a segmentation approach and uses an efficient segmentation algorithm to compute and evaluate segmentations. Finally, in order to deal with pattern classes that cannot be described explicitly in any apparent way and to reduce the knowledge acquisition effort, tFPR employs a learning mechanism, namely multi-sigmoidal neural networks, that are able to learn complex pattern class definitions and encode complicated relationships when trained by examples.[1]

In general, tFPR can represent and recognize patterns that are static or dynamic depending on time or any other parameter-space. The direct evaluation of static patterns and the linear time complexity of its segmentation method provide the groundwork of a system that may be appropriate for real-time applications.

However, it must be noted that the emphasis on tFPR is not on run-time performance but rather on knowledge acquisition efficiency. Due to its hybrid nature, tFPR provides alternative and resourceful solutions to pattern class specification problems. In addition, it defines a modular pattern recognition architecture that can be effectively and easily

[1]Currently, there are no patterns in our database that are described through multi-sigmoidal neural networks. The physicians that developed our database found it adequate the use of simple fuzzy sets to describe their pattern classes of interest.

extended through its set of operators.

The overall goal of this research has been a modular system or architecture developed on solid mathematical foundations and equipped with a wide set of tools and methods for pattern recognition and specification. tFPR seems to have met fully that goal.

Bibliography

[Alsina and Trillas, 1983] C. Alsina and E. Trillas. On almost distributive Lukasiewicz triplets. *Fuzzy Sets and Systems*, 92:175–178, 1983.

[Alsina *et al.*, 1983] C. Alsina, E. Trillas, and L. Valverde. On some logical connectives for fuzzy set theory. *Journal of Mathematical Analysis and Applications*, 93(1):15–26, 1983.

[Ash, 1989] T. Ash. Dynamic node creation in backpropagation networks. *Connection Science*, 1(4):365–375, 1989.

[Bellman and Giertz, 1973] R. Bellman and M. Giertz. On the analytic formalization of the theory of fuzzy sets. *Information Sciences*, 5:149–156, 1973.

[Bellman *et al.*, 1966] R.E. Bellman, R. Kalaba, and L.A.Zadeh. Abstraction and pattern classification. *Journal of Mathematical Analysis and Applications*, 13:1–7, 1966.

[Bezdek and Pal, 1992] J.C. Bezdek and S.K. Pal. *Fuzzy Models for Pattern Recognition.* IEEE Press, New York, 1992.

[Bezdek, 1973] J.C. Bezdek. Cluster validity with fuzzy sets. *Journal of Cybernetics*, 3:58–73, 1973.

[Bezdek, 1981] J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Plenum Press, New York, 1981.

[Carpenter *et al.*, 1992] G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, and D.B. Rosen. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3(5):698–713, September, 1992.

[Charniak, 1991] E. Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63, 1991.

[Cheeseman, 1986] P. Cheeseman. Probabilistic versus fuzzy reasoning. In L.N. Kanal and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 85–102. North Holland, Amsterdam and New York., 1986.

[Cohen and Hudson, 1992] M.E. Cohen and D.L. Hudson. Integration of neural network techniques with approximate reasoning in knowledge based systems. In A. Kandel and G. Langholz, editors, *Hybrid Architectures for Intelligent Systems*, pages 72–85. CRC Press, Boca Raton, FL, 1992.

[Cooper, 1987] G.F. Cooper. Probabilistic inference using belief networks is NP-hard. Technical Report KSL–87–27, Medical Computer Science Group, Stanford University, 1987.

[Cottrell *et al.*, 1987] G.W. Cottrell, P. Munro, and D. Zipser. Learning internal representations from gray-scale images: An example of extensional programming. In *Proceeding of the Ninth Annual Conference of the Cognitive Science Society*, 1987.

[DasGupta and Schnitger, 1993] B. DasGupta and G. Schnitger. The power of approximating: A comparison of activation functions. In S.J. Hanson, J.D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 615–622. Morgan Kaufmann, San Mateo, CA, 1993.

[Dawson and Schopflocher, 1992] M.R.W. Dawson and D.P. Schopflocher. Modifying the generalized delta rule to train networks of non-monotonic processors for pattern classification. *Connection Science*, 4(1):19–31, 1992.

[Delgado and Moral, 1987] M. Delgado and S. Moral. On the concept of possibility-probability consistency. *Fuzzy Sets and Systems*, 21:311–318, 1987.

[Dempster, 1968] A.P. Dempster. A generalization of Bayesian inference. *Journal of the Royal Statistical Society, Series B*, 30:205–247, 1968.

[Drakopoulos and Hayes-Roth, 1994] J.A. Drakopoulos and B. Hayes-Roth. tFPR: A fuzzy and structural pattern recognition system of multi-variate time-dependent patterns based on sigmoidal functions. Technical Report KSL–94–42, Knowledge Systems laboratory, Stanford University, 1994. Submitted for publication to *International Journal of Fuzzy Sets and Systems*,.

[Drakopoulos, 1991] J.A. Drakopoulos. FPR: A fuzzy pattern recognizer based on sigmoidals. Technical Report KSL–91–75, Knowledge Systems laboratory, Stanford University, 1991.

[Drakopoulos, 1994a] J.A. Drakopoulos. Probabilities, possibilities, and fuzzy sets. Technical Report KSL–94–40, Knowledge Systems Laboratory, Stanford University, 1994. To appear in the *International Journal of Fuzzy Sets and Systems.*

[Drakopoulos, 1994b] J.A. Drakopoulos. Sigmoidal theory. Technical Report KSL–94–41, Knowledge Systems Laboratory, Stanford University, 1994. To appear in the *International Journal of Fuzzy Sets and Systems.*

[Drakopoulos, 1994c] J.A. Drakopoulos. tFPR user's manual. The manual is available from the author, 1994.

[Drakopoulos, 1995] J.A. Drakopoulos. Multi-sigmoidal neural networks and backpropagation. In To appear in *Fourth International Conference on Artificial Neural Networks*, 1995.

[Dubois and Prade, 1983] D. Dubois and H. Prade. Unfair coins and necessity measures: towards a possibilistic interpretation of histograms. *Fuzzy Sets and Systems*, 10:15–20, 1983.

[Dubois and Prade, 1987] D. Dubois and H. Prade. Fuzzy numbers: An overview. In J.C. Bezdek, editor, *Analysis of Fuzzy Information*, pages 3–39. CRC Press, Boca Raton, Fla., 1987.

[Dubois *et al.*, 1993] D. Dubois, H. Prade, and R. Yager. *Readings in Fuzzy Sets for Intelligent Systems.* Morgan Kaufman, San Mateo, CA, 1993.

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

[Dunn, 1973] J.C. Dunn. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3:32–57, 1973.

[Fahlman and Lebiere, 1990] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532, San Mateo, CA, 1990. Morgan Kaufmann.

[Fahlman, 1988] S.E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In D.S. Touretzky, G.E. Hinton, and T.J. Sejnowski, editors, *Proceedings, 1988 Connectionist Models Summer School*, pages 38–51, Los Altos, CA, 1988. Morgan Kaufmann.

[Fahlman, 1991] S.E. Fahlman. The recurrent cascade-correlation architecture. In R.P. Lippmann, J.E. Moody, and D.S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 190–196. Morgan Kaufmann, San Mateo, CA, 1991.

[Feigenbaum, 1977] E.A. Feigenbaum. The art of artificial intelligence: Themes and case studies in knowledge engineering. In *IJCAI 5*, pages 1014–1029, 1977.

[Fu, 1980] K.S. Fu. Recent developments in pattern recognition. *IEEE Transactions on Computers*, C-29(10):845–857, 1980.

[Fu, 1982] K.S. Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Fukunaga, 1972] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, New York, 1972.

[Genesereth and Nilsson, 1987] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kauffmann, Los Altos, CA, 1987.

[Gorman and Sejnowski, 1988] R.P. Gorman and T.J. Sejnowski. Analysis of hidden units in a layered network rained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

[Gupta *et al.*, 1975] M.M. Gupta, G.K.Knopf, and P.N. Nikiforuk. Sinusoidal-based cognitive mapping functions. In M.M. Gupta and T. Yamakawa, editors, *Fuzzy Logic in Knowledge-Based Systems, Decision and Control*, pages 69–89. Academic Press, New York, 1975.

[Hamacher, 1976] H. Hamacher. On the logical connectives of fuzzy statements and their affiliated truth function. In *Proceedings of the Third European Meeting on Cybernetics and Systems Research*, Vienna, Austria, 1976.

[Hanson and Pratt, 1989] S.J. Hanson and L.Y. Pratt. Some comparisons of constraints for minimum network construction with back-propagation. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 177–185. Morgan Kaufmann, San Mateo, CA, 1989.

[Hanson, 1990] S.J. Hanson. Meiosis networks. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 533–541, San Mateo, CA, 1990. Morgan Kaufmann.

[Hassibi and Stork, 1993] B. Hassibi and D.G. Stork. Second order derivatives for network prunning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5*, 1993.

[Hayes-Roth *et al.*, 1992] B. Hayes-Roth, R. Washington, D. Ash, A. Collinot, R. Hewett, A. Vina, and A. Seiver. Guardian: A prototype intelligent agent for intensive care monitoring. *Artificial Intelligence in Medicine*, 4:165–185, 1992.

[Hayes-Roth, 1985] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[Haykin, 1994] S. Haykin. *Neural Networks.* Macmillan, New York, 1994.

[Hebb, 1949] D. Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[Hecht-Nielsen, 1987] R. Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. In *Proceedings of the International Conference on Neural Networks, III*, pages 11–13, New York, 1987. IEEE Press.

[Hecht-Nielsen, 1990] R. Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, Reading, MA, 1990.

[Henkind and Harrison, 1988] S.J. Henkind and M.C. Harrison. Analysis of four uncertainty calculi. *IEEE Trans. on Man Systems and Cybernetics*, 18(5):700–714, 1988.

[Hertz *et al.*, 1991] J.A. Hertz, R.G. Palmer, and A.S. Krogh. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, CA, 1991.

[Hinton, 1989] G.E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

[Hornik *et al.*, 1989] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[Hornik, 1993] K. Hornik. Some new results on neural network approximation. *Neural Networks*, 6:1069–1072, 1993.

[Jain, 1987] A.K Jain. Advances in statistical pattern recognition. In P.A. Devijver and J. Kittler, editors, *Pattern Recognition Theory and Applications*, pages 1–19. Springer-Verlag, Berlin; New York, 1987.

[Kandel, 1982] A. Kandel. *Fuzzy Techniques in Pattern Recognition.* Wiley, New York, 1982.

[Kaufmann, 1975] A. Kaufmann. *Introduction to the Theory of Fuzzy Subsets.* Academic Press, New York, 1975.

[Keller *et al.*, 1992] J.M. Keller, R.R. Yager, and H. Tahani. Neural network implementation of fuzzy logic. *Fuzzy Sets and Systems*, 45:1–12, 1992.

[Klir and Parviz, 1992] G. Klir and B. Parviz. Probability-possibility transformations: A comparison. *Int. Journal of General Systems*, 21(1):291–310, 1992.

[Klir, 1989] G. Klir. Is there more to uncertainty than some probability theorists would have us believe? *Int. Journal of General Systems*, 15(4):347–378, 1989.

[Kolmogorov, 1950] A.N. Kolmogorov. *Foundations of the Theory of Probability.* Chelsea, New York, 1950.

[Kolmogorov, 1957] A.N. Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doclady Academy Nauk SSSR*, 114:953–956, 1957.

[Kosko, 1990] B. Kosko. Fuzziness vs. probability. *Int. Journal of General Systems*, 17(2–3):211–240, 1990.

[Lapedes and Farber, 1987] A.S. Lapedes and R.M. Farber. Nonlinear signal processing using neural networks: prediction and system modeling. Technical Report LA-UR–87–2662, Los Alamos National Laboratory, 1987.

[LeCun *et al.*, 1990] Y. LeCun, J.S. Denker, and S.A. Solla. Optimal brain damage. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605, San Mateo, CA, 1990. Morgan Kaufmann.

[Leshno *et al.*, 1993] M. Leshno, V.Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6:861–867, 1993.

[Lindley, 1987] D.V. Lindley. The probability approach to the treatment of uncertainty in artificial intelligence and expert systems. *Statistical Science*, 2(1):17–24, 1987.

[Lippmann, 1989] R.P. Lippmann. Pattern classification using neural networks. *IEEE Communications Magazine*, November, 27(11):47–64, 1989.

[Lukasiewicz, 1970] J. Lukasiewicz. Logical foundations of probability theory. In L. Berkowski, editor, *Jan Lukasiewicz, Selected Works*, pages 16–43. North-Holland, Amsterdam, 1970.

[McCulloch and Pitts, 1943] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[Minsky and Papert, 1988] M.L. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, expanded edition, 1988.

[Moody and Darken, 1989] J. Moody and C.J. Darken. Fast learning in networks of locally tuned processing units. *Neural Computation*, 1:281–294, 1989.

[Moody and Yarvin, 1992] J. Moody and N. Yarvin. Networks with learned unit response functions. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 1048–1055. Morgan Kaufmann, San Mateo, CA, 1992.

[Mooney *et al.*, 1990] R. Mooney, J. Shavlik, G. Towell, and A. Gove. An experimental comparison of symbolic and connectionist learning algorithms. In J. Shavlik and T. G. Dietterichs, editors, *Readings in Machine Learning*, pages 171–176. Morgan Kaufmann, San Mateo, CA, 1990.

[Mozer and Smolensky, 1989] M.C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 107–115, San Mateo, CA, 1989. Morgan Kaufmann.

[Neapolitan, 1990] E. Neapolitan. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. Wiley, New York, 1990.

[Nilsson, 1965] N.J. Nilsson. *Learning Machines*. McGraw-Hill, New York, 1965.

[Nilsson, 1986] N.J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986.

[Pal and Dutta-Mazamder, 1986] S.K. Pal and D. Dutta-Mazamder. *Fuzzy Mathematical Approach in Pattern Recognition Problems*. Wiley, New York, 1986.

[Pal and Majumder, 1986] S.K. Pal and D.K. Dutta Majumder. *Fuzzy mathematical approach in pattern recognition problems*. Wiley, New York, 1986.

[Pal and Mitra, 1992] S.K. Pal and S. Mitra. Multilayer perceptron, fuzzy sets, and classification. *IEEE Transactions on Neural Networks*, 3(5):683–697, September, 1992.

[Papadimitriou and Steiglitz, 1982] C.H. Papadimitriou and K.S. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Papoulis, 1991] A. Papoulis. *Probability, random variables, and stochastic processes*. McGraw-Hill, New York, 1991.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kauffman, Palo Alto, 1988.

[Pearl, 1990] J. Pearl. Reasoning under uncertainty. *Annual Review of Computer Science*, 4:37–72, 1990.

[Peterson and Silberschatz, 1985] J.L. Peterson and A. Silberschatz. *Operating System Concepts*. Addison-Wesley, second edition, 1985.

[Poggio and Girosi, 1989] T. Poggio and F. Girosi. A theory of networks for approximation and learning. Technical Report No. 1140, MIT AI Memo Lab, 1989.

[Rosenblatt, 1958] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[Rosenblatt, 1961] F. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, Washington, DC, 1961.

[Rumelhart *et al.*, 1986a] D.E. Rumelhart, G.E. Hinton, and R.J.Williams. Learning internal representations by error propagation. In D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, I: Foundations*, pages 318–363. MIT Press, Cambridge, MA, 1986.

[Rumelhart *et al.*, 1986b] D.E. Rumelhart, J.L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, I: Foundations.* MIT Press, Cambridge, MA, 1986.

[Ruspini, 1969] E. Ruspini. A new approach to clustering. *Information Control*, 15:22–52, 1969.

[Ruspini, 1970] E. Ruspini. Numerical methods for fuzzy clustering. *Information Sciences*, 2:319–350, 1970.

[Salzberg, 1990] S.L. Salzberg. *Learning with Nested Generalized Exemplars.* Kluwer Academic, Hingham, MA, 1990.

[Schalkoff, 1992] R.J. Schalkoff. *Pattern Recognition: Statistical, Structural, and Neural Approaches.* Wiley, New York, 1992.

[Schiffman *et al.*, 1993] W. Schiffman, M. Joost, and R. Werner. Comparizon of optimized backpropagation algorithms. In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 97–104, Brussels, 1993.

[Sejnowski and Rosenberg, 1987] T.J. Sejnowski and C.R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.

[Shafer, 1976] G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, Princeton, NJ, 1976.

[Shortliffe, 1976] E.H. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* Elsevier, New York, 1976.

[Simpson, 1992] P.K. Simpson. Fuzzy min-max neural networks–part1: Classification. *IEEE Transactions on Neural Networks*, 3(5):776–786, September, 1992.

[Sprecher, 1965] D.A. Sprecher. On the structure of continuous functions of several variables. *Transactions American Mathematical Society*, 115(3):340–355, 1965.

[Stoll, 1963] R.R. Stoll. *Set Theory and Logic.* W.H. Freeman and Company, San Fransisco and London, 1963.

[Takagi and Hayashi, 1991] H. Takagi and I. Hayashi. Artificial neural network driven fuzzy reasoning. *International Journal of Approximate Reasoning*, 5:191–212, 1991.

[Takagi, 1990] H. Takagi. Fusion technology of fuzzy theory and neural network: Survey and future directions. In *Proceedings of the International Conference on Fuzzy Logic and Neural Networks*, pages 13–26, Iizuka, 1990.

[Thodberg, 1991] H.H. Thodberg. Improving generalization of neural networks through prunning. *International Journal of Neural Systems*, 1(4):317–326, 1991.

[Uhr, 1973] L. Uhr. *Pattern Recognition, Learning, and Thought.* Prentice-Hall, Englewood Cliffs, NJ, 1973.

[Utgoff, 1988] P.E. Utgoff. Perceptron trees: a case study in hybrid concept representation. In *AAAI-88: Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 601–606, San Mateo, CA, 1988. Morgan-Kaufmann.

[Weigend *et al.*, 1990] S.A. Weigend, B.A. Hubermann, and D.E. Rumelhart. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1:193–209, 1990.

[Widrow, 1962] B. Widrow. Generalization and information storage in networks of adaline neurons. In G.T. Yovitts, editor, *Self-Organizing Systems.* Spartan Books, Washington, DC, 1962.

[Wynne-Jones, 1992] M. Wynne-Jones. Node splitting: A constructive algorithm for feed-forward neural networks. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 1072–1079. Morgan Kaufmann, San Mateo, CA, 1992.

[Yager *et al.*, 1987] R.R. Yager, S. Ovchinnikov, R.M. Tong, and H.T.Nguyen, editors. *Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh.* Wiley, New York, 1987.

[Yager, 1980] R.R. Yager. Generalized "and/or" operators for multivalued and fuzzy logic. In *Proceedings of the Tenth Symposium of Multiple-Valued Logic*, pages 214–218, Evaston, Illinois, 1980.

[Yamakawa and Furukawa, 1992] T. Yamakawa and M. Furukawa. A design algorithm of membership functions for a fuzzy neuron using example-based learning. In *Proceedings of IEEE 1st International Conference on Fuzzy Systems*, pages 75–82, San Diego, CA, 1992.

[Zadeh, 1965] L.A. Zadeh. Fuzzy sets. *Inf. Control*, 8(338), 1965.