Design Optimization of Fuzzy Logic Systems

Paolo Dadone

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Electrical Engineering

May 18, 2001
Blacksburg, Virginia

Keywords: Fuzzy logic systems, Supervised learning, Optimization,
Non-differentiable optimization

Design Optimization of Fuzzy Logic Systems

Paolo Dadone

(ABSTRACT)

Fuzzy logic systems are widely used for control, system identification, and pattern recognition problems. In order to maximize their performance, it is often necessary to undertake a design optimization process in which the adjustable parameters defining a particular fuzzy system are tuned to maximize a given performance criterion. Some data to approximate are commonly available and yield what is called the supervised learning problem. In this problem we typically wish to minimize the sum of the squares of errors in approximating the data.

We first introduce fuzzy logic systems and the supervised learning problem that, in effect, is a nonlinear optimization problem that at times can be non-differentiable. We review the existing approaches and discuss their weaknesses and the issues involved. We then focus on one of these problems, i.e., non-differentiability of the objective function, and show how current approaches that do not account for non-differentiability can diverge. Moreover, we also show that non-differentiability may also have an adverse practical impact on algorithmic performances.

We reformulate both the supervised learning problem and piecewise linear membership functions in order to obtain a polynomial or factorable optimization problem. We propose the application of a global nonconvex optimization approach, namely, a reformulation and linearization technique. The expanded problem dimensionality does not make this approach feasible at this time, even though this reformulation along with the proposed technique still bears a theoretical interest. Moreover, some future research directions are identified.

We propose a novel approach to step-size selection in batch training. This approach uses a limited memory quadratic fit on past convergence data. Thus, it is similar to response surface methodologies, but it differs from them in the type of data that are used to fit the model, that is, already available data from the history of the algorithm are used instead of data obtained according to an experimental design. The step-size along the update direction (e.g., negative gradient or deflected negative gradient) is chosen according to a criterion of minimum distance from the vertex of the quadratic model. This approach rescales the complexity in the step-size selection from the order of the (large) number of training data, as in the case of exact line searches, to the order of the number of parameters (generally lower than the number of training data). The quadratic fit approach and a reduced variant are tested on some function approximation examples yielding distributions of the final mean square errors that are improved (i.e., skewed toward lower errors) with respect to the ones in the commonly used pattern-by-pattern approach. Moreover, the quadratic fit is also competitive and sometimes better than the batch training with optimal step-sizes, thus showing an improved performance of this approach. The quadratic fit approach is also tested in conjunction with gradient deflection strategies and memoryless variable metric methods, showing errors smaller by 1 to 7 orders of magnitude. Moreover, the convergence speed by using either the negative gradient direction or a deflected direction is higher than that of the pattern-by-pattern approach, although the computational cost of the algorithm per iteration is moderately higher than the one of the pattern-by-pattern method. Finally, some directions for future research are identified.

This research was partially supported by the Office of Naval Research (ONR), under MURI Grant N00014-96-1-1123.

To my family:

Iris,
Antonella and Andrea,
Claudia and Fabrizia.

Acknowledgments

The biggest thanks are due to my advisor Prof. Hugh VanLandingham, he always supported my scientific endeavors and was, and will be, the source of inspiration for both my research and my life. Coming to Virginia Tech was accompanied by mixed emotions of hype for the challenge and the new experience, as well as fear of the unknown. I consider myself extremely lucky in having found Prof. VanLandingham as my advisor, not only he has been an excellent scientific advisor, but he is also a great man. His continuous faith in my work and his valuable and “out-of-the-box” perspectives were very useful in the course of these studies. I wish him all the best in his years as an Emeritus professor.

I am also greatly indebted to my committee for being nice and available to me and for carefully reviewing my research. In particular, I would like to thank Prof. William Baumann for his continuous support and for the interesting discussions. I would also like to thank Prof. Hanif Sherali for introducing me to the wonderful world of optimization with the preciseness and clarity so typical of him. Finally, I would also like to thank Prof. Subhash Sarin and Prof. Dusan Teodorovic for their extreme kindness and for helping me throughout all the phases of the doctoral work.

These years as a graduate student were also interesting in experiencing a multidisciplinary research environment and being exposed to several researchers and ideas. I have to thank the MURI project and the ONR for this, specifically Prof. Ali Nayfeh and Dr. Kam Ng for the support of my studies as well as for the organization of the MURI.

Part of my capabilities and thinking process are due to my excellent undergraduate education at the Politecnico di Bari. I would like to thank all my teachers, and especially Prof. Michele Brucoli, Prof. Luisi, and Prof. Bruno Maione.

On a more personal basis, these years as a graduate student carry more than just my scientific progress. Indeed, they helped me in meeting some wonderful people. One of those people, that alone makes this journey worth it, is the lovely Iris Stadelmann. Iris' love, support, calm, and organization taught me a lot and made me, and is still making me, a better person.

I would also like to thank my parents Antonella and Andrea and my sisters Claudia and Fabrizia. Their continuing support and love, and knowing that they are always there for me in any situation, made this experience, and makes my life, a lot easier. Moreover, my parents taught me the morals, the thinking process, and the love for an intellectual challenge that I have now, for this I am also greatly indebted to them.

I would also like to thank my good friend Christos Kontogeorgakis for the help and advise he gave me, as well as for the good times spent together. Thanks also to Lee Williams and Craig Pendleton for being there and for the nice partying, to Aysen Tulpar and Emre Isin for the companionship and help in important moments. Thanks also to my Italian friends, faraway so close, Pierluigi, Marco, Sabrina, Annamaria, Sergio, Valentina, Paolo, and Diego.

On a final note, I would like to thank all the people in the lab for the interesting discussions; moreover, their presence and companionship lit some dark and difficult moments in the course of my research. In particular I would like to thank Farooq, Joel, Xin-Ming, and Marcos.

Table of Contents

List of Figures

List of Tables

Glossary

$A_{jk(j,l)}$: fuzzy set for the j-th input in the l-th rule

α-cut: $A_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha\}$

$B^{h(l)}$: fuzzy set consequent of the l-th rule

Complement: $\overline{A}$: $\mu_{\overline{A}}(x) = 1 - \mu_A(x)$

Containment: $A \subseteq B \Leftrightarrow \forall x \in X : \mu_A(x) \leq \mu_B(x)$

Convex fuzzy set: $\forall x_1, x_2 \in X \wedge \forall \lambda \in [0,1]$, $\mu_A[\lambda x_1 + (1-\lambda)x_2] \geq \min\{\mu_A(x_1), \mu_A(x_2)\}$

Core: $\text{core}(A) = \{x \in X \mid \mu_A(x) = 1\}$

Crossover points: Points at which $\mu_A(x) = 0.5$

δ_i: i-th consequent constant (or constants for local models) or parameters in general

$\dfrac{\partial E}{\partial \boldsymbol{w}}$: gradient of the mean square error with respect to the adjustable parameters

$E(\boldsymbol{w})$: mean square error (MSE), $E(\boldsymbol{w}) = \dfrac{1}{2N}\sum_{i=1}^{N}[y(\boldsymbol{x}_i, \boldsymbol{w}) - y_{di}]^2$, error in approximating the training data

$E_i(\boldsymbol{w})$: instantaneous error, $E_i(\boldsymbol{w}) = \dfrac{1}{2}[y(\boldsymbol{x}_i, \boldsymbol{w}) - y_{di}]^2$, error in approximating the i-th datum

Epoch: entire presentation of the data set

FLS: Fuzzy logic system

Fuzzy set: Set of ordered pairs $A = \{(x, \mu_A(x)) \mid x \in X\}$, where, $\mu_A(x)$ is the membership function

Fuzzy singleton: Fuzzy set whose support is a single point with $\mu_A(x) = 1$

$h(l)$: function defining the fuzzy consequent for the l-th rule; $h : \{1,2,\ldots, R\} \rightarrow \{1,2,\ldots, H\}$

H: Number of consequents

η*:* learning rate, also step-size

K_j*:* Number of fuzzy sets (i.e., partitions) on the j-th input *k*(*j*,*l*): function defining the fuzzy set on the *j*-th input that is an antecedent in the *l*-th rule; $k : \{1,2, \ldots,n\}\times\{1,2,\ldots, R\} \rightarrow \aleph$, $1 \leq k(j,l) \leq K_j$.

Intersection: $C = A \cap B$: $\mu_C(x) = \mu_A(x) \otimes \mu_B(x)$

LMQ: Limited memory quadratic fit

LMRQ: Limited emory reduced quadratic fit

Membership function: Mapping of each element of a universe of discourse to a membership grade between 0 and 1 (included)

MSE: mean square error, see $E(\boldsymbol{w})$

$\mu_{ij}(x)$: membership function for the fuzzy set A_{ij}

N: Number of training data

n: number of inputs

Normal fuzzy set: Fuzzy set with nonempty core

P: number of adjustable parameters

R: Number of rules

RLT: Reformulation-linearization technique

Support: $\text{support}(A) = \{x \in X \,|\, \mu_A(x) > 0\}$

Strong α*-cut*: $A'_\alpha = \{x \in X \mid \mu_A(x) > \alpha\}$

Union: $C = A \cup B$: $\mu_C(x) = \mu_A(x) \oplus \mu_B(x)$

t-conorm ($\oplus$): Operator, $\oplus : [0,1] \times [0,1] \rightarrow [0,1]$ used for union of sets; generally max or algebraic sum

t-norm ($\otimes$): Operator, $\otimes : [0,1] \times [0,1] \rightarrow [0,1]$ used for intersection of sets; generally min or product

$\boldsymbol{w}$: adjustable parameters

$\boldsymbol{w_a}$: adjustable antecedent parameters

$\boldsymbol{w_c}$: adjustable consequent parameters

$\boldsymbol{x}_i$: *i*-th (input) training point

y_{di}: *i*-th desired output training point

$$\bigotimes_{i=1}^{n} \alpha_i = \alpha_1 \otimes \alpha_2 \otimes \ldots \otimes \alpha_n$$

Chapter 1
A Guided Tour of Fuzzy Logic Systems

"All natural languages are inherently ambiguous."
(Anonymous)

1.1 Introduction

Even though fuzzy sets were introduced in their modern form by Zadeh [88] in 1965, the idea of a multi-valued logic in order to deal with vagueness has been around from the beginning of the century. Peirce was one of the first thinkers to seriously consider vagueness, he did not believe in the separation between true and false and believed everything in life is a continuum. In 1905 he stated: "I have worked out the logic of vagueness with something like completeness" [61]. Other famous scientists and philosophers probed this topic further. Russell claimed, "All language is vague" and went further saying, "vagueness is a matter of degree" (e.g., a blurred photo is vaguer than a crisp one, etc.) [67]. Einstein said that "as far as the laws of mathematics refer to reality, they are not certain, and as far as they are certain, they do not refer to reality" [7]. Lukasiewicz took the first step towards a formal model of vagueness, introducing in 1920 a three-valued logic based on true, false, and *possible* [39]. In doing this he realized that the laws of the classical two-valued logic might be misleading because they address only a fragment of the domain. A year later Post outlined his own three-valued logic, and soon after many other multi-valued logics proliferated (Godel, von Neumann, Kleene, etc.) [42]. A few years later, in 1937 Black outlined his precursor of fuzzy sets [7]. He agreed with Peirce in terms of the continuum of vagueness and with Russell in terms of the degrees of vagueness. Therefore, he outlined a logic based on degrees of *usage*, based on the probability that a certain object will be considered belonging to a certain class [7]. Finally, in 1965 Zadeh [88] elaborated a multi-valued logic where degrees of *truth* (rather than *usage*) are possible.

Fuzzy set theory generalizes classical set theory in that the membership degree of an object to a set is not restricted to the integers 0 and 1, but may take on any value in [0,1]. By elaborating on the notion of fuzzy sets and fuzzy relations we can define fuzzy logic systems (FLS). FLSs are rule-based systems in which an input is first *fuzzified* (i.e., converted from a crisp number to a fuzzy set) and subsequently processed by an inference engine that retrieves knowledge in the form of fuzzy rules contained in a rule-base. The fuzzy sets computed by the fuzzy inference as the output of each rule are then composed and *defuzzified* (i.e., converted from a fuzzy set to a crisp number). A fuzzy logic system is a nonlinear mapping from the input to the output space.

This chapter serves to provide the necessary background to understand the developments of the next chapters; readers already familiar with FLSs can skip to Section 1.6 for the necessary problem assumptions, notation, and a general FLS model. The basic notion of fuzzy set will be introduced and discussed in the introductory section on fuzzy sets. Most of this material was edited from the brilliant introduction provided in Bezdek [6]. Fuzzy logic and fuzzy relations will be discussed as in Mendel [43], finally leading to fuzzy logic and fuzzy logic systems and their principles of operation. As the reader will notice there are many possible choices in the design of a FLS, we will discuss the most common choices and present the formulation of the corresponding nonlinear mapping implemented by a FLS.

1.2 Introduction to Fuzzy Sets

Fuzzy sets are generalized sets introduced by Professor Zadeh in 1965 as a mathematical way to represent and deal with vagueness in everyday life [88]. To paraphrase Zadeh [89], we have been developing a wealth of methods to deal with *mechanistic* systems (e.g., systems governed by differential equations) but there is a lack of methods for *humanistic* systems. Our natural tendency would be to still use the same good and proven methods we already developed, but they might not work as well for this other class of systems. Indeed, Zadeh informally states what he calls the principle of incompatibility:

"As the complexity of a system increases, our ability to make precise and yet significant statements about its behavior diminishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics."

Simply put, fuzzy sets are a clever way to deal with vagueness as we often do in our daily life [6]. For example, suppose you were advising a driving student on when to apply the brakes, would your advise be like: "Begin braking *74 feet* from the crosswalk," or would it be more like: "Apply the brakes *when approaching the crosswalk*"? Obviously the latter, since the former instruction is too precise to easily implement. Everyday language is the cornerstone example of vagueness and is representative of how we assimilate and act upon vague situations and instructions. It may be said that we all assimilate and use (act on) fuzzy data, vague rules, and imprecise information, just as we are able to make decisions about situations which seem to be governed by an element of chance. Accordingly, computational models of real systems should also be able to recognize, represent, manipulate, interpret, and use (act on) both fuzzy and statistical uncertainties.

Fuzzy interpretations of data structures are a very natural and intuitively plausible way to formulate and solve various problems. Conventional (i.e., crisp) sets contain objects that satisfy *precise properties* required for membership. The set H of numbers from 6 to 8 is crisp; we write $H = \{r \in \Re \mid 6 \le r \le 8\}$, where $\Re$ is the set of real numbers. Equivalently, H is described by its *membership* (or characteristic, or indicator) *function* (MF), $\mu_H : \Re \to \{0,1\}$, defined as

$$\mu_H(r) = \left\{\begin{matrix} 1 & 6 \le r \le 8 \\ 0 & \text{otherwise} \end{matrix}\right\}.$$

Every real number (r) either is in H or is not. Since μ_H maps all real numbers $r \in \Re$ onto the two points (0,1), crisp sets correspond to a two-valued logic: is or is not, on or off, black or white, 1 or 0. In logic, values of μ_H are called truth-values with reference to the question, "Is r in H?" The answer is yes if and only if $\mu_H(r) = 1$; otherwise, no. In conventional set theory, sets of real objects, such as the numbers in *H*, are equivalent to, and isomorphically described by, a unique membership function such as μ_H. However, there is no set-theory equivalent of "real objects" corresponding to fuzzy sets. Fuzzy sets are always (and only) *functions*, from a "universe of objects," say *X*, into [0,1]. As defined, *every* function $\mu : X \to [0,1]$ is a fuzzy set. While this is true in a formal mathematical sense, many functions that qualify on this ground cannot be suitably interpreted as realizations of a conceptual fuzzy set. In other words, functions that map *X* into the unit interval *may* be fuzzy sets, but *become* fuzzy sets when, and only when, they match some intuitively plausible semantic description of imprecise properties of the objects in *X*.

Defining the real numbers between 6 and 8 is a problem that is inherently crisp (i.e., mechanistic system) and would not require the use of fuzzy sets. A situation closer to what we would find in everyday life (i.e., humanistic system) consists of deciding whether a person is tall or not. The property "tall" is fuzzy *per se*. Indeed, reasoning according to Aristotelian logic, we would need to define a height threshold that divides tall people from non-tall ones. If someone is taller than the threshold (even by 1/10 of an inch) than he or she is tall, otherwise, not tall. This is obviously far from the way we decide whether someone is tall or not. Our perception of the person is better described as a sort of soft switching rather than a threshold mechanism. This is also why we often add a modifier to the word "tall" (i.e., not, not very, somewhat, very, etc.) in order to express "degrees of tall" rather than absolute true or false answers. The difference in a fuzzy and a crisp definition of tall is illustrated in Fig. 1.1 where for different heights the corresponding

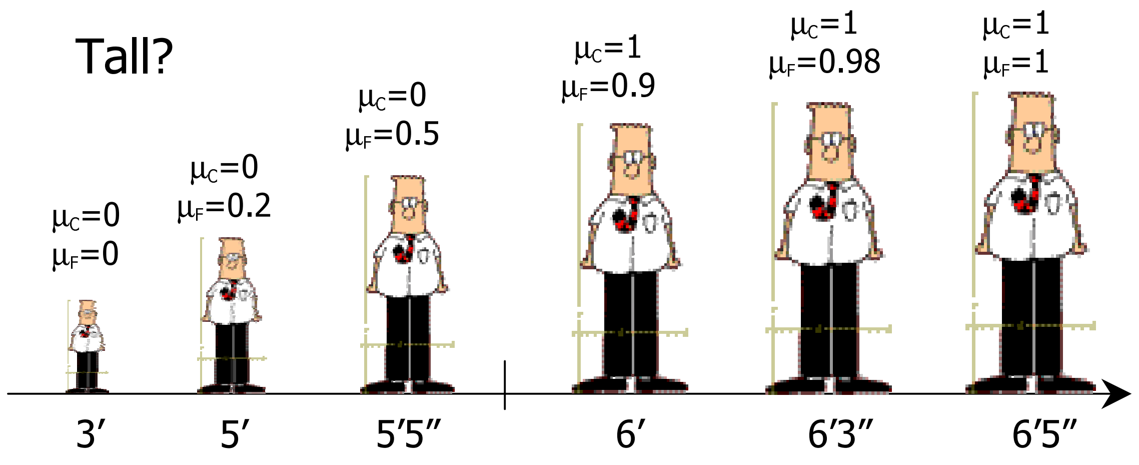

Figure 1.1. Crisp and fuzzy sets for the attribute "tall person"

degree of membership to some subjective crisp and fuzzy sets tall are indicated with μ_C and μ_F, respectively. In defining the crisp "tall person" set we fixed a threshold somewhere between 5'5" and 6', say 5'10". Therefore, someone who is 5'9" would not be tall, while someone who is 5'11" would. Conversely, in the fuzzy set "tall person" a degree of tall is defined, thus providing a *continuum* rather than an abrupt transition from true to false.

Consider next the set F of real numbers that are *close to 7*. Since the property "close to 7" is fuzzy (as the property "tall person" is), there is *not a unique* membership function for F. Rather, the modeler must decide, based on the potential application and properties desired for F, what μ_F should be. Properties that might seem plausible for μ_F include:

(i) Normality, i.e., $\mu_F(7) = 1$;

(ii) Monotonicity, i.e., the closer r is to 7, the closer $\mu_F(r)$ is to 1, and conversely;

(iii) Symmetry, i.e., numbers equally far left and right of 7 should have equal memberships.

Given these intuitive constraints, a lot of different functions could be a representation for F. One can easily construct a MF for F so that every number has some positive membership in F, but we would not expect numbers "far from 7," 10^9 for example, to have much! One of the biggest differences between crisp and fuzzy sets is that the former generally have unique MFs, whereas every fuzzy set has an infinite number of MFs that may represent it. This is at once both a weakness and strength; uniqueness is sacrificed, but this gives a concomitant gain in terms of flexibility, enabling fuzzy models to be "adjusted" for maximum utility in a given situation.

One of the first questions asked about this scheme, and the one that is still asked most often, concerns the **relationship of fuzziness to probability**. Are fuzzy sets just a clever disguise for statistical models? Well, in a word, NO. Perhaps an example will help.

Example 1. *Potable liquid: fuzziness and probability*. Let the set of all liquids be the universe of objects, and let fuzzy subset L = {all *potable* (i.e., "suitable for drinking") liquids}. Suppose you had been in the desert for a week without drink and you came upon two bottles, A and B. You are told that the (fuzzy) membership of the liquid in A to L is 0.9 and also that the probability that the liquid in B belongs to L is 0.9. In other words, A contains a liquid that is potable with *degree of membership* 0.9, while B contains a liquid that is potable with *probability* 0.9. Confronted with this pair of bottles and given that you must drink from the one that you choose, which would *you* choose to drink from first? Why? Moreover, after an observation is made regarding the content of both bottles what are the (possible) values for membership and probability? The bottle you should drink from is *A*, because this 0.9 value means that the liquid contained in *A* is fairly close to being a potable liquid[1], thus it is very likely to not be harmful. On the other hand, *B* will contain a liquid that is very probably potable but it could be very harmful for us 1 out of 10 times on average, so we could be drinking sulfuric acid from *B*! Moreover, after an observation is made and the content of the bottles is revealed, the membership for *A* stays the same while the probability for *B* changes and becomes either 0 or 1 depending on the fact that the liquid inside is potable or not.

Another common misunderstanding about fuzzy models over the years has been that they were offered as *replacements* for crisp (or probabilistic) models. To expand on this, first note that every crisp set is fuzzy, but not conversely. Most schemes that use the idea of fuzziness use it in this sense of *embedding*; that is, we work at preserving the conventional structure, and letting it dominate the output whenever it can, or whenever it must. Another example will illustrate this idea.

Example 2. *Taylor series of the bell-shaped function.* Consider the plight of early mathematicians, who knew that the Taylor series for the real (bell-shaped) function $f(x) = 1 / (1 + x^2)$ was divergent at $x = \pm 1$ but could not understand why, especially since f is differentiable infinitely often at these two points. As is common knowledge for any student of complex variables nowadays, the *complex* function $f(z) = 1 / (1 + z^2)$ has poles at $z = \pm i$, two purely imaginary numbers. Thus, the complex function, which is an embedding of its real antecedent, cannot have a convergent power series expansion anywhere on the boundary of the unit disk in the plane; in particular at $z = \pm 0i \pm 1$, i.e., at the real numbers $x = \pm 1$. This exemplifies a general principle in mathematical modeling: given a real (seemingly insoluble) problem; enlarge the space, and look for a solution in some

[1] Unless the modeler that assigned a 0.9 membership value to this values was wrong.

“imaginary” superset of the real problem; finally, specialize the “imaginary” solution to the original real constraints.

In Example 2 we spoke of “complexifying” the function *f* by embedding the real numbers in the complex plane, followed by “decomplexification” of the more general result to solve the original problem. Most fuzzy models follow a very similar pattern. Real problems that exhibit non-statistical uncertainty are first “fuzzified,” some type of analysis is done on the larger problem, and then the results are specialized back to the original problem. In Example 2 we might call the return to the real line decomplexifying the function; in fuzzy models, this part of the procedure has come to be known as defuzzification. Defuzzification is usually necessary, of course, because even though we instruct a student to “apply the brakes when approaching the crosswalk,” in fact, the brake pedal must be operated crisply, at some real time. In other words, we cannot admonish a motor to “speed up a little,” even if this instruction comes from a fuzzy controller we must alter its voltage by a specific amount. Thus defuzzification is both natural and necessary.

<u>*Example 3.*</u> *Inverted pendulum.* As a last, and perhaps more concrete, example about the use of fuzzy models, consider a simple inverted pendulum free to rotate in a vertical plane on a pivot attached to a cart. The control problem is to keep the pendulum vertical at all times by applying a restoring force (control signal) $F(t)$ to the cart at some discrete times (t) in response to changes in both the linear and angular position and velocity of the pendulum. This problem can be formulated in many ways. In one of the simpler versions used in conventional control theory, linearization of the equations of motion results in a model of the system whose stability characteristics are determined by examination of the *real parts* of the eigenvalues $\{\lambda_i\}$ of a 4×4 matrix of system constants. It is well known that the pendulum can be stabilized by requiring $\mathrm{Re}(\lambda_i) < 0$. This procedure is so commonplace in control engineering that most designers do not even think about the use of imaginary numbers to solve real problems, but it is clear that this process is exactly the same as was illustrated in Example 2 – a real problem is solved by temporarily passing to a larger, imaginary setting, analyzing the situation in the superset, and then specializing the result to

get the desired answer. This is the case for a lot of problems in science and engineering, from the use of Laplace, Fourier, and Z-transform, to phasors, etc.

An alternative solution to this control problem is based on fuzzy sets. This approach to stabilization of the pendulum is also well known, and yields a solution that in some ways is much better; e.g., the fuzzy controller is much less sensitive to changes in parameters such as the length and mass of the pendulum [35]. Note again the embedding principle: fuzzify, solve, defuzzify, control. The point of Example 3? Fuzzy models are not really that different from more familiar ones. Sometimes they work better, and sometimes not. This is really the only criterion that should be used to judge any model, and there is much evidence nowadays that fuzzy approaches to real problems are often a good alternative to more familiar schemes.

Let us now discuss a little bit about the history of fuzzy sets. The enormous success of commercial applications that are at least partially dependent on fuzzy technologies fielded (in the main) by Japanese companies has led to a surge of curiosity about the utility of fuzzy logic for scientific and engineering applications. Over the last two decades, fuzzy models have supplanted more conventional technologies in many scientific applications and engineering systems, especially in control systems and pattern recognition. A *Newsweek* article indicates that the Japanese now hold thousands of patents on fuzzy devices used in applications as diverse as washing machines, TV camcorders, air conditioners, palm-top computers, vacuum cleaners, ship navigators, subway train controllers, and automobile transmissions [91]. It is this wealth of deployed, successful applications of fuzzy technology that is, in the main, responsible for current interest in the subject area.

Since 1965, many authors have generalized various parts of subdisciplines in mathematics, science, and engineering to include fuzzy cases. However, interest in fuzzy models was not really very widespread until their utility in a wide field of applications became apparent. The reasons for this delay in interest are many, but perhaps the most accurate explanation lies with the salient facts underlying the development of any new technology. Every new technology begins with naive euphoria – its inventor(s) are usually submersed in the ideas themselves; it is their immediate colleagues that experience most of the wild enthusiasm. Most technologies are initially over promised, more often than not simply to generate funds to continue the work, for funding is an integral part of scientific development; without it, only the most imaginative and revolutionary ideas make it beyond the embryonic stage. Hype is a natural companion to over-promise, and most technologies build rapidly to a peak of hype. Following this, there is almost always an overreaction to ideas that are not fully developed, and this inevitably leads to a crash of sorts, followed by a period of wallowing in the depths of cynicism. Many new technologies evolve to this point, and then fade away. The ones that survive do so because someone finds a good use (i.e., true user benefit) for the basic ideas. What constitutes a “good use”? For example, there are now many “good uses” in real systems for complex numbers, as we have seen in Examples 2 and 3, but not many mathematicians thought so when Wessel, Argand, Hamilton, and Gauss made imaginary numbers sensible from a geometric point of view in the later 1800s. And in the context of fuzzy models, of course, “good use” corresponds to the plethora of products alluded to above.

1.3 Fuzzy Set Theory

This section introduces some elements of fuzzy set theory in a more formal way than the previous one. The properties and features of classical set theory are used to introduce their corresponding fuzzy counterparts. Most of the operators and essential definitions are also collected in a glossary in the front of the dissertation.

Let X be a space of objects and x be a generic element of X. A classical set A, $A \subseteq X$, is defined as a collection of elements or objects $x \in X$, such that each element (x) can either belong or not to the set A. By defining a **characteristic** (or **membership**) **function** for each element x in X, we can represent a classical set A by a set of ordered pairs $(x,0)$ or $(x,1)$, which indicates $x \notin A$ or $x \in A$, respectively. Unlike a conventional set, a fuzzy set expresses the *degree* to which an element belongs to a set. Hence the membership function of a fuzzy set is allowed to have values *between* 0 and 1 that denote the *degree of membership* of an element in the given set.

Definition 1.1. *Fuzzy sets and membership functions*. If X is a collection of objects denoted generically by x, then a **fuzzy set** A in X is defined as a set of ordered pairs $A = \{(x,\mu_A(x)) \mid x \in X\}$, where, $\mu_A(x)$ is called the **membership function** (MF) for the fuzzy set A. The MF maps each element of X to a membership degree between 0 and 1 (included).

Obviously, the definition of a fuzzy set is a simple extension of the definition of a classical (crisp) set in which the characteristic function is permitted to have any value between 0 and 1. If the value of the membership function is restricted to either 0 or 1, then A is reduced to a classical set. For clarity, we shall also refer to classical sets as ordinary sets, crisp sets, non-fuzzy sets, or just sets. Usually, X is referred to as the **universe of discourse,** or simply the **universe,** and it may consist of discrete (ordered or non-ordered) objects or it can be a continuous space. This can be clarified by the following examples.

Example 4. *Fuzzy sets with a discrete non-ordered universe*. Let X = {Baltimore, San Francisco, Boston, Los Angeles} be the set of cities one may choose to live in. The fuzzy set A = "desirable city to live in" may be described as follows: A = {(Baltimore, 0.95), (San Francisco, 0.9), (Boston, 0.8), (Los Angeles, 0.2)}. Apparently, the universe of discourse X is discrete and it contains non-ordered objects – in this case, four big cities in the United States. As one can see, the foregoing membership grades listed above are quite subjective; anyone can come up with four different but legitimate values to reflect his or her preference.

Example 5. *Fuzzy sets with a discrete ordered universe*. Let $X = \{0, 1, 2, 3, 4, 5, 6\}$ be the set of numbers of children a family may choose to have. Then the fuzzy set B = "desirable number of children in a family" may be described as follows: $B = \{(0, 0.1), (1, 0.3), (2, 1.0), (3, 0.8), (4, 0.7), (5, 0.3), (6, 0.1)\}$. Here we have a discrete ordered universe X. Again, the membership grades of this fuzzy set are obviously subjective measures.

Example 6. *Fuzzy sets with a continuous universe*. Let $X = \Re^+$ (i.e., the set of non-negative real numbers) be the set of possible ages for human beings. Then the fuzzy set C = "about 50 years old" may be expressed as $C = \{(x, \mu_C(x)) \mid x \in X\}$, where

$$\mu_C(x) = \frac{1}{1+\left(\frac{x-50}{10}\right)^4}.$$

From the previous examples, it is obvious that the construction of a fuzzy set depends on two things: the identification of a suitable universe of discourse and the specification of an appropriate membership function. The specification of membership functions is *subjective*, which means that the membership functions specified for the same concept by different persons may vary considerably. This subjectivity comes from individual differences in perceiving or expressing abstract concepts and has little to do with randomness. Therefore, the **subjectivity** and **non-randomness** of fuzzy sets is the primary difference between the study of fuzzy sets and probability theory, which deals with objective treatment of random phenomena.

In practice, when the universe of discourse X is a continuous space, we usually partition it into several fuzzy sets whose MFs cover X in a more or less uniform manner. These fuzzy sets, which usually carry names that conform to adjectives appearing in our daily linguistic usage, such as "large," "medium," or "small," are called *linguistic values* or *linguistic labels*. Thus, the universe of discourse X is often called the *linguistic variable*. An example on this follows.

Example 7. *Linguistic variables and linguistic values*. Suppose that X = "age." Then we can define fuzzy sets "young," "middle aged," and "old" that are characterized by MFs. Just as a variable can assume various values, a linguistic variable "age" can assume different linguistic values, such as "young," "middle aged," and "old" in this case. If "age" assumes

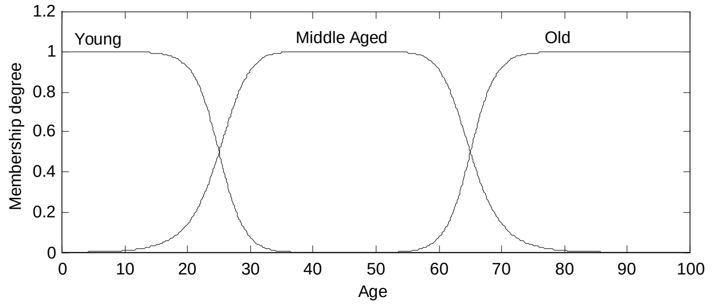

Figure 1.2. Example MFs of linguistic values "young", "middle aged", and "old"

the value of "young," then we have the expression "age is young," and so forth for the other values. An example of MFs for these linguistic values is displayed in Fig. 1.2, where the universe of discourse *X* is totally covered by the MFs and the transition from one MF to another is smooth and gradual.

We will now define some nomenclature used in the literature.

Definition 1.2. *Support*. The **support** of a fuzzy set *A* is the set of all points with nonzero membership degree in *A*:

$$\text{support}(A) = \{x \in X \,|\, \mu_A(x) > 0\}$$

Definition 1.3. *Core*. The **core** of a fuzzy set *A* is the set of all points with unit membership degree in *A*:

$$\text{core}(A) = \{x \in X \,|\, \mu_A(x) = 1\}$$

Definition 1.4. *Normality*. A fuzzy set *A* is **normal** if its core is nonempty. In other words, we can always find at least a point $x \in X$ such that $\mu_A(x) = 1$.

Definition 1.5. *Crossover points*. A **crossover point** of a fuzzy set *A* is a point $x \in X$ at which $\mu_A(x) = 0.5$.

Definition 1.6. *Fuzzy singleton*. A fuzzy set whose support is a single point in *X* with $\mu_A(x) = 1$ is called a **fuzzy singleton.**

Definition 1.7. *α-cut, strong α-cut*. The **α-cut** or **α-level set** of a fuzzy set *A* is a crisp set defined by $A_\alpha = \{x \in X \mid \mu_A(x) \geq \alpha\}$. **Strong α-cut** or **strong α-level set** are defined

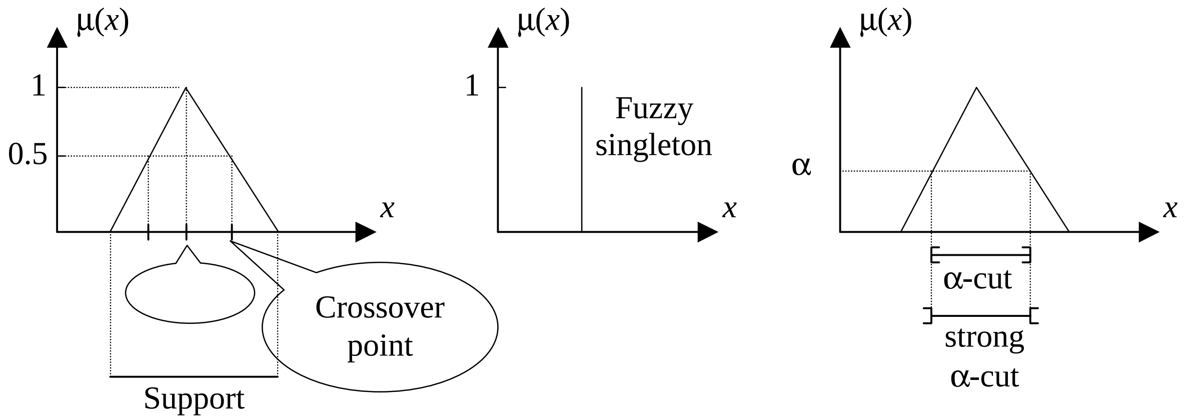

Figure 1.3. Graphical examples of core, support, crossover point, fuzzy singleton, α-cut and strong α-cut

similarly: $A'_\alpha = \{x \in X \mid \mu_A(x) > \alpha\}$.

Using this notation, we can express the support and core of a fuzzy set A as:

$$\text{support}(A) = A'_0 \quad \text{and} \quad \text{core}(A) = A_1.$$

The entities previously defined are graphically illustrated in Fig. 1.3.

Definition 1.8. *Convexity*. A fuzzy set A is **convex** if and only if:

$$\forall x_1, x_2 \in X \wedge \forall\lambda \in [0,1], \mu_A[\lambda x_1 + (1-\lambda)x_2] \geq \min\{\mu_A(x_1), \mu_A(x_2)\}.$$

Alternatively, A is convex if all its α-level sets are convex. Note that the definition of convexity of a fuzzy set is not as strict as the common definition of convexity of a function. Indeed, it corresponds to the definition of quasi-concavity of a function.

Definition 1.9. *Fuzzy numbers*. A fuzzy number A is a fuzzy set in the real line that satisfies the conditions for both normality and convexity. Most fuzzy sets used in the literature satisfy the conditions for normality and convexity, so fuzzy numbers are the most basic type of fuzzy sets.

Union, intersection, and complement are the most basic operations on classical sets. On the basis of these three operations, a number of identities can be established. Corresponding to the ordinary set operations of union, intersection, and complement, fuzzy sets have similar operations, which were initially defined in Zadeh's seminal paper [88]. Before introducing these three fuzzy set operations, first we shall define the notion of containment, which plays a central role in both ordinary and fuzzy sets. This definition of containment is, of course, a natural extension of the one for ordinary sets.

Definition 1.10. *Containment or subset*. Fuzzy set A is **contained** in fuzzy set B (or, equivalently, A is a **subset** of B, or A is smaller than or equal to B, $A \subseteq B$) if and only if:

$$\forall x \in X : \mu_A(x) \leq \mu_B(x).$$

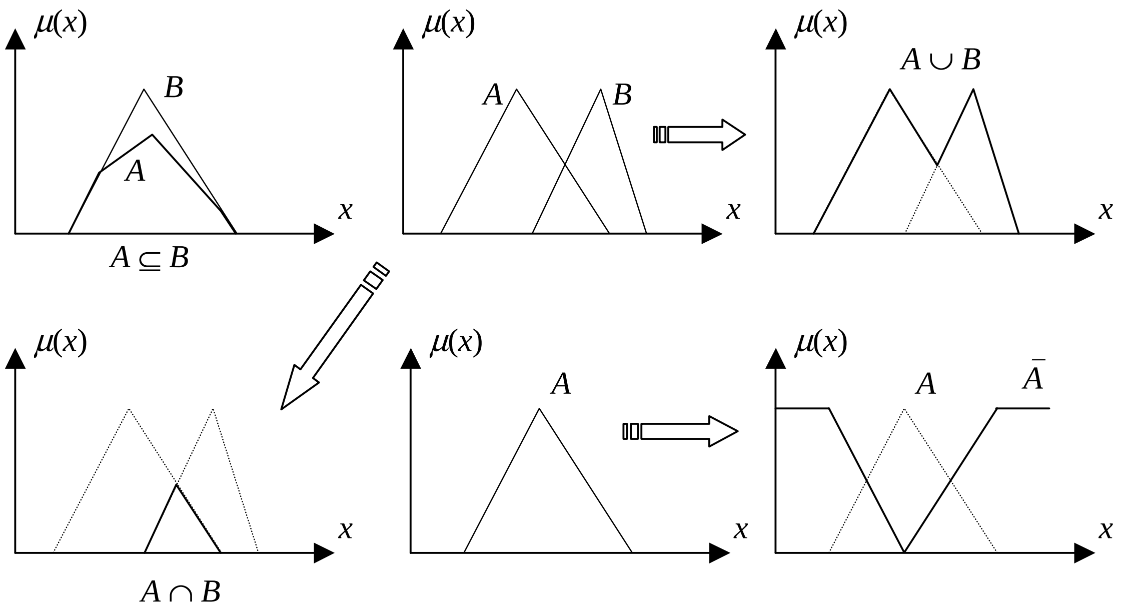

Figure 1.4. Graphical examples of containment, union, intersection, and complement

Definition 1.11. *Union (disjunction).* The **union** of two fuzzy sets A and B is a fuzzy set C, written as $C = A \cup B$ or $C = A$ OR B, whose MF is related to those of A and B by:

$$\mu_C(x) = \max(\mu_A(x), \mu_B(x)).$$

Definition 1.12. *Intersection (conjunction).* The **intersection** of two fuzzy sets A and B is a fuzzy set C, written as $C = A \cap B$ or $C = A$ AND B, whose MF is related to those of A and B by:

$$\mu_C(x) = \min(\mu_A(x), \mu_B(x)).$$

Definition 1.13. *Complement (negation).* The complement of fuzzy set A, denoted by $\overline{A}$ or NOT A, is defined as $\mu_{\overline{A}}(x) = 1 - \mu_A(x)$.

The containment property and the operations of union, intersection, and complement introduced in the previous Definitions (1.10) to (1.13) are graphically illustrated in Fig. 1.4. Note that these operations perform exactly as the corresponding operations for ordinary sets if the values of the membership functions are restricted to either 0 or 1. However, it is understood that these functions are not the only possible generalizations of the crisp set operations. For each of the aforementioned three set-operations, several different classes of functions with desirable properties have been proposed subsequently in the literature (e.g., algebraic sum for union and product for intersection). In general, union and intersection of two fuzzy sets can be defined through generic *t-conorm* (or *s-norm*) and *t-norm* operators respectively. As pointed out by Zadeh [88], a more intuitive but equivalent definition of union is the, "smallest" fuzzy set containing both A and B. Alternatively, if D is any fuzzy set that contains both A and B, then it also contains $A \cup B$. Analogously, the intersection of A and B is the "largest" fuzzy set which is contained in both A and B. Thus we can revise Definitions (1.11) and (1.12).

Definition 1.11r. *Union (disjunction).* The **union** of two fuzzy sets A and B is a fuzzy set C (written as $C = A \cup B$ or $C = A$ OR B) that is the "smallest" fuzzy set containing both A and B. Its MF is related to those of A and B by

$$\mu_C(x) = \mu_A(x) \oplus \mu_B(x).$$

Definition 1.12r. *Intersection (conjunction).* The **intersection** of two fuzzy sets A and B is a fuzzy set C (written as $C = A \cap B$ or $C = A$ AND B) that is the "largest" fuzzy set which is contained in both A and B. Its MF is related to those of A and B by

$$\mu_C(x) = \mu_A(x) \otimes \mu_B(x).$$

In Definitions (1.11r) and (1.12r) the symbols $\oplus$ and $\otimes$ represent a generic choice of *t-conorm* and *t-norm*, respectively. These two operators are functions $\oplus,\otimes : [0,1]\times[0,1] \rightarrow [0,1]$ satisfying some convenient boundary, monotonicity, commutativity and associativity properties. Very common choices for *t-conorms* are max and algebraic sum, while common choices for *t-norm* are min and product.

The two fundamental (Aristotelian) laws of crisp set theory are:

- *Law of Contradiction*: $A \cup \overline{A} = X$ (i.e., a set and its complement must comprise the universe of discourse, any object must belong to a set or to its complement);

- *Law of Excluded Middle*: $A \cap \overline{A} = \varnothing$ (i.e., a set and its complement are disjoint, any object can only be in one of either a set or its complement, it cannot simultaneously be in both).

It can be easily noted that for every fuzzy set that is non-crisp (i.e., whose membership function does not only assume values 0 and 1) both laws are broken (i.e., for fuzzy sets $A \cup \overline{A} \neq X$ and $A \cap \overline{A} \neq \varnothing$). Indeed $\forall x \in A$ such that $\mu_A(x) = \alpha$, $0 < \alpha < 1$: $\mu_{A \cup \overline{A}}(x) = \max\{\alpha, 1\text{-}\alpha\} \neq 1$ and $\mu_{A \cap \overline{A}}(x) = \min\{\alpha, 1\text{-}\alpha\} \neq 0$. In fact, one of the ways to describe the difference between crisp set theory and fuzzy set theory is to explain that these two laws do not hold in fuzzy set theory. Consequently, any other mathematics that relies on crisp set theory, such as (frequency based) probability, must be different from fuzzy set theory.

We will now introduce the concept of *relations* in both crisp and fuzzy sets; this will later help us in approaching fuzzy logic. A **crisp relation** represents the *presence or absence* of association, interaction or interconnectedness between the elements of two or more sets. Given two sets X and Y a relation R between X and Y is itself a set $R(X,Y)$ subset of $X \times Y$[2]. For example, the ordering relation "less than" ($<$) is a relation in $\mathfrak{R}^2$ defined as $LT(\mathfrak{R},\mathfrak{R}) = \{(x,y) \mid x < y\}$. The point (1,123) belongs to $LT(\mathfrak{R},\mathfrak{R})$ while obviously (123,1) does not.

Definition 1.14. *Fuzzy relation.* A **fuzzy relation** represents a *degree of presence or absence* of association, interaction or interconnectedness between the elements of two or more sets. Some examples of (binary) fuzzy relations are: x is much larger than y, y is very close to x, z is much greener than y. Let X and Y be two universes of discourse. A fuzzy relation $R(X,Y)$ is a fuzzy set in the product space $X \times Y$, i.e., it is a fuzzy subset of $X \times Y$, and is characterized by the membership function $\mu_R(x,y)$:

$$R(X,Y) = \{((x,y),\mu_R(x,y)) \mid (x,y) \in X \times Y\}.$$

The difference between a fuzzy relation and a crisp relation is that for the former any membership value in [0,1] is allowed while for the latter only 0 and 1 memberships are.

[2] By $X \times Y$ we indicate the Cartesian product of sets X and Y, that is the set of ordered couples with values from X and Y, respectively, i.e., $X \times Y = \{(x,y) \mid x \in X \text{ and } y \in Y\}$.

This is why a fuzzy relation is expressing not only the interconnection between the elements of two or more sets (e.g., as a crisp relation does) but also the degree or extent of this association. Because fuzzy relations are fuzzy sets in product space, set theoretic operations can be defined for them using Definitions (1.11) through (1.13).

Next, we consider the composition of crisp relations from different product spaces that share a common set, namely $P(X,Y)$ and $Q(Y,Z)$. The *composition* of these two relations is denoted by $R(X,Z) = P(X,Y) \circ Q(Y,Z)$ and is defined as:

$$R(X,Z) \subseteq X \times Z : (x,z) \in R(X,Z) \Leftrightarrow \exists y \in Y : (x,y) \in P(X,Y) \wedge (y,z) \in Q(Y,Z).$$

This can be expressed in terms of characteristic functions through either the max-min or the max-product compositions respectively defined as

$$\begin{gathered}\mu_{P\circ Q}(x,z) = \max_y \{\min[\mu_P(x,y), \mu_Q(y,z)]\} \\ \mu_{P\circ Q}(x,z) = \max_y [\mu_P(x,y)\mu_Q(y,z)]\end{gathered} \tag{1.1}$$

The composition of fuzzy relations from different product spaces that share a common set is defined analogously to the crisp composition, except that in the fuzzy case the sets are fuzzy.

Definition 1.15. *Composition of fuzzy relations*. Given two relations $P(X,Y)$ and $Q(Y,Z)$ and their associated membership functions $\mu_P(x,y)$ and $\mu_Q(y,z)$, the **composition** of these two relations is denoted by $R(X,Z) = P(X,Y) \circ Q(Y,Z)$ (or simply $R = P \circ Q$) and is defined as a subset $R(X,Z)$ of $X \times Z$ defined by the membership function

$$\mu_{P\circ Q}(x,z) = \sup_{y\in Y}[\mu_P(x,y) \otimes \mu_Q(y,z)] \tag{1.2}$$

Motivation for using the *t-norm* operator ($\otimes$) comes from the crisp max-min and max-product compositions, because both the min and the product are *t-norms*. This is also sometimes referred to as *sup-star* composition due to an alternative symbol for the *t-norm* (e.g., $\star$). Although it is permissible to use other *t-norms*, the most commonly used *sup-star* compositions are the sup-min and sup-product. Observe that, unlike the case of crisp compositions, for which exactly the same results are obtained using either the max-min or the max-product composition, the same results are not obtained in the case of fuzzy compositions. This is a major difference between fuzzy composition and crisp composition.

Suppose fuzzy relation P is just a fuzzy set, so that $\mu_P(x,y)$ becomes $\mu_P(x)$, e.g., “x is medium large and z is smaller than y.” Then $Y = X$ and the membership function for the composition of P and Q becomes

$$\mu_{P\circ Q}(z) = \sup_{x\in X}\left[\mu_P(x)\otimes\mu_Q(x,z)\right] \tag{1.3}$$

Note that now this is a function of only the variable z. This equation will be useful in the following developments of fuzzy reasoning.

1.4 Fuzzy Logic

It is well established that prepositional logic is isomorphic to set theory under the appropriate correspondence between components of these two mathematical systems. Furthermore, both of these systems are isomorphic to a Boolean algebra, which is a mathematical system defined by abstract entities and their axiomatic properties. The isomorphism among Boolean algebra, set theory, and propositional logic guarantees that every theorem in any one of these theories has a counterpart in each of the other two theories. These isomorphisms allow us, in effect, to cover all these theories by developing only one of them. We will not spend a lot of time reviewing crisp logic; but we must spend some time on it, especially on the concept of implication, in order to reach the comparable concept in fuzzy logic.

Fuzzy rules are the cornerstone of fuzzy logic systems. Rules are a form of proposition. A *proposition* is an ordinary statement involving terms that have been defined, e.g., “The damping ratio is low.” Consequently, we could have the following rule: “IF the damping ratio is low, THEN the system's impulse response oscillates a long time before it dies out.” In traditional propositional logic, a proposition must be meaningful to call it “true” or “false,” whether or not we know which of these terms properly applies. Logical reasoning is the process of combining given propositions into other propositions, and then doing this over and over again. Propositions can be combined in many ways, all of which are derived from three fundamental operations: *conjunction* (denoted $p\wedge q$), where we assert the simultaneous truth of two separate propositions p and q; *disjunction* ($p \vee q$), where we assert the truth of either or both of two separate propositions; and *implication* ($p \rightarrow q$) which usually takes the form of an IF-THEN rule (also known as "production rule"). The IF part of an implication is called the *antecedent*, whereas the THEN part is called the *consequent*. In addition to generating propositions using conjunction, disjunction or implication, a new proposition can be obtained from a given one by prefixing the clause "it is false that …"; this is the operation of *negation* ($\sim p$). Additionally, $p \leftrightarrow q$ is the *equivalence* relation; it means that both p and q are either true or false.

In traditional propositional logic we combine unrelated propositions into an implication, and *we do not assume any cause or effect relation to exist*. We will see later that this last statement causes serious problems when we try to apply traditional propositional logic to engineering applications, where cause and effect rule (i.e., a – causal – system does not respond until an input is applied to it). In traditional propositional logic an implication is said to be true if one of the following holds (see also Table 1.1): 1) (antecedent is true, consequent is true), 2) (antecedent is false, consequent is false), 3) (antecedent is false, consequent is true); the implication is called false when 4) (antecedent is true, consequent is false). Situation 1) is the familiar one of common experience. Situation 2) is also reasonable, for if we start from a false assumption we expect to reach a false conclusion, however, intuition is not always reliable. We may reason correctly from a false antecedent to a true consequent (e.g., IF 1 = 2 is false, but, adding 2 = 1 to this false statement, let us correctly conclude that 3 = 3); hence, a false antecedent can lead to a consequent which is either true or false, and thus both situations 2) and 3) are allowed in traditional propositional logic. Finally, situation 4) is in accord with our intuition, for an implication is clearly false if a true antecedent leads to a false consequent.

A logical structure is constructed by applying the above four operations to propositions. The objective of a logical structure is to determine the truth or falsehood of all propositions that can be stated in the terminology of this structure. *A truth table* is very convenient for showing relationships between several propositions. The fundamental truth tables for conjunction, disjunction, implication, equivalence and negation are collected together in Table 1.1, in which symbol T means that the corresponding proposition is true, and symbol F that it is false. The fundamental axioms of traditional propositional logic are:

Table 1.1. Truth table for five proposition operations

p	***q***	***p∧q***	***p∨q***	***p→q***	***p↔q***	***~p***
T	*T*	*T*	*T*	*T*	*T*	*F*
T	*F*	*F*	*T*	*F*	*F*	*F*
F	*T*	*F*	*T*	*T*	*F*	*T*
F	*F*	*F*	*F*	*T*	*T*	*T*

1) every proposition is either true or false, but not both true and false (laws of contradiction and excluded middle), 2) the expressions given by defined terms are propositions, and, 3) the truth table (in Table 1.1) for conjunction, disjunction, implication, equivalence, and negation. Using truth tables, we can derive many interpretations of the preceding operations and can also prove relationships about them.

A *tautology* is a proposition formed by combining other propositions, which is true regardless of the truth or falsehood of the forming propositions. The most important tautologies for our work are: $(p \to q) \leftrightarrow \sim[p \wedge (\sim q)] \leftrightarrow (\sim p) \vee q$. These tautologies can be verified substituting all the possible combinations for p and q and observing how the equivalence always holds true. The importance of these tautologies is that they let us express the membership function for $p \to q$ in terms of membership functions of either propositions p and $\sim q$ or $\sim p$ and q. Thus yielding

$$\begin{gathered}\mu_{p\to q}(x,y) = 1 - \mu_{p\cap\bar{q}}(x,y) = 1 - \{\mu_p(x) \otimes [1 - \mu_q(y)]\} \\ \mu_{p\to q}(x,y) = \mu_{\bar{p}\cup q}(x,y) = [1 - \mu_p(x)] \oplus \mu_q(y)\end{gathered} \tag{1.4}$$

These two equations can be verified substituting 1 for true and 0 for false, they hold for any choice of max or sum for *t-conorm* and min or product for *t-norm*.

In traditional propositional logic there are two very important inference rules, *Modus Ponens* and *Modus Tollens.*

Modus Ponens: *Premise 1:* "*x* is *A*";

Premise 2: "IF *x* is *A* THEN *y* is *B*",

Consequence: "*y* is *B*."

Modus Ponens is associated with the implication "*A* implies *B*." In terms of propositions *p* and *q*, Modus Ponens is expressed as : $(p \wedge (p \rightarrow q)) \rightarrow q$.

Modus Tollens: *Premise 1*: "*y* is not *B*";
Premise 2: "IF *x* is *A* THEN *y* is *B*";
Consequence: "*x* is not *A*."

In terms of propositions *p* and *q*, Modus Tollens is expressed as $((\sim q) \wedge (p \rightarrow q)) \rightarrow (\sim p)$.

Fuzzy logic begins by borrowing notions from crisp logic, just as fuzzy set theory; however, as we shall demonstrate below, doing this is inadequate for engineering applications of fuzzy logic, because, in engineering, cause and effect is the cornerstone of modeling, whereas in traditional propositional logic it is not. Ultimately, this will cause to define "engineering" implication operators [43]. Before doing this, let us develop an understanding of why the traditional approach fails us in engineering. As in our extension of crisp set theory to fuzzy set theory, replacing the bivalent membership functions of crisp logic with fuzzy membership functions makes our extension of crisp logic to fuzzy logic. Hence, the IF-THEN statement "IF *x* is *A*, THEN *y* is *B*," where $x \in X$ and $y \in Y$, has a membership function $\mu_{A \rightarrow B}(x,y) \in [0,1]$. Note that $\mu_{A \rightarrow B}(x,y)$ measures the degree of truth of the implication relation between *x* and *y*. This membership function can be defined as for the crisp case above. In fuzzy logic, Modus Ponens is extended to *Generalized Modus Ponens*.

Generalized Modus Ponens: *Premise 1:* "*x* is *A**";
Premise 2: "IF *x* is *A* THEN *y* is *B*";
Consequence: "*y* is *B**."

Compare Modus Ponens and Generalized Modus Ponens to see their subtle differences, namely, in the latter, fuzzy set *A** is not the necessarily the same as rule antecedent fuzzy set *A*, and fuzzy set *B** is not necessarily the same as rule consequent *B*.

Example 8. *Generalized modus ponens.* Consider the rule “IF a man is short, THEN he will not make a very good professional basketball player.” Here fuzzy set *A* is *short man* and fuzzy set *B* is *not a very good professional basketball player*. We are now given Premise 1, as “This man is under five feet tall.” Here *A** is the fuzzy set *man under five feet tall*. Clearly *A* and *A** are different but similar. We now draw the following consequence: “He will make a poor professional basketball player.” Here *B** is the fuzzy set *poor professional basketball player*, and it is different from *B*, although they are indeed similar. Note how Premise 1 could have been “This man is five feet tall” (this would correspond to a fuzzy singleton) and we would have reached the same conclusion.

We see that in crisp logic a rule will be fired (i.e., action taken on it) only if the first premise is exactly the same as the antecedent of the rule, and, the result of such rule firing is the rule’s actual consequent. In fuzzy logic, on the other hand, a rule is fired so long as there is a nonzero degree of similarity between the first premise and the antecedent of the rule, and the result of such rule firing is a consequent that has nonzero degree of similarity to the rule consequent. Generalized Modus Ponens is a fuzzy composition where the first fuzzy relation is merely the fuzzy set *A**. Consequently, $\mu_{B*}(y)$ is obtained from the sup-star composition as

$$\mu_{B^*}(y) = \sup_{x \in A^*}\left[\mu_{A^*}(x) \otimes \mu_{A \to B}(x, y)\right] \tag{1.5}$$

Let us now think at an application of this approach. Given an observation x_1 we want to determine what is the correct action (i.e., reaction) y_1 corresponding to this observation. The observation needs to correspond to the first premise in generalized modus ponens, thus it needs to be a fuzzy set (e.g., *A**). But it is really a crisp number, thus it needs to first be transformed into a fuzzy set (*fuzzification*); the rule (or rules) is then processed producing an output fuzzy set (*B**) that needs to be transformed into a crisp number (*defuzzification*) to be useful in the real world. The operations that we just described correspond to the mode of functioning of a fuzzy logic system. Thus in a FLS, a (crisp) input is fuzzified, then processed by an inference engine according to some rules defined in a rule-base, and finally defuzzified to produce a usable (crisp) output. There are several types of fuzzifiers, defuzzifiers, and inference engines, their discussion is postponed to the next Section 1.5.

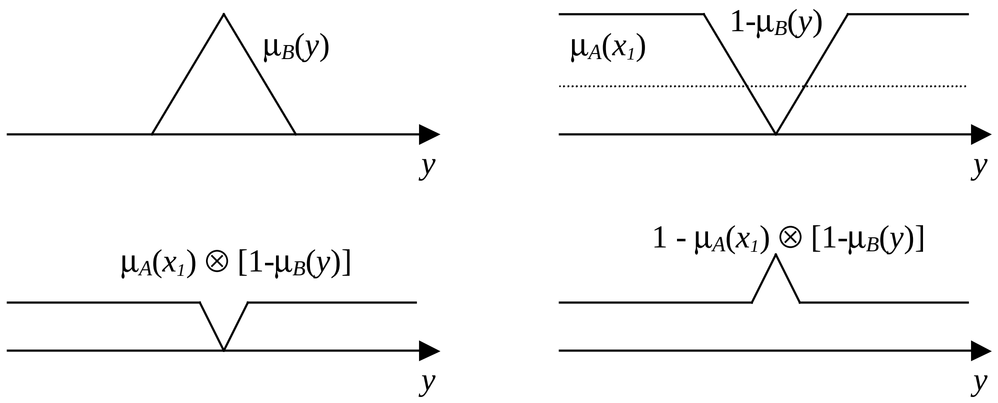
$\mu_B(y)$
y
$\mu_A(x_1)$
$1-\mu_B(y)$
y
$\mu_A(x_1) \otimes [1-\mu_B(y)]$
y
$1 - \mu_A(x_1) \otimes [1-\mu_B(y)]$
y

Figure 1.5. Graphical representation of rule processing with regular implication operators

The most popular fuzzifier is the singleton fuzzifier. In this fuzzification scheme an observation x_1 is transformed into a fuzzy set being a singleton with support $\{x_1\}$, thus $\mu_{A*}(x)$ is zero everywhere except at $x = x_1$. Applying this simplification to Equation (1.5) and using the fact that unity and zero are respectively the neutral and the null element with respect to any *t-norm* (i.e., $\mu \otimes 1 = \mu$ and $\mu \otimes 0 = 0 \;\; \forall \mu \in [0,1]$) we obtain

$$\mu_{B*}(y) = 1 \otimes \mu_{A\to B}(x_1, y) = \mu_{A\to B}(x_1, y) \tag{1.6}$$

Combining Equation (1.4) with (1.6) we finally obtain

$$\mu_{B*}(y) = 1 - \mu_A(x_1) \otimes [1 - \mu_B(y)] \tag{1.7}$$

Figure 1.5 shows a graphical interpretation of this equation using a triangular membership function for $\mu_B(y)$ (a common choice in FLSs) and min *t-norm* (an analogous result holds by using a product *t-norm*). Given a specific input $x = x_1$, the result of firing a specific rule is a fuzzy set whose support is infinite, even though the consequent B is associated with a specific fuzzy set of finite support, (the base of the triangle). Clearly, this does not make much sense from an engineering perspective, where cause (e.g., system input) should lead to effect (e.g., system output), and noncause should not lead to anything. Mamdani [40] seems to have been the first one to recognize the problem we have just demonstrated, although he does not explain it this way, that is, as in Mendel [6]. Mamdani [40] chose to work with a *minimum implication* defined as

$$\mu_{A\to B}(x,y) = \min\{\mu_A(x), \mu_B(y)\} \tag{1.8}$$

His reason for choosing this definition does not seem to be based on cause and effect, but, instead on simplicity of computation. Later, Larsen [34] proposed a *product implication* defined as

$$\mu_{A\rightarrow B}(x,y) = \mu_A(x)\,\mu_B(y) \tag{1.9}$$

Again, the reason for this choice was simplicity of computation rather than cause and effect. Today, minimum and product inferences are the most widely used inferences in the engineering applications of fuzzy logic; but what do they have to do with traditional propositional logic? It can be easily seen that neither minimum inference nor product inference agree with the accepted propositional logic definition of implication, given in Table 1.1. Hence, minimum and product inferences have nothing to do with traditional propositional logic. Interestingly enough, minimum and product inferences preserve cause and effect, i.e., the implication is fired only when the antecedent and the consequent are both true. Thus, they are sometimes collectively referred to as *engineering implications* [6].

1.5 Fuzzy Logic Systems: Principles of Operation

Fuzzy logic systems are one of the main developments and successes of fuzzy sets and fuzzy logic. A FLS is a rule-base system that implements a nonlinear mapping between its inputs and outputs. A FLS is characterized by four modules (as already introduced earlier):

- fuzzifier;
- defuzzifier;
- inference engine;
- rule base.

A schematic representation of a FLS is presented in Fig. 1.6. The operation of a FLS is based on the rules contained in the rule base. The *l*-th rule in the rule-base has the following form:

$$R^{(l)}:\ \text{IF } u_1 \text{ is } A_{1l} \text{ and } u_2 \text{ is } A_{2l} \text{ and } \ldots\ u_n \text{ is } A_{nl}\text{ , THEN } v \text{ is } B^l \tag{1.10}$$

The first n terms are called the *antecedents* of the rule while the last term (the one after the "THEN") is the *consequent* of the rule. The terms u_i are fuzzy variables and the terms A_{il}

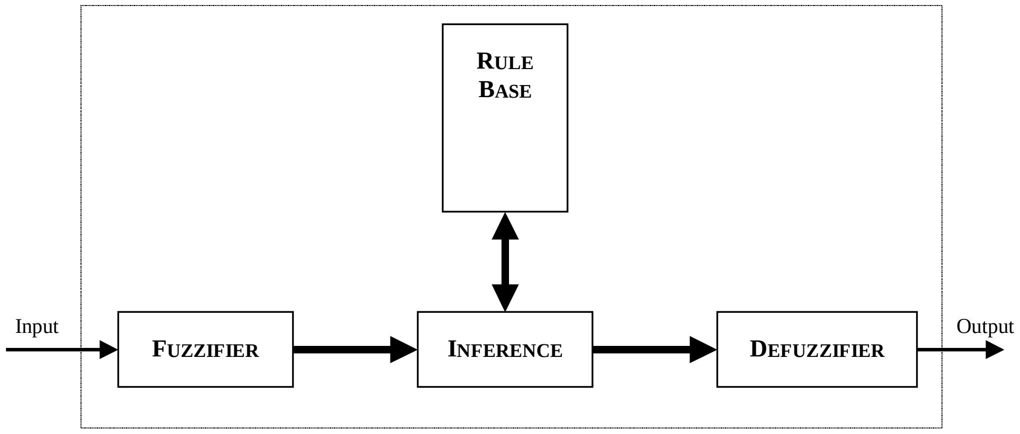

Figure 1.6. Structure of a fuzzy logic system

are linguistic variables. It can be noted that the inputs to a FLS somehow correspond to the antecedents of the rules in the rule base. A difference exists though. Indeed, the inputs to the FLS, as can be seen in Fig. 1.6, come from the outside world (e.g., controlled process) and are *crisp* variables in general. On the contrary, the antecedents of the fuzzy rules are always fuzzy sets. The role of the fuzzifier in a FLS is to convert a *crisp* input variable into a fuzzy set that is ready to be processed by the inference engine. The inference engine using the fuzzified inputs and the rules stored in the rule base processes the incoming data and produces an (fuzzy) output. This output needs to be used in the outside world and thus needs to be converted from fuzzy to crisp; the defuzzifier performs this operation. We will now expand on the operations of every module in order to finally formulate the nonlinear parameterized mapping realized by the FLS.

1.5.1 Fuzzifier

Fuzzification can be defined as the operation that maps a crisp object to a fuzzy set, i.e., to a membership function. Fuzzifiers are generally divided in *singleton* and non-*singleton* ones. A singleton fuzzifier maps an object to the singleton fuzzy set centered at the object itself (i.e., with support and core being the set containing only the given object). A non-singleton fuzzifier, maps an object to a fuzzy set generally centered at the object itself (i.e., the core of the fuzzy set contains the object) and with support containing the object but being a set bigger then only the object itself. A non-singleton fuzzifier maps an object into a non-singleton fuzzy set generally centered at the object itself. Typically, the use of a singleton fuzzifier is very common. Non-*singleton* fuzzifiers are also used, especially in the presence of e.g., noisy measurements. Indeed, in this case the input crisp value is affected by some uncertainty, thus, the corresponding input fuzzy set can reflect this uncertainty by allowing non-zero membership values around the (noisy) measurement. Therefore, when a non-*singleton* fuzzifier is used, the width of the corresponding fuzzy set is generally proportional to the amount of noise affecting the measurement. Figure 1.7 shows an example of singleton and non-singleton fuzzification.

Singleton and non-singleton fuzzifiers can be defined in more precise mathematical terms. Let $\wp$ be the set of all possible continuous membership functions over continuous sets. This can be loosely defined as

$$\wp = \{\mu \mid \mu : X \rightarrow [0,1], \mu \in C^0(X)\} \qquad (1.11)$$

where X is any continuous set (e.g., the set of real numbers $\Re$), and $C^0(X)$ denotes the set of all continuous functions in X. A singleton fuzzifier is thus a mapping

$$sg : X \rightarrow \wp \quad \ni \quad \forall\, x \in X \rightarrow \mu_x(\bullet) = sg[x] \ : \ \mu_x(z) = \begin{cases} 1 & z = x \\ 0 & z \neq x \end{cases} \qquad (1.12)$$

Analogously a non-singleton fuzzifier is a mapping

$$nsg : X \rightarrow \wp \ \ni \ \forall\, x \in X \rightarrow \mu_x(\bullet) = nsg[x] : \mu_x(x) = 1 \wedge \text{Support}[\mu_x(\bullet)] \supset \{x\} \quad (1.13)$$

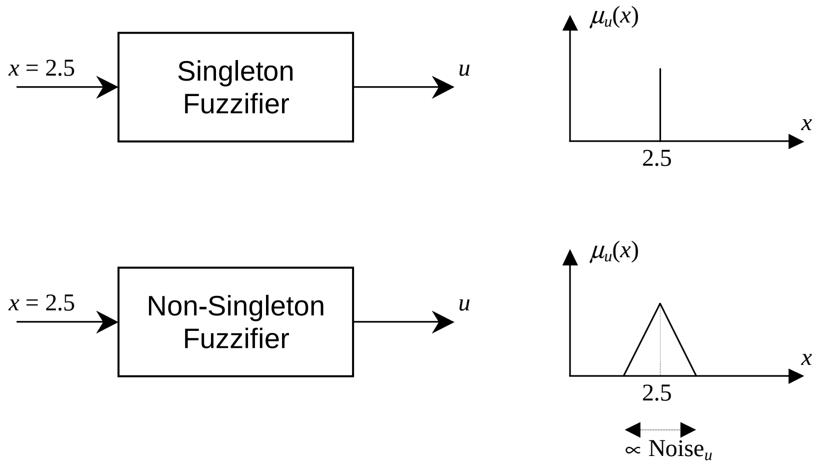

Figure 1.7. Example of singleton and non-singleton fuzzifiers

1.5.2 Inference Engine and Rule Base

Once the inputs are fuzzified, the corresponding input fuzzy sets are passed to the inference engine that processes current inputs using the rules retrieved from the rule base. The outcome of these rules in *generalized modus ponens* will be an output fuzzy set B^{l*} *close* to B^l. The input in this case is different, not being a scalar anymore, but a vector. Thus in this case we have $A = A^l = A_{1l} \times A_{2l} \times \ldots \times A_{nl}$ and $B = B^l$, but still the fuzzy engine would be mapping fuzzy sets into fuzzy sets. Thus from Equation (1.5) we obtain

$$\mu_{B^{l*}}(y) = \sup_{\boldsymbol{x} \in A^*} \left[\mu_{A^*}(\boldsymbol{x}) \otimes \mu_{A^l \to B^l}(\boldsymbol{x}, y) \right]$$

where using one of the engineering implications (i.e., a *t-norm*) (1.8) or (1.9):

$$\mu_{B^{l*}}(y) = \sup_{\boldsymbol{x} \in A^*} \left[\mu_{A^*}(\boldsymbol{x}) \otimes \mu_{A^l}(\boldsymbol{x}) \otimes \mu_{B^l}(y) \right] = \mu_{B^l}(y) \otimes \sup_{\boldsymbol{x} \in A^*} \left[\mu_{A^*}(\boldsymbol{x}) \otimes \mu_{A^l}(\boldsymbol{x}) \right] \quad (1.14)$$

using a *t-norm* as the and connector for antecedents and denoting by $\boldsymbol{x}_i$ the observation corresponding to A^* we have

$$\begin{aligned} \mu_{A^*}(\boldsymbol{x}) &= \mu_{x_{i1}}(x_1) \otimes \mu_{x_{i2}}(x_2) \otimes \ldots \otimes \mu_{x_{in}}(x_n) \\ \mu_{A^l}(\boldsymbol{x}) &= \mu_{A_{1l}}(x_1) \otimes \mu_{A_{2l}}(x_2) \otimes \ldots \otimes \mu_{A_{nl}}(x_n) \end{aligned} \quad (1.15)$$

Substituting (1.15) in (1.14) and rearranging we obtain

$$\mu_{B^{l*}}(y) = \mu_{B^l}(y) \otimes \sup_{\boldsymbol{x} \in A^*} \left\{ \begin{array}{l} \left[\mu_{x_{i1}}(x_1) \otimes \mu_{A_{1l}}(x_1) \right] \otimes \\ \left[\mu_{x_{i2}}(x_2) \otimes \mu_{A_{2l}}(x_2) \right] \otimes \ldots \otimes \\ \left[\mu_{x_{in}}(x_n) \otimes \mu_{A_{nl}}(x_n) \right] \end{array} \right\} \quad (1.16)$$

Recalling that the $\mu_{x_{ij}}$ in (1.16) are singletons, that unity is neutral with respect to any *t-norm*, and thus that the argument of the sup operator is independent of $\boldsymbol{x}$, allows us to ignore the sup operation (maybe one of the best reasons for the success of singleton fuzzification), and Equation (1.16) greatly simplifies in

$$\mu_{B^{l*}}(y) = \mu_{B^l}(y) \otimes \mu_{A_{1l}}(x_{i1}) \otimes \mu_{A_{2l}}(x_{i2}) \otimes \ldots \otimes \mu_{A_{nl}}(x_{in}) \quad (1.17)$$

Equation (1.17) is the final expression for the membership function of the fuzzy set output by the *l*-th fuzzy rule when an engineering implication operator is used along with singleton fuzzification.

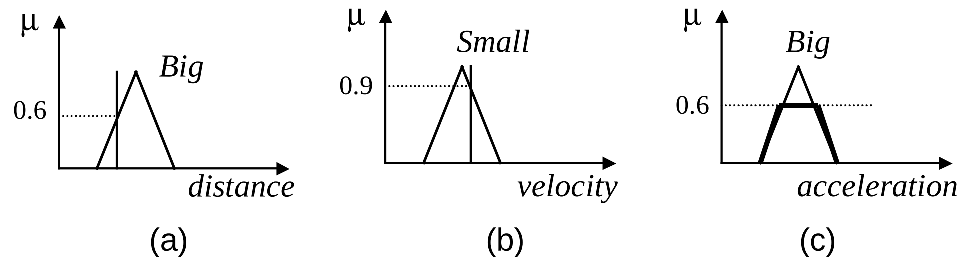

Figure 1.8. Example of inference

<u>Example 9.</u> *Inference mechanics.* Suppose we desire to control the acceleration of a moving object (e.g., car, train, missile) in order to reach a goal position. Let us also suppose that the measurements available in order to decide acceleration values at every time instant are the distance of the moving object from the target position and its velocity. One possible rule in the rule-base of a fuzzy logic system could be: IF *distance* is *big* AND *velocity* is *small* THEN *acceleration* is *big*. At a given time a distance and velocity measurement are available. Say they are fuzzified using a singleton fuzzifier. The fuzzified distance measurement is intersected with the *big* distance membership function in order to compute the first term inside the sup of Equation (1.16). In this particular case (singleton fuzzification) the result is the same regardless of the *t-norm* adopted. Let us say the situation is the one depicted in Fig. 1.8 (a) and that therefore this processing yields the result 0.6 as shown. Analogously the velocity measurement is fuzzified and intersected with the rule antecedent *small* yielding a result of 0.9 as illustrated in Fig. 1.8 (b). These results need now to be intersected inside the sup as in Equation (1.16), therefore yielding: $0.6 \otimes 0.9 = 0.6$ if a minimum *t-norm* is selected. We are now left with the processing of the output of the rule, that is, we need to intersect the results of the sup with the membership function of the consequent (i.e., *big acceleration*) as in Equation (1.16). This step is illustrated in Fig. 1.8 (c) where, once again, a min *t-norm* is employed. This yields the output fuzzy set depicted in bold in Fig. 1.8 (c).

Applying (1.16) or (1.17) to each of the *R* rules in the rule base yields a fuzzy set output for each one of the rules. These *R* fuzzy sets ($\mu_{B^{l*}}$) need to be connected to generate the total output fuzzy set $\mu_Y(y)$. It might seems reasonable to connect the rules output fuzzy sets using a *t-conorm*, that is to connect them taking the union of the output fuzzy sets, as illustrated in Fig. 1.4. Indeed, if each rule were separated by the OR, ALSO, ELSE connectives this choice of connection would make sense. Mendel [43] has a more thorough discussion on connecting rules. Some of the connectives are also tightly linked with the type of defuzzification used, which is the topic for the next section.

1.5.3 Defuzzifier

At the output of the fuzzy inference there will always be a fuzzy set $\mu_Y(y)$ that is obtained by the composition of the fuzzy sets output by each of the rules using Equation (1.16). In order to be used in the real world, the fuzzy output needs to be interfaced to the *crisp* domain by the defuzzifier.

The output fuzzy set indicates what is the output in fuzzy terms. This fuzzy output will be a membership function that provides the degree of membership of several possible crisp outputs. Thus, the point corresponding to the highest degree of membership in the fuzzy output has to be sought. This operation would correspond to a type of defuzzification, called *max* defuzzification. Unfortunately, in most practical cases the situation is not so simple, since there might be many points having the same maximum degree of membership in the fuzzy output, and an indecision on which one of these points to choose arises. Moreover, choosing the maximum point of the membership function is an operation that discards most of the information contained in the membership function itself.

There is the need for a technique that *summarizes* the information contained in the membership function. The *crisp* output corresponding to a certain fuzzy output set should be a number that takes into account all the points in the support of this fuzzy output, weighing the points with high membership degree more than the ones with small or no membership degree. This corresponds to a center of gravity operation, and it is illustrated through an example in Fig. 1.9. Thus, one widely used defuzzifier is the *centroid* defuzzifier that transforms a fuzzy output set into a number that is the *x*-coordinate of the set's center of gravity. The output of this defuzzifier is a number y_d given by

$$y_d = \frac{\int_S y\mu_Y(y)dy}{\int_S \mu_Y(y)dy} \tag{1.18}$$

where S is the support of $\mu_Y(y)$. One drawback of this kind of defuzzification is the complexity involved with finding the center of gravity (i.e., integration). For this and other reasons (see [6] for a more detailed account of defuzzification approaches), easier defuzzification schemes are generally employed for reduced computational burden.

One of the most popular defuzzifiers is the center of area (COA) defuzzifier (also called height defuzzifier). In this approach the overall center of gravity is approximated by the center of gravity of "point-masses" located at the center of gravity of each individual rule's output fuzzy set, with "mass" equal to the membership degree at that point. This is not an approximation if the supports of the fuzzy sets corresponding to the output of each rule do not overlap and the consequent membership functions are isosceles triangles with equal bases. In the general case of overlap this might be an approximation to the actual center of gravity depending also on the rule connective that is used. Calling δ_l the center of gravity of fuzzy set B^l* output of the l-th rule, the output of the COA defuzzifier is given by

$$y_d = \frac{\sum_{l=1}^{R} \delta_l \mu_{B^{l*}}(\delta_l)}{\sum_{l=1}^{R} \mu_{B^{l*}}(\delta_l)} \tag{1.19}$$

Equation (1.19) is very easy to use since the centers of gravity of commonly used membership functions are known ahead of time. Regardless of the *t-norm* used (minimum or product) the center of gravity for commonly used symmetric membership functions (triangular, Gaussian, trapezoidal, bell shaped) does not change after inference. In other

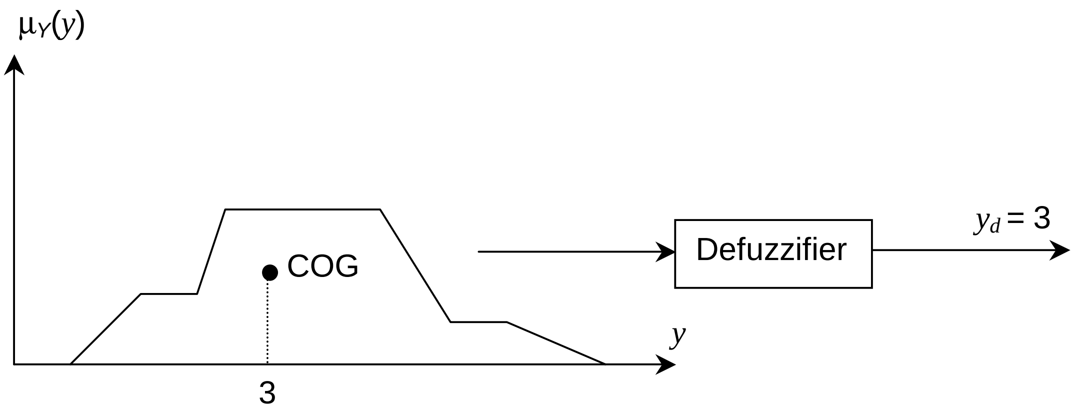

Figure 1.9. Example of defuzzification by centroid defuzzifier

words for commonly used symmetric consequent membership functions, the center of gravity of B^l and B^l* is the same, therefore making the application of (1.19) very easy and thus, appealing.

1.6 Problem Assumptions

In the previous sections fuzzy sets, fuzzy logic and fuzzy logic systems were described along with their principle of operations. Given a fuzzy logic system and an input to the system, its output can be determined using (1.12) or (1.13), (1.16), and (1.18) or (1.19). The problem in this fashion is still too vaguely formulated to be able to analyze and exploit its structure and properties. Moreover, of all the available choices some became more popular in the practice and use of FLSs because of their ease of use and reduced computational burden. More specifically, singleton fuzzification is used most of the time along with center of area defuzzification. In this formulation the specification of the consequents membership functions is not so important as the specification of its center of gravity δ_l, the only parameter that really contributes to the output computation in (1.19). Thus, without any loss of generality we can regard this class of fuzzy systems as having singleton consequent fuzzy sets. Let us now analyze what happens to the system output with these choices.

Using COA defuzzification, and thus, singleton consequent membership functions, Equation (1.19) is further simplified. The output fuzzy set corresponding to the *l*-th rule becomes a singleton with center δ_l:

$$\mu_{B^l}(y) = \begin{cases} 1 & y = \delta_l \\ 0 & y \neq \delta_l \end{cases}$$

Equation (1.17) becomes

$$\mu_{B^l*}(y) = \begin{cases} \mu_{A_{1l}}(x_{i1}) \otimes \mu_{A_{2l}}(x_{i2}) \otimes \ldots \otimes \mu_{A_{nl}}(x_{in}) & y = \delta_l \\ 0 & y \neq \delta_l \end{cases} \tag{1.20}$$

For notational ease we can define

$$\bigotimes_{i=1}^{n} \alpha_i = \alpha_1 \otimes \alpha_2 \otimes \ldots \otimes \alpha_n \tag{1.21}$$

that could be substituted in (1.20) to express the activation strength for rule *l*:

$$\mu_{B^{l*}}(\delta_l) = \bigotimes_{j=1}^{n} \mu_{A_{jl}}(x_{ij}) \tag{1.22}$$

Once all the rules are processed their outcome can be combined and defuzzified. Using COA defuzzification the output of the FLS corresponding to the input $\boldsymbol{x_i}$ will be computed by plugging (1.22) in (1.19), thus, yielding

$$y(\boldsymbol{x_i}) = y(x_{i1}, x_{i2}, \ldots, x_{in}) = \frac{\sum_{l=1}^{R} \delta_l \left[\bigotimes_{j=1}^{n} \mu_{A_{jl}}(x_{ij}) \right]}{\sum_{l=1}^{R} \left[\bigotimes_{j=1}^{n} \mu_{A_{jl}}(x_{ij}) \right]} \tag{1.23}$$

Equation (1.23) more precisely describes the nonlinear input-output mapping implemented by a FLS with the common choices of singleton fuzzification and center of area defuzzification. This equation is still not operational since a choice for antecedent membership functions and *t-norm* needs to be made.

Before proceeding any further we also need to complicate the notation a little to have a more precise FLS description. Indeed, in (1.23) the terms $\mu_{A_{jl}}(x_{ij})$ correspond to the membership function of fuzzy set A_{jl}. Fuzzy set A_{jl} is the fuzzy set corresponding to the *j*-th input variable and for the *l*-th rule. In general, for each input (i.e., linguistic variable) the universe of discourse is partitioned in fuzzy sets (e.g., small, medium, large) that could correspond to numeric indices (e.g., respectively 1,2,3). Thus the second index *l* in A_{jl} is not really appropriate since it should really be an index depending on *l* and with values in the set of numbers describing the partition of the input space. In more formal terms, if the *j*-th input is partitioned in K_j membership functions each of one uniquely identifiable with an integer between 1 and K_j then the fuzzy set for the *j*-th input in the *l*-th rule should be $A_{jk(j,l)}$ where $k(j,l)$ is a function $k : \{1,2, \ldots,n\}\times\{1,2,\ldots, R\} \rightarrow \aleph$, where $\aleph$ is the set of integers. More specifically $1 \leq k(j,l) \leq K_j$. Moreover, we can ease the notation if we denote by $\mu_{ij}(x)$ the membership function for A_{ij}. The same discussion holds for the consequent part of the FLS. In this case we define the function $h(l)$, $h : \{1,2,\ldots, R\} \rightarrow \{1,2,\ldots, H\}$, where H is the number of membership functions defined for the consequent. Note that the functions $k(j,l)$

and $h(l)$ univocally describe the rule base. With this modified and more precise notation (1.23) becomes

$$y(\boldsymbol{x_i}) = y(x_{i1}, x_{i2}, \ldots, x_{in}) = \frac{\sum_{l=1}^{R} \delta_{h(l)} \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}) \right]}{\sum_{l=1}^{R} \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}) \right]} \tag{1.24}$$

In (1.24) the dependence from some adjustable parameters is not clearly shown. The adjustable parameters ***w*** that we choose to consider for this problem (as is customarily done) are the parameters defining the antecedent membership functions, $\boldsymbol{w_a}$, and the parameters defining the consequents functions, $\boldsymbol{w_c}$. Note that we could consider $\boldsymbol{w_c}$ to merely be the set of parameters $\delta_{h(l)}$ but we will not do so yet, in order to still leave some degrees of freedom in the parameterization (e.g., to later consider more general Takagi-Sugeno models [77], see next section). Thus, we can include the dependence from ***w*** in (1.24), rewriting it as

$$y(\boldsymbol{x_i}, \boldsymbol{w}) = y(x_{i1}, x_{i2}, \ldots, x_{in}, \boldsymbol{w_a}, \boldsymbol{w_c}) = \frac{\sum_{l=1}^{R} \delta_{h(l)}(\boldsymbol{w_c}) \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}, \boldsymbol{w_a}) \right]}{\sum_{l=1}^{R} \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}, \boldsymbol{w_a}) \right]} \tag{1.25}$$

Let us now explain this notation with an example.

<u>Example 10</u>. *Notation to describe a FLS.* Consider a two inputs and one output FLS with three partitions for each of the inputs and the rule base shown in Table 1.2. Rules are generally expressed in the form of a table like the one shown here. The rows and columns of the table correspond to the antecedent part of the rule, while a cell corresponds to the consequent corresponding to that combination of antecedents. For example, corresponding to the term N for x_1 and P for x_2 we read zero, that is, the corresponding rule is: IF u_1 is N and u_2 is P THEN y is ZE. With the notation that we introduced, this FLS is described by: $n = 2$ (inputs), $K_1 = K_2 = 3$ (membership functions per input), $H = 5$ (consequent membership functions), $R = 9$ (rules), $h = \{1,2,3,2,3,4,3,4,5\}$, and $k = \{1,2,3,1,2,3,1,2,3; 1,1,1,2,2,2,3,3,3\}$.

Table 1.2. Rule base for Example 10

$x_1 \rightarrow$ / $x_2 \downarrow$	N [A_{11}]	Z [A_{12}]	P [A_{13}]
N [A_{21}]	NL [δ_1]	NS [δ_2]	PS [δ_3]
Z [A_{22}]	NS [δ_2]	ZE [δ_3]	PS [δ_4]
P [A_{23}]	ZE [δ_3]	PS [δ_4]	PL [δ_5]

1.7 Takagi-Sugeno Fuzzy Logic Systems

The fuzzy logic systems described in the above sections are commonly referred to as *pure* fuzzy logic systems [16] or Mamdani fuzzy logic systems [40,41]. An alternative to these FLSs is offered by Takagi-Sugeno (TS) fuzzy logic systems [77]. In a TS-FLS the consequent of each rule is not a fuzzy set but it is a local model of the system (function) to be controlled (approximated). Thus the l-th rule of a TS-FLS has the form:

$R^{(l)}$: IF u_1 is $A_{1k(1,l)}$ and u_2 is $A_{2\,k(2,l)}$ and … u_n is $A_{n\,k(n,l)}$, THEN $y = g_l(x_1, x_2, \ldots, x_n)$

The idea of this approach is to fit enough local models g_l to adequately describe the system while the fuzzy inference provides for some smooth interpolation between models. These local models are generally linear ([77] and all the following papers); thus the l-th rule of a TS-FLS becomes:

$R^{(l)}$: IF u_1 is $A_{1k(1,l)}$ and u_2 is $A_{2\,k(2,l)}$ and … u_n is $A_{n\,k(n,l)}$,

THEN $y = p_{0l} + p_{1l}x_1 + p_{2l}x_2 + \ldots + p_{nl}x_n$

Under the same common assumptions for implication, the output of a such a FLS can be computed the same way as the output of a pure FLS, with the difference that the terms

$\delta_{h(l)}(\boldsymbol{w_c})$ in (1.25) are not constant but are given by the consequent part of the TS rules (i.e., the local models):

$$\delta_{h(l)}(\boldsymbol{w}_c) = p_{0h(l)} + p_{1h(l)}x_1 + p_{2h(l)}x_2 + \ldots + p_{nh(l)}x_n \tag{1.26}$$

Thus, the output of a TS-FLS is completely described by (1.25) and (1.26). Note how a pure FLS with COA defuzzification can be regarded as a TS-FLS with constant consequents (local models). Note also how in the original formulation of a TS-FLS all the local models are in general different, thus $h(l) = l$, in order to account for one local model per rule. On the other hand, this can be achieved in the same framework of the formulation previously presented, with $H = R$ (number of different consequent parts = number of rules) and $h(l) = l$. The output of a TS-FLS and the output of a pure FLS share the same property, they are linear in the consequent parameters. As we will see this is sometimes exploited in the optimization of fuzzy systems, which is the topic for the subsequent chapters.

1.8 Conclusions

In this chapter we introduced fuzzy sets, fuzzy logic and fuzzy logic systems. Their principle of operation was described to finally reach a formulation for the FLS output. We adopted the common assumptions of singleton fuzzification, and center of average defuzzification, while the inference process is still generically defined by a *t-norm* ($\otimes$). Equation (1.25) shows the output of a FLS with the assumptions specified in Section 1.6 above; from this equation it is clear that a FLS is a parametric nonlinear mapping between inputs and output. The adjustable parameters of a FLS change the antecedent and consequent membership functions making FLSs powerful function approximators. Finally we also presented a variation on the standard FLSs, TS-FLS. We also saw how a TS-FLS does not really represent anything too different from the FLS described before, thus allowing the same output representation of (1.25) with the additional constraint given by (1.26). A glossary of some of the quantities introduced in this chapter may be found in the front part of the dissertation.

Chapter 2
Introduction to Design Optimization of Fuzzy Logic Systems

2.1 Introduction

Chapter 1 introduced fuzzy logic systems, and showed how they easily merge a powerful nonlinear static mapping (from the mathematical point of view) as shown in Equation (1.25), with a rule-base controller of easy interpretation and setup (from a practical, application-oriented point of view). The latter vision has been one of the main drives for the wide spread of fuzzy logic systems applications. Following the publication of Zadeh's papers [88,89], as an example of practical application Mamdani set up a controller for a model industrial plant (steam engine and boiler combination) [41]. Through some simple identification tests the plant proved to be highly nonlinear, thus possessing noticeably different characteristics at different operating points. Using a classical proportional-integral-derivative (PID) digital control approach, the controller had to be retuned (by trial-and-error) every time the operating point was changed. The plant seemed to be easily controllable, though, through a few intuitive "linguistic" rules. This stimulated an experiment on the linguistic synthesis of a controller [41]. Fuzzy logic was used to convert heuristic control rules, stated by a human operator, into an automatic control strategy that proved to be far better than expected. After this first pioneering application of FLSs many followed, first in Europe (control of a kiln for cement production [34]) and later in Japan. Finally, things also caught up in the USA, where everything had started. A more thorough account of the successful applications of fuzzy logic can be found in [35]. A common denominator of all the applications of fuzzy logic was the existence of known practical approaches to the control problem. Indeed, in most of these problems there was an expert that was successful in controlling the plant and that was used as the basis in implementing a strategy in an automatic fashion. Fuzzy logic was the perfect tool for acquiring the knowledge of an expert and embedding it in a systematic and sound mathematical framework. Moreover, when an expert was not available, some easy and intuitive control rules could be stated by an understanding of the first principles at the basis of the system's functioning. The general design approach for a FLS was based on understanding the (human) expert approach to solving a control problem, implementing the strategy by direct translation of linguistic control rules, and testing the developed FLS. This process will lead to a satisfactory FLS design, but in general a sub-optimal one. The parameters of such a heuristic FLS design can be further adjusted, such that some opportunely defined performance measure is maximized. The general approach to this successive tuning has been by trial-and-error, a very common engineering practice that always yields some good practical results, but that offers no guarantees of optimality and no automatization of its evolution (i.e., man-time is "wasted" in a process that could be generally made automatic).

In this setting the next step is quite obvious. Quite some emphasis of the early 90's research in FLSs was put on their design optimization, in order to achieve some design optimization approach instead of a merely human driven trial-and-error process. The design optimization of a FLS can be logically partitioned into structural and parametric design. Indeed, remembering the principles of operation of a FLS, many details about a FLS need to be fixed in order to determine a particular class of FLSs that are still differentiated through the set of parameters that define them. In this sense the structural learning or design of a FLS is characterized by the choice of fuzzification, *t-norm*, inference, defuzzification, number of inputs, type and number of membership functions used for each input and output, rule base (i.e., number of rules and rules). The parametric learning (or design) problem is a parametric problem, whereby the numeric value of the parameters defining a particular FLS design needs to be fixed in order to maximize a given performance metric. This problem can be regarded as an optimization problem as we shall see in the following section.

The approaches to design optimization of a FLS tried to tackle the structure and parameter learning in two ways:

1. Iteratively performing structural and parametric learning in an alternating fashion, until convergence is reached;
2. Performing structural and parametric learning at the same time in order to reach the true optimum for the problem (the first approach might not converge to such an optimum).

While the structural learning approaches are generally heuristic, the parametric learning approaches tend to be cast in the form of optimization problems in general solved by gradient descent or variations thereof. In the following we will concentrate on the parameter-learning problem. The structure learning problem is generally solved by heuristics; it seems more intuitive to approach it on a problem-by-problem basis where, either through the help of experts or by our own understanding, we can define the necessary number of membership functions etc. Moreover, since design simplicity is always desired, we can always start with a simple structure and try to further understand where the trade-off between performance and simplicity can be set. Different *t-norms* and inferences can be tried, and an understanding of the best one for the problem at hand can be achieved. The parameter-learning problem is probably the most time consuming, thus some trial-and-error can be left with the structure learning process. Furthermore, techniques developed for parametric learning could be integrated with the existing structural learning methods. Finally, the parametric learning problem seems to be a good start for the problem of optimizing a given design.

The following Section 2.2 introduces the supervised learning problem. The fuzzy to neuro-fuzzy “leap” that stimulated quite some work in the training of FLSs is then discussed in Section 2.3 along with two different types of FLSs. Section 2.4 derives the final gradient-descent optimization algorithms for both types of FLS. Finally, Section 2.5 reviews the different approaches to the supervised learning problem presented in the literature, and Section 2.6 offers a comprehensive and synthetic discussion of the problem issues.

2.2 The Supervised Learning Problem

Parameter learning for many control, system identification, adaptive control and classification problems can be reduced to a function approximation problem where given a function we want to adjust the FLS parameters as to best approximate it. Thus, a *teacher* (i.e., function samples) is always available to *supervise* the learning; hence the name supervised learning (sometimes self-learning, tuning).

Some classes of FLSs have been proven to be universal approximators, that is, given a smooth function there always exists a FLS with appropriate structure that can approximate this function with arbitrary accuracy. This is more formally stated by the universal approximation theorem.

Universal Approximation Theorem. For any given real continuous function g on a compact set $U \subset \Re^n$ and arbitrary $\varepsilon > 0$, there exists a fuzzy logic system implementing a mapping f: $U \rightarrow \Re$ such that

$$\sup_{x \in U} \left| f(x) - g(x) \right| \leq \varepsilon$$

Several versions of this theorem have been proved for different types of FLSs [19,26,28,32,33,84,85] (among others), however no proof of this theorem for an arbitrary FLS has been generated yet [43]. Unfortunately, these theorems are not much help to the practice of a FLS designer since they are not constructive, i.e., they do not provide any direction of how to build such a FLS. These theorems give the motivation to undertake any function approximation problem with the assurance that if the structure of the FLS is appropriate (i.e., big enough), the function approximation problem can be tackled by the FLS. In the following we will restrict our attention to a multi-input single-output (MISO) problem since any multi-output (MIMO) problem can be approached as the parallel of MISO solutions.

The supervised learning problem can be described as follows. Given N function samples $(\boldsymbol{x}_i, y_{di})$ $i = 1,2, \ldots, N$ and $\boldsymbol{x}_\mathrm{i} \in \Re^\mathrm{n}$ (where the sub-index d in y_{di} stands for "desired" output, and the sub-index i corresponds to the datum number), we want to adjust the parameters of a given FLS in order to approximate the given samples with the least error. What is really desired is to approximate the underlying (unknown) function that produced the samples. The ability of doing so is commonly referred to as *generalization*. Partitioning the available data into a training set and a test set commonly tests generalization. The training set is used for parameter training purposes; and, once a goal performance measure value is achieved, the corresponding FLS approximation error on the test data is measured. It has been frequently observed with neural networks that a minimum training error does not necessarily correspond to a minimum test error, or best generalization (see [20] for some examples and references). This can be explained by several considerations. First of all the data used for training might be noisy (especially if they are obtained by some measurement process) and a very small training error would correspond to having learned both the data and the noise model. Obviously this is not desired, since the real objective of the learning process is to model the underlying (noiseless) system (function). Moreover, if the structure is too big (i.e., there are too many parameters) the system will be trying to exactly approximate the data set in an interpolatory way (i.e., overfitting). On the other hand what is really desired is to approximate the given data in a least square sense since this should help to average the noise out of the model. In my opinion this can be generally achieved by using small structures and optimizing them for minimum error; testing them, and eventually increasing the structure size in case of dissatisfaction with the performance of the current structure. In this work we will not consider generalization, we will only think at approximating the given function samples.

It was shown in Chapter 1 that a FLS could be regarded as a nonlinear parametric mapping between input and output; we can express it as $y = f(\boldsymbol{x}, \boldsymbol{w})$ where y is the scalar FLS output, $\boldsymbol{x}$ is the n-dimensional input vector and $\boldsymbol{w}$ is the p-dimensional vector containing all the FLS's adjustable parameters. For our particular problem assumptions (singleton fuzzification, singleton output membership functions or TS model, and COA defuzzification) the function f is given by (1.25) eventually accompanied by (1.26) in case of TS fuzzy models. It is desired to adjust $\boldsymbol{w}$ so that f best approximates the data samples. In general the concept of "best approximation" is expressed through a mean square error

(MSE) between approximated and desired data, even though we are not restricted to this type of error formulation. Thus we can define a cost, or error, function as

$$E(\boldsymbol{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left[y(\boldsymbol{x}_i, \boldsymbol{w}) - y_{di}\right]^2 \tag{2.1}$$

This function expresses the MSE in approximating the data samples. Thus, the optimization of a FLS can be stated as finding the parameters $\boldsymbol{w}$ that minimize $E(\boldsymbol{w})$. Note that we can define some *instantaneous* approximation errors

$$E_i(\boldsymbol{w}) = \frac{1}{2}\left[y(\boldsymbol{x}_i, \boldsymbol{w}) - y_{di}\right]^2 \quad i = 1,2,\ldots,N \tag{2.2}$$

and thus rewrite the error function $E(\boldsymbol{w})$ as the average of the instantaneous errors

$$E(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N} E_i(\boldsymbol{w}) \tag{2.3}$$

A common approach in the FLS and neural network literature, motivated by ease of computation, consists of presenting a sample datum, say the i-th, to the FLS (or neural network) and updating the adjustable parameters in order to only minimize $E_i(\boldsymbol{w})$. By constantly repeating this process while alternating the presentation of all the data and using small updates, it is hoped to minimize the whole $E(\boldsymbol{w})$. This procedure is a type of stochastic gradient algorithm generally named *pattern-by-pattern* (or on-line) training, to suggest the one-sample-at-a-time approach; as opposed to the *batch* (or off-line) training mode where the samples are presented to the network as a batch and the correction of the adjustable parameters is applied only at the end of the batch. The presentation of the entire set of data to the FLS is generally referred to as an *epoch*. Thus, in batch training the adjustable parameters are updated once every epoch, while in pattern-by-pattern training they are updated N times every epoch. The pattern-by-pattern approach is an approximation to the minimization of (2.1) and it has some rigorous theoretical foundations in the stochastic gradient approximation (SGA) [20,66].

We are now going to develop the problem formulation for both the pattern-by-pattern and the batch training mode. The supervised learning problem can be formulated as

$$\min_{\boldsymbol{w}} E(\boldsymbol{w}) \tag{2.4}$$

where the adjustable parameters ***w*** are unconstrained and $E(\boldsymbol{w})$ is given by (2.2) or (2.3) for pattern-by-pattern or batch training, respectively. Using (1.25) we can rewrite (2.2) as

$$E_i(\boldsymbol{w}) = E_i(\boldsymbol{w}_a, \boldsymbol{w}_c) = \left\{ \frac{\sum_{l=1}^{R} \delta_{h(l)}(\boldsymbol{w}_c) \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}, \boldsymbol{w}_a) \right]}{\sum_{l=1}^{R} \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}, \boldsymbol{w}_a) \right]} - y_{di} \right\}^2 \quad i = 1,2,...N \tag{2.5}$$

In batch training the parameters ***w*** are updated by gradient descent according to

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \eta \left. \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} \right|_{\boldsymbol{w}=\boldsymbol{w}^{old}} \tag{2.6}$$

where η is called the learning rate (i.e., the step-size) and is generally a small positive constant. In the case of pattern-by-pattern training the parameter updates are obtained by

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \eta \left. \frac{\partial E_i(\boldsymbol{w})}{\partial \boldsymbol{w}} \right|_{\boldsymbol{w}=\boldsymbol{w}^{old}} \quad i = 1,2,...N \tag{2.7}$$

where the order in which they are applied is generally randomized. That is, training points are generally shuffled between epochs so that they are always presented to the FLS in a different order. Using Equation (2.3) the update for ***w*** in batch mode can be computed as

$$\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial E_i(\boldsymbol{w})}{\partial \boldsymbol{w}} \tag{2.8}$$

Comparing Equations (2.6) and (2.7) using (2.8) we can understand the difference between batch and pattern-by-pattern training. In batch mode, the adjustable parameters are “frozen” and all the sensitivities of the instantaneous errors are calculated, averaged, and finally applied, once per epoch. Conversely, in the pattern-by-pattern training mode the same sensitivities are computed but the parameters are instantly updated thus yielding N updates per epoch and a different algorithmic path. The sensitivities of interest (i.e., partial derivative of the instantaneous error function with respect to the adjustable parameters) can be computed by chain-rule derivation of Equation (2.2), thus, yielding

$$\frac{\partial E_i(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{\partial E_i}{\partial y} \frac{\partial y}{\partial \boldsymbol{w}} = [y(\boldsymbol{x_i}, \boldsymbol{w}) - y_{di}] \frac{\partial y}{\partial \boldsymbol{w}} \tag{2.9}$$

Substituting Equation (2.9) in (2.8) and then in (2.6) we obtain the batch training update equation

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \frac{\eta}{N}\sum_{i=1}^{N}\left[y(\boldsymbol{x_i},\boldsymbol{w}) - y_{di}\right]\left.\frac{\partial y}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\boldsymbol{w}^{old}} \tag{2.10}$$

The update equations for the pattern-by-pattern training are obtained by substitution of (2.9) in (2.7) and yield

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} - \eta\left[y(\boldsymbol{x_i},\boldsymbol{w}) - y_{di}\right]\left.\frac{\partial y}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\boldsymbol{w}^{old}} \qquad i = 1,2,\ldots N \tag{2.11}$$

The pattern-by-pattern approach expressed by (2.11) is probably the most common approach used and presented in the literature. A common step for applying either pattern-by-pattern or batch training is the computation of the partial derivative of the output of the FLS with respect to the adjustable parameters (i.e., $\partial y/\partial \boldsymbol{w}$). Its computation will be illustrated in the following Section 2.4 for two different types of FLSs that will be introduced in Section 2.3 along with a few historical notes on the supervised learning problem.

Using singleton consequent membership functions or TS models along with COA defuzzification the output of the FLS is linear in the consequent parameters (as can be seen from (1.25)), once the antecedent parameters are fixed. Thus, least square approaches can be developed to solve this problem alone to (global) optimality.

2.3 From Fuzzy to Neuro-Fuzzy

Takagi and Sugeno [77] proposed a new format of fuzzy reasoning, as explained in Section 1.7, where the consequent is a function constituting a local model. In their approach they also started looking at the fuzzy identification problem and developed an identification procedure for both structure and parameter learning. A simple structure is first selected and its parameters are chosen by alternatively and repeatedly finding least square estimates of the consequent parameters and optimized antecedent parameters through a heuristic that they call the complex method. In this approach constraints are given for the parameter change, and parameter values are tested at the boundary of the constraint set and the best parameter values are retained. The structure is subsequently enlarged until no significant improvement (or a predefined error goal) is achieved and thus the procedure is stopped. This contribution, dated 1985, can be considered one of the first impulses towards fuzzy identification or optimized design of FLSs. This technique makes it possible to tune a FLS based on existing input-output data, but it offers no guarantees of convergence (since heuristics are used) to either a local or a global minimum, or even to a point of zero gradient.

In the early '90s some researchers started looking at FLSs as adaptive networks (i.e., Adaptive Network Based Fuzzy Inference System, ANFIS [26,27,28]; Fuzzy Neural Network, FNN [37]; Simplified Fuzzy Inference Network, SFIN [80]). These approaches to FLSs generate what are called neuro-fuzzy systems that, in the view of all the different authors and subsequent users, bring together the ease of – linguistic – interpretation and maintenance of FLSs, with the computational power of neural networks that can be trained through a gradient-type similar to the back-propagation (BP) algorithm. The BP algorithm is a gradient descent algorithm in which the derivatives of an objective function (generally an approximation error in the form of (2.1) or (2.2)) with respect to the parameters are calculated by the chain derivative rule. A first forward pass is performed to determine the network output and a second backward pass is performed to adjust the parameters for better approximation in a pattern-by-pattern fashion. In the second pass gradient information is computed through the quantities calculated in the forward pass, originating some computational savings. Thus, the algorithms developed for supervised learning or tuning of FLSs adjust the parameters based on gradient information.

The adaptive-network view of FLSs is definitely useful in translating many existing approaches from the neural network field to the fuzzy system field, as well as to generate hybrid approaches, comparisons etc. On the other hand, the literature has the constant reference to the back-propagation as if it were *the* algorithm that allowed the user to tune FLSs. As shown in the previous Section 2.2, supervised learning of FLSs is an optimization problem and, as such, it can be solved, or be attempted to solve, with many existing optimization approaches that are *not* limited to gradient descent. Gradient descent is one of those approaches that is sometimes preferable due to its ease of implementation, low storage requirements, etc. It is gradient-descent-based optimization that allows us to easily execute supervised learning of FLSs, and *not* their analogies to neural networks. In many instances this does not seem to be clearly understood, or at least stated. The merit of the back-propagation algorithm is to devise an efficient computational approach for the determination of the gradients necessary for updating all the (many) network parameters. This aspect lies at the implementation end however, and has nothing to do with the formulation and solution of the optimization problem.

Supervised learning of FLSs with antecedent and consequent adjustable parameters is a nonlinear programming problem with its own characteristics, that many times are different from those of neural network optimization problems (e.g., non-differentiable membership functions). The nonlinear nature of the learning problem makes it such that many approaches that were presented in the literature were aimed at only optimizing the consequent parameters for pure FLSs with singleton consequents or for TS fuzzy models. This constitutes a linear problem and, as such, can be approached through least squares and recursive least squares approaches, that are well known and robust parameter estimation approaches. The main reason for this type of approach consists of the latter statement, that is, the techniques that can be used to attack this problem are well known and robust. Moreover, some authors mention a problem of interpretability or readability of the tuned fuzzy system whenever the antecedent membership functions are adapted. This readability problem will be discussed in Section 2.6. From a function approximation point of view it is intuitively important to tune the antecedent parameters in order to increase the performance of the FLS. Moreover, the membership functions generally come from the evaluation of experts and will be probably sub-optimal.

Finally, some authors present integrated structure and parameter learning approaches in which the structural learning part includes setting the number and position of membership functions on each input (sometimes only that), and the parameter learning consists of tuning the consequents via least square estimation. Generally, the structural learning part is achieved through heuristics mostly based on clustering approaches. No proof or indication of how good these heuristics really are in general cases is given. For this reason their value consists of, perhaps giving a good initialization for parameter learning approaches, but not of directly defining the final (best) values for the input membership functions. From this perspective neuro-fuzzy approaches helped the state of the art to evolve in the direction of optimizing both antecedent and consequent parameters, and not only consequent parameters. Indeed, the common idea that emerges from the literature is that by using the BP algorithm we can tune the antecedent parameters as easily as we can tune the consequent ones. It is gradient-descent that allows tuning of the antecedent parameters, however the neuro-fuzzy approach has at least the merit of having motivated people to transition to tuning antecedent parameters as well.

Among the many existing neuro-fuzzy approaches we can recognize two main classes. In the first class of approaches classical FLSs as presented in Chapter 1 are adjusted, while in the second class the FLS has one distinct antecedent fuzzy set (membership function) per rule. The difference between the two approaches will be now more clearly explained. The l-th rule of a FLS has the following form:

$$R^{(l)}\text{: IF } u_1 \text{ is } A_{1l} \text{ and } u_2 \text{ is } A_{2l} \text{ and } \dots\ u_n \text{ is } A_{nl}\text{ , THEN } v \text{ is } B^{l}$$

With the rules in this format the fuzzy set appearing in the l-th rule (A_{jl}) is dependent only on the rule, thus every rule has its own membership function. But, in a traditional FLS the input space is partitioned into linguistic variables (e.g., small, medium and large) and these linguistic variables are used and combined to produce the rules. Thus, a specific membership function will likely appear in more then one rule. Thus, to properly represent a conventional FLS the term A_{jl} should be substituted by $A_{jk(j,l)}$. This was already discussed in Section 1.6 to transform (1.23) to (1.24). Thus, the correct representation of the l-th rule of a conventional fuzzy system is:

$$R^{(l)}\text{: IF } u_1 \text{ is } A_{1k(1,l)} \text{ and } u_2 \text{ is } A_{2k(2,l)} \text{ and } \dots\ u_n \text{ is } A_{nk(n,l)}\text{ , THEN } v \text{ is } B^{h(l)}$$

The two different classes of approaches described above implement either a conventional FLS (as in the rule above) or a FLS with different membership functions (i.e., fuzzy set) for each rule. In the following we will denote an FLS with independent membership functions in each rule by IMF-FLS, as opposed to a conventional FLS denoted by FLS. Let us illustrate this concept through an example continued from Example 9 of Chapter 1.

<u>Example 10.</u> *FLS and IMF-FLS.* Let us consider again Example 9 where we were deciding the acceleration to impart to a car in order to reach a target position, given distance and velocity measurements. Two possible rules for such a control system could be:

IF *distance* is *big* AND *velocity* is *small* THEN *acceleration* is *big* (*i*)

IF *distance* is *big* AND *velocity* is *big* THEN *acceleration* is *medium* (*ii*)

In a FLS the fuzzy set *big* defined on the linguistic variable *distance* is the same for both rules (*i*) and (*ii*). For example, it could be a triangular membership function centered at 10 m. Therefore, while tuning the center of this membership function, we need to account for the fact that the variation of this parameter affects the firing of both rules, and thus, we need to account for this, for example, in deriving gradient descent update rules. This aspect obviously complicates the derivation of parameter update equations. In an IMF-FLS the fuzzy set *big* has a different meaning in rules (*i*) and (*ii*). Thus, in the first rule it could be a triangular membership function centered at 10 m, while in the second rule it might be a triangular membership function centered at 12 m. Even though the fuzzy sets could be the same at the beginning of the training algorithm (same initialization), they are allowed to depart from each other during the evolution of the training. Obviously, this greatly simplifies the derivation of parameter update equations for an IMF-FLS since each parameter affects the firing of only one rule. The same considerations hold for the consequent part of the rules. In an IMF-FLS each consequent can have the same linguistic label, but might have different meanings in different rules.

Note that a (conventional) FLS results in the most general approach. Indeed, an IMF-FLS can be obtained from an FLS by imposing

$$k(j,l) = l,\ \ h(l) = l,\ \ K_j = R,\ \ H = R \quad \forall j \in \{1,2,\ldots,n\} \quad \forall l \in \{1,2,\ldots,R\} \qquad (2.12)$$

in which R is the number of rules of the IMF-FLS and the other quantities have been defined in Section 1.6 and can be reviewed in the glossary.

The optimization approaches for the non-conventional case (IMF-FLS) are definitely easier to derive algebraically than their FLS counterpart. In the following Section 2.4 we will describe the analytical derivation of the gradient components for IMF-FLSs first and for FLSs later. A review of the approaches presented in the literature will follow in Section 2.5.

2.4 Supervised Learning Formulation

In this section we describe the complete formulation of pattern-by-pattern and batch training by gradient descent for both FLS and IMF-FLS. This is accomplished by employing the equations derived in the preceding Section 2.2 and computing the partial derivative of the output of the FLS (or IMF-FLS) with respect to the adjustable parameters (i.e., $\partial y/\partial \boldsymbol{w}$).

2.4.1 Supervised Learning Formulation for an IMF-FLS

In the case of an IMF-FLS the output of the system is given by (1.23). With the understanding that $\mu_{jl}(x_{ij})$ is the membership function for the fuzzy set A_{jl}, and highlighting the dependence from antecedent and consequent parameters ($\boldsymbol{w_a}$ and $\boldsymbol{w_c}$ respectively) (1.23) becomes

$$y(\boldsymbol{x_i},\boldsymbol{w}) = y(x_{i1}, x_{i2}, \ldots, x_{in}, \boldsymbol{w_a}, \boldsymbol{w_c}) = \frac{\sum_{l=1}^{R} \delta_l(\boldsymbol{w_c}) \left[\bigotimes_{j=1}^{n} \mu_{jl}(x_{ij}, \boldsymbol{w_a}) \right]}{\sum_{l=1}^{R} \left[\bigotimes_{j=1}^{n} \mu_{jl}(x_{ij}, \boldsymbol{w_a}) \right]} \tag{2.13}$$

Defining the *firing strength* (or simply *strength*) of rule *l* as

$$s_l(\boldsymbol{x},\boldsymbol{w_a}) = \bigotimes_{j=1}^{n} \mu_{jl}(x_j, \boldsymbol{w_a}) \tag{2.14}$$

we can rewrite (2.13) as

$$y(\boldsymbol{x},\boldsymbol{w}) = \frac{\sum_{l=1}^{R} \delta_l(\boldsymbol{w_c}) s_l(\boldsymbol{x},\boldsymbol{w_a})}{\sum_{l=1}^{R} s_l(\boldsymbol{x},\boldsymbol{w_a})} \tag{2.15}$$

Thus, the sensitivities of the fuzzy system output with respect to the consequent parameters are easily computed as

$$\frac{\partial y}{\partial \boldsymbol{w_c}} = \frac{\sum_{l=1}^{R} \frac{\partial \delta_l(\boldsymbol{w_c})}{\partial \boldsymbol{w_c}} s_l(\boldsymbol{x}, \boldsymbol{w_a})}{\sum_{l=1}^{R} s_l(\boldsymbol{x}, \boldsymbol{w_a})} \tag{2.16}$$

Now we can use the idea intrinsic in the IMF-FLS that every rule has its own independent fuzzy set. Thus, if we denote with $\boldsymbol{w_{cl}}$ the set of consequent parameters appearing in the *l*-th rule (and only in the *l*-th rule), those parameters will only influence the output of the *l*-th rule, that is

$$\frac{\partial \delta_i(\boldsymbol{w_c})}{\partial \boldsymbol{w_{cj}}} = 0 \ \Leftrightarrow i \neq j, \ \ i \in \{1,2,...R\} \ \ \ j \in \{1,2,...R\} \tag{2.17}$$

Using the property expressed by (2.17), we can rewrite (2.16) as

$$\frac{\partial y}{\partial \boldsymbol{w_{cm}}} = \frac{s_m(\boldsymbol{x}, \boldsymbol{w_a})}{\sum_{l=1}^{R} s_l(\boldsymbol{x}, \boldsymbol{w_a})} \frac{\partial \delta_m(\boldsymbol{w_c})}{\partial \boldsymbol{w_{cm}}} \qquad m = 1,2,...,R \tag{2.18}$$

Note that this formulation is still general enough to accommodate constant, first order, or any other type of consequent (including the TS type). We now need to determine the corresponding partial derivatives with respect to the antecedent parameters $\boldsymbol{w_a}$. From derivation of (2.15) with respect to $\boldsymbol{w_a}$ we obtain

$$\frac{\partial y}{\partial \boldsymbol{w_a}} = \frac{\left[\sum_{l=1}^{R} \delta_l(\boldsymbol{w_c}) \frac{\partial s_l(\boldsymbol{x}, \boldsymbol{w_a})}{\partial \boldsymbol{w_a}}\right]\left[\sum_{l=1}^{R} s_l(\boldsymbol{x}, \boldsymbol{w_a})\right] - \left[\sum_{l=1}^{R} \delta_l(\boldsymbol{w_c}) s_l(\boldsymbol{x}, \boldsymbol{w_a})\right]\left[\sum_{l=1}^{R} \frac{\partial s_l(\boldsymbol{x}, \boldsymbol{w_a})}{\partial \boldsymbol{w_a}}\right]}{\left[\sum_{l=1}^{R} s_l(\boldsymbol{x}, \boldsymbol{w_a})\right]^2} \tag{2.19}$$

Using (2.15) to substitute for the second term in the denominator of (2.19) and simplifying, we have

$$\frac{\partial y}{\partial \boldsymbol{w_a}} = \frac{\sum_{l=1}^{R} [\delta_l(\boldsymbol{w_c}) - y(\boldsymbol{x_i}, \boldsymbol{w})] \frac{\partial s_l(\boldsymbol{x}, \boldsymbol{w_a})}{\partial \boldsymbol{w_a}}}{\sum_{l=1}^{R} s_l(\boldsymbol{x}, \boldsymbol{w_a})} \tag{2.20}$$

Let us denote with $\boldsymbol{w_{al}}$ the set of consequent parameters related with the *l*-th rule, those parameters will only influence the *l*-th rule, that is:

$$\frac{\partial s_i(\boldsymbol{x},\boldsymbol{w}_a)}{\partial \boldsymbol{w}_{aj}} = 0 \Leftrightarrow i \neq j, \quad i \in \{1,2,...R\} \quad j \in \{1,2,...R\} \tag{2.21}$$

Using (2.21), we can rewrite (2.20) as

$$\frac{\partial y}{\partial \boldsymbol{w_{am}}} = \frac{\delta_m(\boldsymbol{w_c}) - y(\boldsymbol{x},\boldsymbol{w})}{\sum_{l=1}^{R} s_l(\boldsymbol{x},\boldsymbol{w_a})} \frac{\partial s_m(\boldsymbol{x},\boldsymbol{w_a})}{\partial \boldsymbol{w_{am}}} \quad m = 1,2,...,R \tag{2.22}$$

The only missing part of this formulation is the last derivative term that depends on the type of consequent in the case of (2.18) or in the type of *t-norm* and membership function for (2.22). Once the specific case at hand is fixed, these last sensitivities are easy to derive, as we shall see in later Chapters 3 and 5 for some specific instances of FLS.

Substituting Equation (2.18) in (2.10) we obtain the batch training update for the consequent parameters:

$$\boldsymbol{w_{cm}^{new}} = \boldsymbol{w_{cm}^{old}} - \frac{\eta_c}{N} \sum_{i=1}^{N} \frac{[y(\boldsymbol{x_i},\boldsymbol{w}) - y_{di}] s_m(\boldsymbol{x}_i,\boldsymbol{w_a})}{\sum_{l=1}^{R} s_l(\boldsymbol{x}_i,\boldsymbol{w_a})} \frac{\partial \delta_m(\boldsymbol{w_c})}{\partial \boldsymbol{w_{cm}}} \qquad m = 1,2,...,R \tag{2.23}$$

While using Equation (2.22), we obtain the batch training update for the antecedent parameters:

$$\boldsymbol{w_{am}^{new}} = \boldsymbol{w_{am}^{old}} - \frac{\eta_a}{N} \sum_{i=1}^{N} \frac{[y(\boldsymbol{x_i},\boldsymbol{w}) - y_{di}][\delta_m(\boldsymbol{w_c}) - y(\boldsymbol{x}_i,\boldsymbol{w})]}{\sum_{l=1}^{R} s_l(\boldsymbol{x}_i,\boldsymbol{w_a})} \frac{\partial s_m(\boldsymbol{x}_i,\boldsymbol{w_a})}{\partial \boldsymbol{w_{am}}} \tag{2.24}$$

$$m = 1,2,...,R$$

Analogously the pattern-by-pattern update equations are obtained using (2.11) instead of (2.10), thus leading to the pattern-by-pattern training update for the consequent parameters:

$$\boldsymbol{w_{cm}^{new}} = \boldsymbol{w_{cm}^{old}} - \eta_c \frac{[y(\boldsymbol{x_i},\boldsymbol{w}) - y_{di}] s_m(\boldsymbol{x}_i,\boldsymbol{w_a})}{\sum_{l=1}^{R} s_l(\boldsymbol{x}_i,\boldsymbol{w_a})} \frac{\partial \delta_m(\boldsymbol{w_c})}{\partial \boldsymbol{w_{cm}}} \qquad \begin{matrix} m = 1,2,...,R \\ i = 1,2,...,N \end{matrix} \tag{2.25}$$

and to the pattern-by-pattern training update for the antecedent parameters:

$$\boldsymbol{w}_{\boldsymbol{am}}^{new} = \boldsymbol{w}_{\boldsymbol{am}}^{old} - \eta_a \frac{\left[y(\boldsymbol{x_i},\boldsymbol{w}) - y_{di}\right]\left[\delta_m(\boldsymbol{w_c}) - y(\boldsymbol{x}_i,\boldsymbol{w})\right]}{\sum_{l=1}^{R} s_l(\boldsymbol{x}_i,\boldsymbol{w_a})} \frac{\partial s_m(\boldsymbol{x}_i,\boldsymbol{w_a})}{\partial \boldsymbol{w_{am}}} \quad \begin{matrix} m = 1,2,\ldots,R \\ i = 1,2,\ldots,N \end{matrix} \tag{2.26}$$

In Equations (2.23) to (2.26) we introduced different learning rates η_a and η_c for antecedent and consequent, respectively. This is often seen in FLS training, thus, antecedent and consequent parameters would have different learning rates, as, for example, also parameters encoding center and width of a membership function would. This does not really correspond to a gradient descent approach, but to a gradient pre-conditioning [3] and will be discussed in more detail in Section 2.6. In the fuzzy systems literature the use of different learning rates for *logically* different parameters is motivated by the analogy to neural networks, where it is suggested to use different learning rates for each layer in the network [20].

2.4.2 Supervised Learning Formulation for a FLS

We will now derive the batch and pattern-by-pattern update equations for the case of a FLS, that is a conventional FLS in which the membership functions for antecedent and consequent may (and in general will) appear in more than one rule. This slightly complicates the formulation since all the rules that are affected by one parameter need to be counted in evaluating the derivative of the output of the FLS with respect to the parameter itself. Therefore, the main difference in the following derivation is that the simplifying assumption given by (2.17) and (2.21) does not hold and a slightly more complicated one is used. As in Equation (2.14) we can define the firing strength of rule *l* as

$$s_l(\boldsymbol{x},\boldsymbol{w_a}) = \bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_j,\boldsymbol{w_a}) \tag{2.27}$$

The output of the FLS is therefore expressed by

$$y(\boldsymbol{x},\boldsymbol{w}) = \frac{\sum_{l=1}^{R} \delta_{h(l)}(\boldsymbol{w_c}) s_l(\boldsymbol{x},\boldsymbol{w_a})}{\sum_{l=1}^{R} s_l(\boldsymbol{x},\boldsymbol{w_a})} \tag{2.28}$$

Let us now denote by $\boldsymbol{w}_{\boldsymbol{c}m}$ the consequent parameters related to the m-th consequent membership function (or local model, for TS fuzzy models), where m = 1,2, …, H. Let us also define the set of rule indices that contain the m-th consequent as their output as

$$I_m^c = \{ l \in \aleph | h(l) = m \} \tag{2.29}$$

A property similar to (2.17) can now be established. Since the derivative of the output terms with respect to $\boldsymbol{w}_{\boldsymbol{c}m}$ is zero for all the rules that do not have the m-th consequent as their output,

$$\frac{\partial \delta_{h(i)}(\boldsymbol{w_c})}{\partial \boldsymbol{w_{cj}}} = 0 \ \Leftrightarrow i \notin I_j^c, \ \ i \in \{1,2,...R\} \ \ j \in \{1,2,...H\} \tag{2.30}$$

Note that (2.17) can be obtained from (2.30) when $I^c{}_m = \{m\}$ for every value of m and H = R. We can now compute the sought after partials of the output of the FLS with respect to the m-th consequent parameters as

$$\frac{\partial y}{\partial \boldsymbol{w_{cm}}} = \frac{\sum\limits_{l \in I_m^c} s_l(\boldsymbol{x}, \boldsymbol{w_a})}{\sum\limits_{l=1}^{R} s_l(\boldsymbol{x}, \boldsymbol{w_a})} \frac{\partial \delta_m(\boldsymbol{w_c})}{\partial \boldsymbol{w_{cm}}} \quad m = 1,2,..., H \tag{2.31}$$

A similar approach will now be developed for the partial derivatives of the FLS output with respect to the antecedent parameters. Let us denote by $\boldsymbol{w}_{\boldsymbol{a}pm}$ the antecedent parameters related to the p-th input and the m-th antecedent membership function, where m = 1,2, …, K_p and p = 1,2, …, n. Let us also define the set of rule indices that contain the m-th antecedent membership function as their p-th input:

$$I_{pm}^a = \{ l \in \aleph | k(p,l) = m \} \tag{2.32}$$

Thus, the firing strength of the l-th rule will have a zero partial derivative with respect to $\boldsymbol{w}_{\boldsymbol{a}pm}$ whenever the m-th antecedent membership function for the p-th input does not appear in the l-th rule. That is

$$\frac{\partial s_i(\boldsymbol{x}, \boldsymbol{w}_a)}{\partial \boldsymbol{w}_{ajm}} = 0 \ \Leftrightarrow i \notin I_{jm}^a, \ \ i \in \{1,2,..., R\} \ \ j \in \{1,2,..., n\} \ \ m \in \{1,2,..., K_j\} \tag{2.33}$$

Using (2.28) and (2.33) and the same approach used in deriving (2.22), we can compute the partials of the output of the FLS with respect to the antecedent parameters as

$$\frac{\partial y}{\partial w_{\boldsymbol{a}pm}} = \frac{\displaystyle\sum_{l \in I_{pm}^{a}} \left[\delta_{h(l)}(\boldsymbol{w_c}) - y(\boldsymbol{x},\boldsymbol{w})\right] \frac{\partial s_l(\boldsymbol{x},\boldsymbol{w_a})}{\partial w_{\boldsymbol{a}pm}}}{\displaystyle\sum_{l=1}^{R} s_l(\boldsymbol{x},\boldsymbol{w_a})} \qquad \begin{array}{c} p = 1,2,\ldots,n \\ m = 1,2,\ldots,K_p \end{array} \tag{2.34}$$

It is easy to verify how (2.22) is a special case of (2.34). Once again, the only missing part of this formulation is the last derivative term that depends on the type of consequent in the case of (2.31) or in the type of *t-norm* and membership function for (2.34). Once the specific case at hand is fixed, these last sensitivities are easy to derive, as we shall see in later Chapters 3 and 5 for some specific instances of FLS.

Substituting Equation (2.31) in (2.10) we obtain the batch training update for the consequent parameters:

$$\boldsymbol{w}_{\boldsymbol{c}m}^{new} = \boldsymbol{w}_{\boldsymbol{c}m}^{old} - \frac{\eta_c}{N}\sum_{i=1}^{N}\left[y(\boldsymbol{x_i},\boldsymbol{w}) - y_{di}\right]\frac{\displaystyle\sum_{l \in I_m^c} s_l(\boldsymbol{x}_i,\boldsymbol{w_a})}{\displaystyle\sum_{l=1}^{R} s_l(\boldsymbol{x}_i,\boldsymbol{w_a})}\frac{\partial \delta_m(\boldsymbol{w_c})}{\partial \boldsymbol{w}_{\boldsymbol{c}m}} \qquad m = 1,2,\ldots,H \tag{2.35}$$

While using Equation (2.34), we obtain the batch training update for the antecedent parameters:

$$\boldsymbol{w}_{\boldsymbol{a}m}^{new} = \boldsymbol{w}_{\boldsymbol{a}m}^{old} - \frac{\eta_a}{N}\sum_{i=1}^{N}\left\{\left[y(\boldsymbol{x_i},\boldsymbol{w}) - y_{di}\right]\frac{\displaystyle\sum_{l \in I_{pm}^{a}} \left[\delta_{h(l)}(\boldsymbol{w_c}) - y(\boldsymbol{x}_i,\boldsymbol{w})\right] \frac{\partial s_l(\boldsymbol{x}_i,\boldsymbol{w_a})}{\partial \boldsymbol{w}_{\boldsymbol{a}pm}}}{\displaystyle\sum_{l=1}^{R} s_l(\boldsymbol{x}_i,\boldsymbol{w_a})}\right\} \tag{2.36}$$

$$p = 1,2,\ldots,n \quad m = 1,2,\ldots,K_p$$

Analogously the pattern-by-pattern update equations are obtained by using (2.11), instead of (2.10), thus leading to the pattern-by-pattern training update for the consequent parameters:

$$\boldsymbol{w}_{\boldsymbol{c}m}^{new} = \boldsymbol{w}_{\boldsymbol{c}m}^{old} - \eta_c\left[y(\boldsymbol{x_i},\boldsymbol{w}) - y_{di}\right]\frac{\displaystyle\sum_{l \in I_m^c} s_l(\boldsymbol{x}_i,\boldsymbol{w_a})}{\displaystyle\sum_{l=1}^{R} s_l(\boldsymbol{x}_i,\boldsymbol{w_a})}\frac{\partial \delta_m(\boldsymbol{w_c})}{\partial \boldsymbol{w}_{\boldsymbol{c}m}} \qquad \begin{array}{c} m = 1,2,\ldots,H \\ i = 1,2,\ldots,N \end{array} \tag{2.37}$$

and to the pattern-by-pattern training update for the antecedent parameters:

$$\boldsymbol{w}_{\boldsymbol{a}m}^{new} = \boldsymbol{w}_{\boldsymbol{a}m}^{old} - \eta_a \left[y(\boldsymbol{x_i}, \boldsymbol{w}) - y_{di} \right] \frac{\sum_{l \in I_{pm}^{a}} \left[\delta_{h(l)}(\boldsymbol{w_c}) - y(\boldsymbol{x}_i, \boldsymbol{w}) \right] \frac{\partial s_l(\boldsymbol{x}_i, \boldsymbol{w_a})}{\partial \boldsymbol{w_{a}}_{pm}}}{\sum_{l=1}^{R} s_l(\boldsymbol{x}_i, \boldsymbol{w_a})} \tag{2.38}$$

$$p = 1,2,...N \quad m = 1,2,..., K_p \quad i = 1,2,..., N$$

Equations (2.35) and (2.36) can be used to implement a batch mode gradient descent, while (2.37) and (2.38) can be used for a pattern-by-pattern gradient descent training. Such a general approach to the learning problem has not been proposed in the literature to date, some approaches utilizing full (or complete) rule bases and with some other restricting assumptions have been proposed and will be discussed in the following Section 2.5. The merit of equations (2.35) to (2.38) is to provide a general framework for optimizing a FLS without a necessarily complete[1] rule base. Moreover, the many existing learning approaches developed for IMF-FLSs can still be derived through the previous equations (2.35) to (2.38). Thus, equations (1.25) along with (2.35) and (2.36) describe the output and the batch update equations of a FLS that could be a pure FLS, a TS-FLS, or an IMF-FLS. Changing the parameters involved in the formulation (e.g., K_j, and H and function $h(l)$ and $k(j,l)$), we can define and tune any FLS we desire, thus implicitly defining a *continuum* of fuzzy logic systems among which to choose. For example, we can start with a FLS with 2 inputs and one output, 3 membership functions for each input and 5 membership functions for each output. In our notation this means: K_1 = 3, K_2 = 3, and H = 5. Let us assume the case of a complete rule-base, i.e., R = 9. In case we decide to increase H to 7 we still have a FLS. Increasing H to 9, we obtain a TS-FLS. All sorts of modifications along these lines can be carried out, thus showing the flexibility and the *continuum* implemented by the adopted modeling approach.

[1] By complete it is generally meant that there are as many rules as all the possible combinations of antecedent fuzzy sets on the different linguistic variables.

2.5 Supervised Learning: State of the Art

In this Section we review the existing supervised learning approaches for FLSs, starting with the ones for IMF-FLSs.

The pioneering attempts to tune FLSs with gradient descent can be traced back to 1990 and 1992. In 1990 Hayashi *et al.* [19] proposed a particular type of FLS where the antecedent membership functions as well as the consequents consisted of neural networks. Such an approach led to easy application of the back-propagation algorithm and was tested on the control of an inverted pendulum system.

Subsequently, Horikawa *et al.* [24,83] presented an approach in which only the antecedent membership functions are approximated by a neural network. Using sigmoid squashing functions in the neural network, they obtain membership functions for the input, that are what they call *pseudo-trapezoidal* membership functions (essentially linear combinations of sigmoids). They use back-propagation on the overall neural network comprised of the rules encoded in the network weights. Product inference is used along with constant consequents and pattern-by-pattern training. They rule out the possibility of using triangular membership functions due to the impossibility of using BP with this type of membership functions. Moreover, they contend that the mapping offered from a FLS using triangular membership functions is of class $\boldsymbol{C}^{\mathbf{0}}$, while other membership functions (i.e., gaussian, bell-shaped) generate $\boldsymbol{C}^{\infty}$ mappings that thus seem more attractive. In their experimentation the learning rate (a small constant) is fixed differently for every layer and is scaled for the quantities to update. They note that using too high a value for the learning rate might result in membership functions with very little linguistic meaning since they might significantly depart from the initial (*ad hoc*) ones. Finally, a very interesting note they make is that in their experience it is the structure of the FLS that affects the performance more than the membership functions parameters themselves. Their approach is tested with a function approximation task. An interesting note about this approach is the use of a dummy variable in the function, in order to verify if the network will discriminate this variable and determine whether it is dummy, or not.

One of the first approaches to tuning antecedent and consequent parameters of a fuzzy logic system through gradient descent is the one of Nomura *et al.* in 1992 [51,52]. In this approach triangular membership functions, product inference, constant consequents, singleton fuzzification, and COA defuzzification are used. In the remainder of this chapter it will be assumed, unless otherwise noted, that singleton fuzzification and COA defuzzification are used, since they are the most common choices for FLSs. Nomura's FLS uses membership functions in the antecedent part set independently for each inference rule, and independent constants for each inference rule as well. Thus, their FLS can be considered one of the first examples of IMF-FLS. They tune the consequent constants and two parameters of the antecedent membership functions: center and width of each (isosceles) triangle membership function. They execute a pattern-by-pattern training alternating:

1) Datum presentation;
2) Update of the consequent parameters;
3) Recalculation of the output and update of the antecedent parameters.

This approach was tested on three function approximation examples and a mobile robot obstacle avoidance problem. It appears to be much faster and slightly better generalizing then a neural network (that has more parameters than their model, thus making the comparison uneven). Moreover, they also compare their approach to *iterative fuzzy modeling*. Iterative fuzzy modeling uses pure FLSs with constant consequents (one per rule, thus a TS-FLS) and membership functions constrained to have a total sum of firing strengths equal to unity. Thus, a triangular membership function has its own center and width defined by the two neighboring membership functions centers. This way a 0.5 overlap is always ensured and the sum of firing strengths is always unity for a full rule base. Nomura *et al.* compare their proposed method with the iterative fuzzy modeling they reference and show that their approach tends to be somewhat slower in converging to the specified error goal than the iterative approach that conversely shows a somewhat higher test error. From these results they infer their approach has higher generalization capabilities. This claim does not necessarily seem to hold; the comparison is based on one optimization run (not necessarily converging to a global solution) and with less than an order of magnitude of difference in the test error. More importantly, the number of parameters of their approach is significantly higher than the number of parameters of iterative fuzzy modeling (80 parameters in their approach versus presumably 24 parameters in the iterative approach). In the presented approach a different learning rate (step size) was chosen for the consequent constants, the triangle centers and the widths, thus having three different learning rates. They also note the non-differentiability of the error function with respect to the triangle center, and pragmatically take care of it by zeroing the corresponding parameter correction when necessary. This is done by introducing a sign function that is +1, -1 or zero when its argument is positive, negative or zero, respectively. No mention on the discontinuity of the derivative with respect to the width parameters is made. Moreover, this non-differentiability consideration (and consequent appropriate definition of the sign function) is exposed in [52] but not in [51]. The first reference [52] was published in the Japanese Journal of Fuzzy Theory and Systems, thus having a more limited circulation than the latter [51] published in the proceedings of the first IEEE international conference on fuzzy systems. Indeed, many subsequent authors have used this approach neglecting the non-differentiability problem.

The work of Nomura *et al.* was subsequently extended by Miyata *et al.* [46] and Ohki *et al.* [57] (same authors, but with different order) to the case of piecewise linear membership functions. Product inference and constant consequents are used. The antecedent membership functions are sort of triangularly shaped but with each line of the triangle replaced by two segments with different slopes. Such a membership function can be no longer represented by two parameters, but three additional parameters are needed. The FLS is an IMF-FLS and the tuning is a pattern-by-pattern tuning. They apply gradient descent for the minimization of the approximation error, and no mention of the non-differentiability problem (even bigger in this case) is made. The updates for consequent and antecedent parameters are not performed at the same time, but alternated as in Nomura's approach. In [46] this approach is applied to the parallel parking of an autonomous mobile robot; constant learning rates are used. Moreover, the use of piecewise linear membership functions is compared with triangular membership functions (two parameters) and asymmetric gaussian membership functions (three parameters). Piecewise linear membership functions seem to require far less training cycles for the same training error goal. This approach is again biased on having used only one optimization run and also by comparing two FLSs with a different number of parameters. Another comparison for a function approximation task is carried on for different choices of step-sizes. This example seems to consistently show a faster convergence of the piecewise linear membership functions as compared to the strictly triangular membership functions. As previously mentioned, the comparison does not seem significant due to the different number of parameters in the two approaches. The work in [46] is then extended in [57] to the use of varying step sizes. Two function approximation problems are presented and used to demonstrate the higher capability of expression of the piecewise linear membership functions versus triangular membership functions. Moreover, a method to modify the learning rate in order to avoid sub-optimal solutions is presented. This approach, named learning algorithm using coefficients modified by a global search (LACG), is not as promising as its name; it is merely a heuristic (not too clearly explained) based on the increase or decrease in objective (error) function. In case of error oscillations the learning rate is subsequently doubled, halved, quadrupled, etc. This LACG approach seems to be effective in reducing the number of learning cycles necessary to achieve a given error goal.

Bersini and Gorrini [4] use Nomura's method [51] for adaptive fuzzy control. They propose a slight modification to the method to make it a conventional FLS instead of an IMF-FLS. They briefly derive the new update equations for the adjustable parameters, without getting into too many details. Those equations are very similar to (2.37) and (2.38) (pattern-by-pattern update equations) even though they are not as general and detailed. The results of their approach are not compared with the traditional Nomura's approach.

In the same year (1992) and at the same conference (IEEE international conference on fuzzy systems) Wang and Mendel present a back-propagation approach very similar to Nomura's one (i.e., yet another IMF-FLS) but employing gaussian instead of triangular membership functions [82,83,85,86]. They show that a FLS can be represented as a three layer feed-forward network, motivating the use of the back-propagation algorithm to optimize it. A FLS with constant consequent terms and product inference is trained using a pattern-by-pattern approach. The adjustable parameters are the constant consequents as well as center and spread for the antecedent Gaussian membership functions. Another subtle difference from Nomura's approach is that the same step size is used for different classes of parameters. Thus, Wang and Mendel approach is a true gradient descent approach. Moreover, in this case the updates for all the adjustable parameters are applied simultaneously (unlike in Nomura's approach). They test the approach on some system identification problems that had been solved with neural networks [47]. The FLS approach is better (faster convergence and less parameters required for the same error goal) than its neural counterpart. Probably an easier and more intuitive initialization of the adjustable parameters has a significant role in this different behavior. Moreover, the outcome of the optimization is something that is readable and makes more sense than the collection of weights of a neural network.

Still Wang and Mendel, and in the same conference, propose an orthogonal least squares learning using fuzzy basis functions [81,85]. The rationale of this approach is to fix the antecedent membership functions parameters in order to reduce the learning problem to a linear problem. Orthogonal least squares techniques are effective in determining the necessary basis functions (i.e., rules) and the corresponding (globally) optimized consequent parameters. This approach certainly gains in terms of readability but looses in terms of function approximation, since a fewer number of parameters is available for adjustment; and the non-adjustable parameters are fixed at generally sub-optimal values.

A different approach to the supervised learning of FLSs was presented in 1993 by Jang [26]. In his paper he proposed an adaptive-network-based fuzzy inference system that he calls ANFIS. In this paper and other following papers and book [27,28] he looks at FLSs as adaptive networks and thus shows how back-propagation learning algorithms can be derived from the corresponding network structure. This view offers some flexibility in deriving the gradient descent rule, avoiding the cumbersome task encountered in deriving the update equations (2.35) to (2.38). On the other hand, the view of pure fuzzy systems and TS-FLS is not a *continuum* anymore since, for example, changing the number of consequent membership functions constitutes a structural change in the underlying network. In his approach Jang uses bell shaped membership functions, even though he mentions that any other membership function such as piecewise linear membership functions would be acceptable; however no mention of the non-differentiability problem of piecewise linear membership functions is made. Product inference is used, and both pure FLSs and TS-FLSs are considered. An adaptive step-size is used, in which the adaptation is given by two simple heuristics (in full analogy to the quick-prop algorithm):

1) Increase the step-size when the error is steadily decreasing;
2) Decrease the step-size when the error is oscillating.

A pattern-by-pattern approach is used and the learning is composed of the alternation of recursive least square approximation for the consequent parameters with successive gradient descent for the antecedent parameters. The ANFIS approach seems to be far superior to neural networks (trained with quick propagation) having the same number of parameters or more, in both training speed and generalization (i.e., test error). The ANFIS approach was tested on function approximation, system identification problems (as in [47]), and the Mackey-Glass differential delay equation, yielding promising results. Moreover, its performances were compared to those of autoregressive models, and high order polynomials (besides neural networks) with excellent improvements over the solutions offered by these alternative approaches.

Departing from the main approaches to supervised learning that utilize gradient descent and neuro-fuzzy analogies, Rashid and Heger [65] present an approach to tuning FLSs based on least square estimation. They choose Gaussian membership functions along with product *t-norm* and constant consequents. The consequents are the only adjustable parameters, in order not to risk loosing readability and interpretability of the FLS. A recursive least square approach to tune the consequents is applied to the control of the level of fluid in a tank. The FLS controller design is optimized to approximate a feedback linearization controller.

Guely and Siarry [16] present an approach to gradient descent tuning, using asymmetric triangular membership functions as well as product and minimum *t-norms*. This approach can be seen as an extension of Nomura’s approach for TS-FLSs with independent membership functions in each rule and constant as well as affine consequents.

They also propose centered-affine consequents, where the consequents are affine in the variables deviation from the antecedents' centers, rather than in the input variables themselves. They test their approach on an interesting function approximation problem (SISO function) that they formulate. They also offer some remarks on the relation between learning rate and training data, membership functions and number of rules. They experience divergence of the algorithm for a small number of rules; indeed in an IMF-FLS a small number of rules is probably not enough to produce full coverage of the input space. Moreover, increasing the number of rules initially results in a decreased approximation error, until a threshold is met where the increase in number of rules is not accompanied anymore by a decrease in error. They also find that the number of training data has a direct impact on the generalization capabilities of their FLS, and that, as a rule of thumb, the number of training data should be at least twice the number of adjustable parameters. The initial number of overlapping membership functions also seems to have an influence on the outcome of gradient descent, thus suggesting the local nature of the tuning approach. In their experiment they obtain the best results with two or three membership functions initially overlapping. Finally, they point to the need for further studies on learning rate in relation with the function to approximate, influence of the number of rules and membership function overlap. In this study (quite) different learning rates are used for antecedent membership functions centers, spreads and consequent constants. Moreover, there is no mention about the non-differentiable nature of the problem with triangular and/or minimum inference.

Jou [30] studies the learning of IMF-FLSs with gaussian antecedent membership functions, constant consequents and product *t-norm*. Pattern-by-pattern training is used and the LMS algorithm is proposed to tune the output constants. Back-propagation (i.e., gradient descent) is also proposed for antecedents and consequents tuning, with the note that the membership functions involved in the training need to be differentiable for the gradient descent to apply. In the gradient descent approach he uses different (constant) learning rates for different classes of parameters. He also proposes a hybrid algorithm that is a heuristic for clustering-based positioning of the antecedent membership functions, with successive application of LMS to the consequents. He also proposes a two-stage back-propagation, that is, a back-propagation initialized by the heuristic clustering. Two function approximation problems and one system identification problem are considered. From his results it can be inferred that tuning some consequent parameters through LMS might be slower then tuning less parameters for antecedent and consequent together and using gradient descent. Testing the approach on a classification problem he also finds that tuning only the consequent parameters by LMS is faster than tuning less parameters for antecedents and consequents (using gradient descent). An interesting result is that the LMS approach will quickly converge to a hard decision boundary between points, while the back-propagation approach slowly converges to a smooth decision boundary. Finally, on the system identification problem he finds that the two-stage back-propagation is faster than the hybrid learning. That is, adjusting the membership functions is useful in general, but the corresponding optimization problem will have many local minima; thus, the initialization is a crucial aspect for the convergence properties of the gradient descent approach.

Katayama *et al.* [31] are the first to propose a gradient based constrained optimization approach that makes use of interior penalty function methods. Triangular membership functions, product inference, and constant consequents are used. The triangular membership functions are set such that each triangle has its own center and it extends to the right and to the left to the centers of the corresponding neighboring membership functions, so that a 0.5 level of overlap and a total firing strength of unity are always guaranteed. Moreover, the membership functions can be easily constrained by imposing that the order between the centers is preserved. A constant step size common to all the different parameters is used throughout the paper. They use a univariate function approximation problem (of their own formulation) to test the developed approach. A smooth learning curve is shown, with final membership functions that satisfy the constraints, and thus still “make sense” from an intuitive FLS point of view. On the other hand, the solution for an unconstrained approach is computed and the result is a set of membership functions that violate the constraints and thus do *not* “make sense”. No mention of the approximation error in this second case is made. They also present a heuristic for rule generation and compare it to other existing heuristics, showing improved results. Finally, they briefly talk about the application of this approach to control and in particular to model reference adaptive control. Again, no mention of the non-differentiable nature of the triangular functions is made.

Glorennec [14] presents some learning algorithms and considerations for neuro-fuzzy networks. He discards the possibility of using gradient descent for antecedent parameters because of: slowness, convergence problems, local minima, difficulty to introduce constraints, and excessive computational hardware requirements. He tunes a TS-FLS with constant outputs, triangular membership functions, and product inference. For the antecedent fuzzy sets he imposes the constraint that when one fuzzy set has unit membership degree, the other fuzzy sets have zero membership degree. As a particular solution he chooses triangular membership functions intersecting at 0.5 membership degree. He proposes to decompose the learning into antecedent and consequent learning that alternate iteratively as in Nomura *et al.* [51,52]. For consequent learning he proposes gradient descent with momentum coefficient (i.e., a gradient deflection approach), least square (off-line) and recursive least square (on-line). For antecedent membership function optimization he recognizes that triangular membership functions are non-differentiable and thus gradient descent cannot be used. He proposes a finite difference approximation of the gradient and uses the approach for a test function approximation problem. The concern with this approach is that if the gradient does not exist, even its finite difference approximation will not. He also proposes the use of random optimization in order to take care of the non-differentiable issues. Finally, he presents some results for only one function approximation case.

Jacomet *et al.* [86] propose the use of the downhill simplex method [50] for the supervised learning of few parameters of a FLS. In choosing their optimization algorithm they discard NN-based methods (gradient based) because of several reasons: they are slow, the penalty function is nonlinear in the parameters, it might be non-differentiable, there might be constraints on the parameters, and it should be possible to use the same algorithm for a simulation and for the real system as well. They claim that the algorithm would be suitable for online applications by proper choice of termination criteria and constant restarting. They look at a very simple second order system and adjust two parameters of a proportional-derivative (PD) controller, achieving a value of penalty function that will turn out to be their best value. They adjust the input and output scaling constants of a 2×1 (i.e., two inputs and one output) fuzzy system (a total of 3 parameters) with 5 triangular membership functions for each input. This approach results in achieving a slightly higher penalty function value than by the PD controller. By adjusting two other parameters (i.e., the position of the center of two triangular functions), they achieve the same performances of the optimal PD controller (5 parameters for the FLS versus 2 parameters for the PD controller). They test the downhill simplex method on a small number of variables; that is, in the conditions where this algorithm generally works very well. On the other hand, it is also known that for more then 5-6 variables the downhill simplex method performance degrades.

In 1997 Arabshahi *et al.* [1,76] present an interesting paper on supervised learning of FLSs with gaussian membership functions, center of gravity defuzzification and min-max operators. They use min-max operators in formulating the gradient descent approach with the comfort that (they say!) min and max are continuous and differentiable functions. Moreover, the min and max operations act as pointers with the result that only the function that gave rise to the min or max for a specific rule is adapted, while the others are not. Note that with a product *t-norm* instead of the min, all the other membership functions that contributed to the output in the product will be adjusted as well. Thus, what they suggest as being an advantage might indeed not be one. They also present a very nice approach to the gradient descent for a two-input FLS with min and max operators. This is somewhat similar to the approach developed in the previous Section 2.4. The interesting aspect of their work is an approach to adaptive pruning of membership functions by annihilation and fusion mechanism. That is, if a membership function becomes too narrow or two of them become too close they are annihilated or fused, respectively, and the rule table is modified accordingly. They use a pattern-by-pattern steepest descent approach with constant learning rates and apply it to a function approximation problem.

Zhang and Knoll [90] propose to use B-spline membership functions instead of the usual triangular or trapezoidal ones. They point to the fact that – heuristically – adjusting the position (center) of membership functions is intuitive, but the same does not hold for the spread. They also point to the need of a systematic design procedure for FLSs to satisfy the increasing application areas. They use a regular FLS approach for the rule base with singleton outputs, one for each rule (TS-FLS with constant local models) and product inference. They look at the FLS as an interpolation process in principle. They use instantaneous errors to define a pattern-by-pattern gradient descent approach in the consequent parameters (singletons) and discover that the objective function is convex with respect to these parameters (that is known since the problem is linear in the consequent parameters). They also provide a heuristic algorithm for placement of the points defining the B-splines. Their proposed training algorithm consists of an alternation (iteration by iteration) of the placement algorithm with the gradient descent approach for consequents. The attractive feature of this approach is that they say that a new membership function can be easily inserted, something that is not easy to do with other approaches. They compare their results with [19,26,27,45] and claim they obtain better results with the same number of adjustable parameters. They always end up with very big structures and claim fast convergence. (They are solving a linear problem.)

Nauck and Kruse [49] propose a neuro-fuzzy system for function approximation that they call NEFPROX. Their learning algorithm is a heuristic able to determine the structure and parameters of a fuzzy system. Their approach is a FLS approach of the type of ANFIS [26] with the added flexibility of allowing more interpretable Mamdani like fuzzy systems (i.e., pure FLSs). Their rationale in deriving a heuristic for training purposes is based on the consideration that a FLS should not be chosen when an exact solution is sought; thus, it is not worth to pursue a complicated learning algorithm. Asymmetric triangular and bell shaped membership functions are used along with min *t-norm* and mean of maxima defuzzification. (They argue this defuzzifcation is computationally faster and gives them the same results as COA defuzzification after training.) The proposed NEFPROX is what they call a fuzzy perceptron, or a 3-layers perceptron, and this neural view of the fuzzy system allows them to formulate a heuristic structure and parameter learning approach. The approach has the flexibility to accommodate constraints, such as the relative positioning of membership functions. The learning rate is varied, increasing it whenever the error steadily decreased and decreasing it when the error oscillates. They test the approach on the Mackey-Glass differential equation and compare it with ANFIS. In this test the membership functions are constrained as to their relative position. Their conclusions on the test are that ANFIS is slower in learning, but learns and generalizes better than NEFPROX. This is done at the cost of readability and interpretability: ANFIS solutions are poorly readable while NEFPROX offers pure FLS solutions that are thus readable. The authors' conclusion is that this is a typical example of trade-off between readability and performance. My interpretation of their results is a little different. Both ANFIS and NEFPROX have the same number of parameters. The ratio of training runtimes of ANFIS to NEFPROX is always smaller (by at least a factor of 2) than the inverse ratio of the achieved training or test error. Thus, the smaller error offered by ANFIS is more than counterbalanced by its higher runtime. Therefore, comparing ANFIS and NEFPROX on the basis of training error versus runtime might lead to opposite conclusions. Obviously, we cannot pursue this comparison since we would need to look at their (time) learning curves. This tendency holds for all the different runs of NEFPROX. Moreover, the first NEFPROX they obtain has 129 rules that are automatically learned and that are of Mamdani type. Even though their FLS is a pure one, and readable as such, the rules are of a number that surely prevents readability, while ANFIS has only 16 rules. Those rules express local models (thus less interpretable), but they are in a limited and still understandable number. Even with another NEFPROX using bell shaped membership functions and with only the best 26 rules, the results are analogous. In the best case the ratio of ANFIS runtime to NEFPROX runtime is almost 26 while the NEFPROX to ANFIS ratio of test errors is almost 39.

To contrast the IMF-FLS approaches Shi and Mizumoto [75] present a neuro-fuzzy learning algorithm for TS-FLS that is characterized by not having independent membership functions for each rule. They develop a gradient descent approach for a two input, single output system with product inference and full rule base. They claim that is not hard to extend the approach to a generic MISO FLS, but their formalism does not seem to make it that easy. This approach is different from the equations derived in the preceding Section 2.4 in many ways. First, it includes only the case of a full rule base with a different constant consequent per rule (a TS-FLS of zero order indeed). Using an instantaneous error measure, they develop a pattern-by-pattern gradient descent approach for both symmetric triangular and Gaussian membership functions. No mention of the non-differentiability of triangular functions is made, even though the sign function for the derivative calculation is defined as in Nomura *et al.* [52] in order not to have any correction if the parameter value corresponds to a point of non-differentiability. A different constant learning rate is used for centers, widths and consequents. They subsequently – loosely – discuss some properties of the developed approach. Obviously, membership functions are not independent of each other. Moreover, fitting training data, they say, is likely to be slower than in the IMF-FLS approaches. Their reasoning in this case is not clear at all to me, and I think this statement would actually deserve more attention. Their rationale in making the previous statement is that for a given membership function "fitting to training data is usually slower than the conventional ones (i.e., IMF-FLS), because it is not only used in one fuzzy rule, but also in other fuzzy rules if it fires. This implies that the membership function learning is sometimes slow due to the fact that it is related to many fuzzy rules." On the grounds of heuristic considerations I would say that each membership function has to satisfy the necessary firing requirements of all the rules it appears in; thus, it might be pulled in different directions due to eventually contrasting objectives. Still from an intuitive point of view, a "harmonious" FLS would probably possess the property that the membership functions all work together to the same goal, thus, not generating conflicting objectives in their updates. These are all theories and suppositions that would need to be verified on the basis of experimental results as well as more rigorous studies. It is also recognized that this type of formulation is more complex than the IMF-FLS one, and that the number of partitions on each input are not necessarily the same; an added flexibility of great importance. Indeed, it might be necessary to have finer partitions on one input with respect to which the approximated function varies more quickly. They also state that the number of tuning parameters in their approach is less than in the IMF-FLS for a MISO system. This is another controversial statement; it obviously depends on the basis for the comparison. Moreover, they state that the representation of the rule table does not change, i.e., it keeps its readability and non-firing states, or weak firing states, can be avoided (states that cause a weak or zero sum of firing strengths). It is intuitively understood that it is easier to constraint the membership functions in this case and that they will probably tend to cover the space even in the unconstrained case, even though a constrained approach is possible in both cases. The setting of initial fuzzy rules is simple in this proposed approach, a non-trivial property since the use of local optimization approaches is heavily dependent on a proper initialization. It also seems that their derivation of the update rules for triangular membership functions has some glitches. They test the approach on two function approximation problems, performing learning for triangular and gaussian membership functions and comparing the results with their IMF-FLS counterparts. The comparison is affected by several biases. In the Gaussian approach they compare to Wang's approach [82,83,85,86] with 16 rules and a total of 80 parameters, while they have still 16 rules, but a total of only 26 parameters. On 10 different initializations their approach seems to be slower in convergence, but it seems to produce smaller average and maximum test error (for fixed values of the training error goal). Indeed, the error function seems to be more uniform in their case rather than with the solution obtained by Wang's approach. It is plausible to expect the proposed FLS to cover the input space in a more uniform way. I think that a more reasonable test would have been to compare the two approaches using the same number of adjustable parameters, since in their testing strategy some of the difference in the results might be due to an over-parameterization and/or over-fitting of the data. Moreover, all these comparisons are always subjective in nature since the initialization is different, and the solution found might be a local optimum. In the case of triangular membership functions the proposed approach is compared to Nomura's approach [51,52]. In this case the tendency seems to be different; the developed method (using less parameters) seems to converge as fast and sometimes even faster than Nomura's method. The two approaches seem to offer similar generalization, since the average and maximum test errors are almost the same for theirs and Nomura's method. Repeating the experiment for triangular membership functions and almost the same number of parameters (31 in the proposed approach versus 45 in Nomura's approach), the authors find that the proposed method is significantly faster and possesses significantly better generalization capabilities.

Because of the slowness of the gradient descent approach, along with non-global convergence, and, some authors say, excessive computational burden many heuristics for the design of FLSs have been proposed [33,55,85] (among many others; this is not a comprehensive list of heuristics). The spirit of most of these approaches is based on clustering of the input space. One of the weakest points of using heuristics is the lack of a corresponding (optimal) term of comparison. Some of these heuristics are also used for structure identification. Indeed the theme of structure identification has been always addressed in a heuristic fashion.

Many adaptive control schemes using tuning of FLSs or IMF-FLSs have been presented in the literature, discussing them would be cumbersome and outside the scope of this work. However, these approaches generally end up solving online function approximation problems (either for identifying the plant or some relationship with a cost function, etc.). They range from very simple cases where only the (linear) consequents are tuned, to complicated cases where gradient descent is employed and alternated with structural learning. Many applications of the supervised learning techniques discussed above were presented in fuzzy control, adaptive control and system identification. Most of these approaches use either Wang's [85] or Nomura's [51] back-propagation approach.

All the methodologies discussed so far are either gradient based or heuristic. They are all local in nature (the objective functions involved are in general not convex) and therefore heavily dependent on their initialization. With fuzzy systems it is generally easy to propose an intuitively good initialization, but as the system becomes bigger, the quality of the initialization might substantially drop. In order to overcome the local nature of all these approaches, some authors have addressed the global optimization problem of FLSs. Almost all the approaches that were proposed are based on the application of stochastic searches like genetic algorithms, evolutionary programming, simulated annealing, tabu search and variations on those [17,18,23,53,54,56,78] (among others). The merit of these approaches is to try to address the local nature of the solution problem. These type of approaches present many drawbacks. They are very slow and thus unsuitable for online applications (unless the time frame for updates is long). Moreover, they might suffer from premature termination (e.g., GAs) or might have parameters that are hard to set (e.g., cooling schedule in simulated annealing). These approaches are very useful in complex problems where the knowledge of the internal mechanisms of the problem are unknown or too complicated to examine; they are formidable engineering tools in cases where there is the need, or desire, not to spend time on problem analysis. Indeed, the time consumption is moved from man-time to computer-time that is definitely more economical. In the application of stochastic searches to the supervised learning, or complete model identification, of FLSs, we are discarding and not exploiting the analytic information on the fuzzy systems themselves.

There are very few global approaches that do not rely on the use of stochastic searches. In the neural network literature the use of tunneling algorithms [9,36,87] for global optimization was proposed. These algorithms are based on a tunneling function that penalizes already found solutions and thus tries to find its way to new improved solutions, finally converging to a global optimal solution. This seems an interesting approach with the drawback of ill conditioning, and no strong guarantee of global convergence. Tunneling algorithms are based on the construction of penalty terms that are added to the original objective function. It is well known that penalty functions approaches are in general ill conditioned.

A different global approach is the expanded range approximation (ERA) [15]. The ERA is based on preserving the optimization procedure (i.e., gradient descent), and adding some data processing stages. The objective function shape and properties depend on the particular data set we are trying to approximate. In the ERA approach the data are all compressed to their mean value (*y*'s only) and progressively expanded in stages, until a final stage with the original data is reached. If the initial problem (with all the data compressed to their average) is globally convex, and the expansion is performed slowly enough, the authors make a convincing argument that the technique will lead to the global optimum. The authors applied it to the XOR problem (where the global optimum is known) reaching a 100% rate of global success (conversely to what happens using a pure gradient descent approach).

2.6 Discussion

In this section a synthetic discussion of the approaches reviewed in the previous Section 2.5 is presented. Some remarks on the problems associated with some of the approaches (non-differentiability, locality, readability, etc.) are made, and some research directions are identified. This discussion is central to the dissertation in that it leads to the experiments of Chapter 3, the new problem formulation of Chapter 4 and the quadratic fitting heuristics of Chapter 5.

2.6.1 Non-differentiability

In the problem description of Section 2.2 we showed that deriving a gradient descent approach for the supervised learning of FLSs always entails taking the derivatives of the output of the FLS with respect to the adjustable parameters. This operation is formally not possible anymore whenever we use a non-differentiable *t-norm* (e.g., minimum operator), a non-differentiable membership function (e.g., triangular), or any other type of non-differentiable (with respect to the adjustable parameters) component of the FLS. In these cases the output of the FLS is not differentiable (everywhere) with respect to the adjustable parameters. There are some points where right and left derivative exist but have different values. In this case a gradient descent approach cannot be applied anymore and any approach that uses a direction based on a derivative is not truly using a gradient direction.

The implications of this consideration are substantial, because if the direction of search ceases to be a descent direction the algorithm might not converge (even locally). In this case, in the progress of the algorithm some non-improving steps (i.e., steps where the objective function does not decrease) might be taken. Thus, from a theoretical point of view all of the gradient-based approaches developed for non-differentiable FLSs (e.g., [16,31,46,49,51,52,74,75]) are not guaranteed to be convergent. From a pragmatic point of view this might not be a concern, using the argument that the probability of falling onto a point of non-differentiability is zero since these points generally form a set of zero measure. The training data constitute a grid in the input space along which the membership functions move during the tuning. Therefore, the adjustable parameters might fall onto one of the training points or at least arbitrarily close to it. The presence of these “glitches” might make the algorithm converge in a slower fashion. The possible divergence and slowed convergence will be illustrated through two examples [11,12] in the following Chapter 3.

The literature has little if no discussion about this topic; many authors that do use gradient methods in non-differentiable conditions do not make any mention of the non-differentiability, but proceed to directly derive and apply gradient update equations. Conversely, some authors recognize the problem [24] and use Gaussian membership functions [30] or propose to zero the updates corresponding to falling onto a point of non-differentiability [51]. Pedrycz [59] observes the problem existing with non-differentiability of the min and max operation, but does not consider it a real problem in the practical operation of a learning algorithm. His concern is related to the possibility that the two-valued derivatives obtained with min and max operators might make the algorithm terminate at a sub-optimal point. Moreover, he observes that using opportune parameterized *t-norms* could eliminate the non-differentiability problem. Teow and Loe [79] propose a Fourier series analysis to estimate the derivative at non-differentiable points. They prove that the estimate is the average between left and right derivatives. Similarly, Glorennec [14] proposes to estimate the derivative with a finite difference approach, once again he will obtain the average of the left and right derivatives. Both approaches fully acknowledge the problem, even though they propose a solution to estimate a derivative that really does not exist. Therefore, the corresponding direction is not necessarily an improving direction for the objective function.

Problems that exhibit non-differentiability of the objective function are well known in the optimization literature since they arise in several contexts (e.g., Lagrangian duality). In these non-differentiable optimization problems the gradient is substituted by new entities called subgradients [3]. Unfortunately, to be able to define and use subgradients the objective function needs to be convex (or concave). In this case some algorithms can be devised in such a way that, even though an update direction might not be an improving direction (as is the gradient), a non-improving step can be taken with a step-size aimed at minimizing the distance from the optimal solution rather than the objective function itself. Thus, generally in these problems it is very critical to choose a suitable step-size. If the step-size selection is conducted properly, the algorithm will converge to the optimal solution even though it will take some non-improving steps. In this case the convexity assumption is key in making an argument on how to select a step-size properly [2,3,21,62,63]. In the case of non-differentiable FLSs the use of right and/or left derivatives would yield some subgradients, if the objective function were convex. Therefore, we cannot directly use the theory of non-differentiable optimization, but there are some similarities that may enable us to use some of the strategies developed in that context. It is also interesting to note that the conditions that the step-size needs to satisfy in order to theoretically prove convergence of subgradient algorithms [3,62,63] are very similar to the ones required by the SGA [20,66].

2.6.2 Step size

The step-size practically used in the training of a FLS is almost always a constant [1,4,16,24,30,31,46,51,52,75,76,82,83,85,86] even though this has numerous drawbacks. From a theoretical point of view in the pattern-by-pattern approach the step-size needs to satisfy some conditions (that a constant step-size does not satisfy) for the SGA approximation to hold. Moreover, a constant step-size might produce an oscillatory behavior in the final stages of the optimization; since, depending on the problem scaling, the parameters might end up oscillating around the optimum solution. A practical solution is to select a very small step-size. In this case the oscillatory behavior would still depend on the scaling of the problem and could not be guaranteed to disappear, even though that would likely happen if the step size were small enough. The initial phases of the algorithm would yield a decrease in objective function value smaller than that with a higher step-size. This constitutes a significant difference since most of the objective function decrease is generally achieved in the first stages of the optimization. Most of the approaches reported in the literature employ very small step sizes (from 10^{-1} to 10^{-4}), thus seriously affecting the convergence speed of these algorithms.

For the above reasons it seems necessary to use an adaptive step-size selection algorithm. The literature presents few examples of adaptive step-size algorithms applied to the design optimization of fuzzy logic systems [26,27,28,49,57]. These approaches revisit the quick-prop algorithm borrowed from the neural network literature [20]. In this algorithm the learning rate (i.e., step-size) is modified according to the convergence behavior of the objective function. A decreasing objective function sequence indicates that the algorithm is not overstepping and thus an increase in step-size might accelerate convergence. Conversely, an oscillatory behavior of the objective function values might indicate overstepping and thus the step-size should be reduced. Interestingly enough, the use of an adaptive step size is connected to non-differentiable programming, where these adaptive step-size selection algorithms proliferate [3,2,21,62,63]. Moreover, the type of increase/decrease approach for the step-size is somewhat similar in both fields.

2.6.3 Pattern-by-pattern versus batch training

A considerable amount of training methods employ pattern-by-pattern training rather than batch training [1,4,14,16,19,24,26,27,28,30,46,51,52,57,75,76,82,83,85,86]. Indeed, with neural networks it was the creation of the pattern-by-pattern training approach that stimulated a lot of applications of neural networks. In the pattern-by-pattern approach a more myopic view of the error function is adopted. The gradients of the instantaneous errors are used as an approximation to the gradient of the mean square error in order to update the parameter values. Making a small update in the instantaneous gradient direction, a parameter update is implemented in a way that should not significantly differ from a batch training update.

The gradient information in a batch training approach is very “rich” since that is the true gradient of the objective function that we seek to minimize. The reason to resort to a pattern-by-pattern approach is purely computational. However, even though the gradient in batch training is more informative, the step size to choose in this direction is unknown. This problem could be approached by employing a line search method, for example, such as a quadratic search [3]. In this case there would be an increased cost in terms of function (i.e., MSE) evaluations that are very expensive for real problems, since their cost depends on the, generally very large, number of training data. A pragmatic solution could be to only move by a small step-size along the batch gradient direction. This would again cost a lot in computations since it would significantly slow algorithmic convergence (for N training data, one batch training update corresponds to N pattern-by-pattern updates) by imposing very little steps. Conversely, using a large step-size would exhibit highly oscillatory behavior.

Therefore, a computationally “cheap” way to choose the step size along the (true) batch gradient direction could give an impulse to batch training and could carry the advantage of using a direction that contains more information about the true objective function which we seek to minimize. In a way reminiscent of what has been done in [25] in the context of response surface methodologies, in Chapter 5 we propose a limited memory quadratic fitting heuristic that can be used in conjunction with any update direction in order to select the step-size along it.

2.6.4 Global and local approaches

We have seen that of the many gradient descent approaches only some of them are "true" gradient descent ones. For these differentiable problems the application of gradient descent will lead to convergence of the approach; the algorithm will in general converge to a point of zero gradient, but not to a global optimal solution of the optimization problem. Thus, of all the approaches presented and used in the literature some are not proven to be convergent, while the remaining ones are all local. In this respect a good initialization of the tuning procedure would help in achieving a good final result. Whenever global convergence is sought the general approach is to use stochastic searches (e.g., [17,18,23,53,54,56,78]).

A true global optimization approach would advance the state of the art in many ways. First of all, a baseline for comparison could be established and used for evaluation of heuristics, online adaptation schemes, etc. A global approach could uncover the true potential of fuzzy systems, being able to express the best that a given FLS design can do. Finally, objective comparisons on different *t-norms*, membership functions, structures, etc. could be made.

A newly developed *reformulation-linearization technique* (RLT) for nonconvex global optimization [71] seems promising in this respect. The idea underlying this technique is to reformulate the original problem in order to generate successive tight linear programming relaxations. The RLT is explained in more detail in Chapter 4 along with a new supervised learning formulation. We will show that the supervised learning problem under this alternative problem formulation is polynomial in the adjustable parameters when triangular membership functions are employed; enabling us to use developed RLT techniques for polynomial nonconvex problems [69,70]. Moreover, whenever a product *t-norm* is used along with Gaussian or bell-shaped membership functions, the alternative problem formulation leads to an optimization problem factorable in the membership functions (i.e., univariate functions), thus, allowing the use of the approach in [73] based on RLT coupled with polynomial approximations. Finally, RLT also offers the possibility of solving mixed-integer zero-one problems and general discrete mixed-integer problems [71]. Thus, the optimization could be extended to the rule base as well, since it is understood that the rule base can be described by integers defining the particular rule consequent. Maybe the number of partitions and rules could be included as well, generating a combined full structure and parametric learning optimization approach.

2.6.5 Higher order methods

The use of higher order methods (i.e., Newton and quasi-Newton methods, conjugate gradient, etc.) is a natural extension of gradient descent; many contributions reference the possibility of using these methods, but few of them actually do. Their use can be helpful, especially when closing in on optimality, in order to increase the convergence rate. These methods have been successfully used in the supervised training of neural networks.

Eklund and Zhou [13] present a study on the use of higher order methods. They compare gradient descent with Gauss-Newton (GN) and Levenberg-Marquardt (LM) approaches on three problem sets. Their results seem to point to an inferiority of these second order methods with respect to gradient descent. Namely, they found that the gradient descent approach succeeds in finding better solutions than GN and LM. This result would need further study and it is contradicted by our findings in Chapter 5.

The use of suitable gradient deflection approaches might be advantageous. In the neural network literature the use of gradient deflection approaches with a constant deflection parameter is often reported as being very successful in escaping local minima of the error function [20]. A more refined gradient deflection approach might be beneficial to an increased convergence rate of supervised learning of FLSs.

It is common practice to use different (constant) step sizes for different types of parameters like membership function centers, widths and consequent constants. This approach is equivalent to pre-multiplying the gradient by a diagonal matrix with diagonal elements being the respective step-size scaling factors. Therefore, this could be considered as a particular gradient deflection approach imitating quasi-Newton methods. It is always reported that a higher step-size for the consequent parameters improves learning convergence. This is in apparent contradiction with the neuro-fuzzy analogy where it is suggested to use a smaller step-size for the output layers [20]. The motivation for the success of such choices is probably due to a second order eigenstructure inherent to the two problems. The different learning rates implement a gradient pre-conditioning that suitably rescales it. This is a first example of different problem characteristics of neural networks and FLS supervised learning problems, reinforcing the argument that the neuro-fuzzy analogy is beneficial, but that the two optimization problems are different and need to be treated as such.

2.6.6 Types of membership functions

An open topic in the research on fuzzy logic systems regards the type of membership functions to use. Common choices are gaussian, triangular and bell shaped membership functions. Moreover, as we saw in Chapter 1 we are not restricted to this type of membership function since any other unimodal function is a suitable candidate to being a membership function.

Some studies conclude that triangular membership functions with 0.5 degree of overlap are the best ones to use in FLSs, based on considerations such as convergence of control, and sensitivity to training data [44]. Moreover, triangular membership functions with 0.5 degree of overlap satisfy some optimal entropy and optimal reconstruction criteria, even though they are not necessarily the only ones [60]. An experimental study [45] compared several types of conventional as well as non-conventional (hyperbolic tangent, sinc, etc.) membership functions on many function approximation problems. This study found the sinc membership function to be the one that gives the best performance in most cases. The problem in using sinc functions as membership functions consists of the side lobes that produce negative degrees of membership that do not make sense from a fuzzy set point of view.

From an intuitive point of view, triangular membership functions are very easy to use and offer the advantage of a finite support, thus giving full control on their domain of operation. The real potential of each membership function has yet to be uncovered however, since all possible comparisons are based on local solutions to function approximation problems.

2.6.7 Readability and constraints

The non-conventional approaches of IMF-FLSs [16,30,46,51,52,57,82,83,85,86] generate a FLS that departs from the original FLSs, in terms of interpretability and linguistic meaning of the system itself. The corresponding IMF-FLS results in more control over the number of rules, easier optimization algorithms, etc. One of the main advantages of fuzzy logic (if not *the* main one) is given by readability and interpretability of the given fuzzy system; that is the way everything started and suggested that FLSs could have a high potential. Maintaining interpretability is maybe one of the most important things to keep in mind. Moreover, in the context of supervised learning it is very appealing to think of a trained FLS as of a system that will learn from data, reducing them and showing humans the synthetic (linguistic) meaning of the model hidden behind the data. The same readability problem can also be encountered with FLSs that are very big, as well as TS-FLSs where each rule has its own consequent. Moreover, the same problem can also be encountered if every possible parameter variation is allowed. Indeed, two membership functions representing small and large could end up exchanging positions and thus loosing their initial semantic meaning!

Some of the recent literature [5,28,49,64,74] looks at this problem as a trade-off between readability and performance. Thus, it is suggested that performance and readability are conflicting metrics and the trade-off depends on the purpose of the application. The main initial drive of FLSs was based on intuitive understanding of their operation; it is reasonable to ask whether readability and performance are really contrasting objectives. It might well be that the many local solutions found through gradient descent or other methods are, or might be, non-readable, while the true global solution is an interpretable one. This is along the lines of reasoning of Oliveira [58], who develops some constraints on FLSs parameters in order to achieve optimal input-output interfaces (perfect reconstruction) along with function approximation. Moreover, he also remarks that enabling these constraints might make it more likely to achieve global solutions instead of being trapped in local ones. This would be especially true if performance and readability are not conflicting objectives! Moreover, some results in comparing FLSs and IMF-FLSs report a lower error achieved with FLSs with fewer parameters than analogous IMF-FLSs [75].

The adjustable parameters ***w*** have been considered unconstrained in the derivation of gradient descent update equations at the beginning of this chapter. In reality they might have to be constrained. Indeed, the parameters related to the antecedent part of the FLS are unconstrained but some configurations might make the FLS lose its interpretability. The adjustable parameters related to the consequent part might be constrained as well, due to their physical meaning in some particular cases (e.g., control applications where they are related to control actions and the actuators might have some limits), but the literature does not offer too many examples of approaches that include constraints on the adjustable parameters (except [31]). This might be another heritage of the neuro-fuzzy "wedding" since in neural networks the network weights are generally unconstrained.

It is also observed that in the evolution of a gradient descent tuning algorithm the membership function width might become very small thus limiting the probability of further updates [22]. Conversely, membership function width might also become very large and absorb the effect of very slim membership functions, which could result in very large membership functions that cover a significant portion of the universe of discourse. Constraining the width of membership functions might reduce or eliminate the likelihood of such outcomes. It is also generally desirable to have at least a minimum degree of membership over the entire universe of discourse. In other words, for every point in the universe of discourse there should always be at least one fuzzy set to which the point belongs with a minimum ε degree of membership.

Another unexplored but interesting avenue of research consists of developing a set of constraints for ensuring readability and completeness. This would require the development of algorithms that can efficiently handle constrained problems since the use of penalty function methods might generate ill-conditioning effects. Moreover, constraints, coupled with a global optimization algorithm, might shed some light on the controversy about readability and performance.

From a point of view of interpretability IMF-FLSs do not seem a very attractive alternative to conventional FLSs; however this should be further studied in terms of performance. Moreover, the interpretability of TS-FLS could also be analyzed and the formalism of (1.25) could be beneficial in this sense, by showing a continuum between

Mamdani FLSs and TS-FLS, and understanding whether there is and, if so, where is the trade-off between readability and performance.

2.6.8 Test cases

A big problem in supervised learning of FLSs is the non-existence of standard and structured benchmark problems. Generally, the proposed approaches are tested on commonly known function approximation problems or system identification problems (very common ones are those of [47]). Unfortunately, these problems are not structured in an exact and rigorous way to enable direct comparison of performance indices. A step in this direction (by at least stating the exact experimental conditions) would ensure some standardization in the evaluation of different approaches.

2.7 Conclusions

This chapter introduced the optimization of fuzzy logic systems. We saw how supervised learning can be considered a function approximation problem, and we provided a general problem formulation. Equations for batch and pattern-by-pattern gradient descent approaches were derived for Mamdani FLSs, TS-FLSs, and IMF-FLS. Those equations were derived from a common matrix, thus yielding the first original result of this work by providing a common framework for the optimization of these three different types of fuzzy systems. This formulation shows the interesting nature, but not the need, of resorting to neural analogies.

The second half of the chapter was devoted to the review of current and past approaches to supervised learning of FLSs. Several approaches to the optimization problem were explained and discussed along with their advantages and shortcomings. The most common ones are Wang's [85], Nomura's [51] and ANFIS [26]. Finally, we concluded the chapter with a discussion on current practices, as well as the identification of open research issues. Some of these will be addressed in the following Chapters 3,4, and 5, while others remain as open questions for future research.

Chapter 3
The Effects of Non-Differentiability

3.1 Introduction

In this chapter we will show some results related to the discussion of Chapter 2 on non-differentiability of FLSs using piecewise linear membership functions or min—max operators. In these cases the output of the FLS is not differentiable (everywhere, with respect to the adjustable parameters), i.e., there are some points in parameter space where right and left derivatives exist, but have different values. In this case a gradient descent approach (as well as any gradient-based method) cannot be applied since the gradient does not always exist. For this reason the application of a *false* gradient descent (i.e., a gradient descent where the derivatives are arbitrarily defined at the points where they do not exist) as presented in the literature [16,31,46,49,51,52,74,75] will not offer any guarantees of convergence (even to a point of zero gradient). The compounding of non-improving steps may cause the algorithm to diverge.

From a pragmatic point of view, the non-differentiable nature of the problem might not be a concern since the probability of falling onto points of non-differentiability is zero, and, of course, one could always use a non-gradient based approach. Given some training data, constituting a grid in the input space, the membership functions, for example, move along this grid during tuning, and might fall onto one of the points or at least arbitrarily close to it. Therefore, the presence of these "glitches" might make the algorithm converge in a slower fashion when not generating divergent behavior.

We will show some of the problems introduced by the non-differentiability of triangular membership functions through two FLSs for function approximation. Section 3.2 illustrates how the convergence in the training of a TS-FLS for a one dimensional function approximation task becomes slower as the adjustable parameters get closer to points of non-differentiability [12]. Section 3.3 shows divergence of the algorithm in the training of a Mamdani FLS for a two dimensional function approximation problem [11].

3.2 Example 1: A TS-FLS for SISO Function Approximation

In this first example we consider a single-input single-output TS-FLS with asymmetrical antecedent membership functions, singleton consequents and product inference. We will derive the gradient descent equations and show some simulation results. This is an expanded version of the work presented in [12].

3.2.1 Problem formulation

The *l*-th rule for the given FLS will be:

$$R^{(l)}:\ \text{IF } x \text{ is } A_l,\ \text{THEN } y = \delta_l \tag{3.1}$$

The output of a single input single output TS-FLS can be expressed by (1.25) along with (1.26). Specializing these equations for a single input case and for constant local models (consequents) we have

$$y = \frac{\sum_{l=1}^{R} \mu_l(x)\delta_l}{\sum_{l=1}^{R} \mu_l(x)} \tag{3.2}$$

In (3.2) R is the number of rules and also the number of membership functions describing the (single) input. Moreover, $\mu_i(x)$ represents the membership function for the linguistic term A_i. The membership functions are triangular and a 0.5 level of overlap is always ensured. This is accomplished by parameterizing the membership functions only through their centers and the centers of the two neighboring membership functions. Figure 3.1

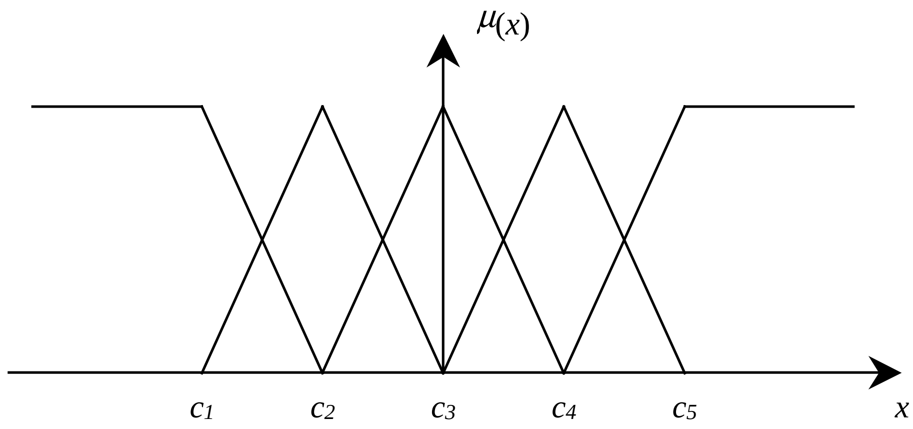

Figure 3.1. Membership functions for Example 1

shows a graphical representation of this type of membership functions for $R = 5$. Note that the extreme left and extreme right membership functions are *shouldered* (i.e., they are saturated at the unit membership degree). Let us introduce the asymmetrical triangular function $tr(a,b,c;x)$:

$$tr(a,b,c;x) = \begin{cases} \dfrac{x-a}{b-a} & if\ a \le x \le b \\ 0 & if\ x < a \vee x > c \\ \dfrac{x-c}{b-c} & if\ b < x \le c \end{cases} \tag{3.3}$$

Using (3.3) the membership functions for the FLS in this example will be

$$\begin{aligned} &\mu_1(x) = tr(-\infty, c_1, c_2; x) \\ &\mu_i(x) = tr(c_{i-1}, c_i, c_{i+1}; x) \quad i = 2,3,\ldots,R-1 \\ &\mu_R(x) = tr(c_{R-1}, c_R, +\infty; x) \end{aligned} \tag{3.4}$$

With this particular choice of membership functions at any point in the input space X the sum of membership degrees of all the fuzzy sets partitioning X is unity, that is

$$\sum_{i=1}^{R} \mu_i(x) = 1 \quad \forall x \in X \tag{3.5}$$

This simplifies the denominator of Equation (3.2); substituting (3.5) in (3.2) we obtain the output of the FLS as a linear combination of the consequent constants with the membership degrees as coefficients:

$$y = \sum_{l=1}^{R} \mu_l(x)\delta_l \tag{3.6}$$

Every membership function is defined by its center (and the two adjacent centers), while every rule has its own consequent. Thus, this model is completely specified with a total of 2*R* parameters: *R* parameters define the antecedent fuzzy sets, and *R* parameters define the consequent constants. Using the supervised learning framework introduced in Chapter 2, we can derive pattern-by-pattern training update equations. Given the datum (x_i, y_{di}) (i = 1,2, …, N), we wish to minimize the instantaneous errors:

$$E_i(\boldsymbol{c},\boldsymbol{\delta}) = \frac{1}{2}e_i^2 \quad e_i = y(x_i,\boldsymbol{c},\boldsymbol{\delta}) - y_{di} \tag{3.7}$$

For a generic parameter *w* the update rule based on gradient descent is (2.7):

$$w^{new} = w^{old} - \eta e_i \frac{\partial y}{\partial w} \tag{3.8}$$

where η is the learning rate (i.e., step size). When *w* is one of the consequent parameters δ_i we obtain

$$\frac{\partial y}{\partial \delta_i} = \mu_i(x) \tag{3.9}$$

Using (3.6) and remembering that every center (c_i) contributes to three membership functions (except for the centers of the rightmost and leftmost membership), we can compute the derivative of the output of the FLS with respect to the antecedent parameters

$$\begin{gathered} \frac{\partial y}{\partial c_1} = \sum_{j=0}^{1} p_{j+1} \frac{\partial \mu_{j+1}(x)}{\partial c_1} \qquad \frac{\partial y}{\partial c_R} = \sum_{j=-1}^{0} p_{R+j} \frac{\partial \mu_{R+j}(x)}{\partial c_R} \\ \frac{\partial y}{\partial c_i} = \sum_{j=-1}^{1} p_{i+j} \frac{\partial \mu_{i+j}(x)}{\partial c_i} \quad i = 2,3,\ldots,R-1 \end{gathered} \tag{3.10}$$

To complete the derivation of the pattern-by-pattern update equations, we need the derivatives of the membership functions with respect to their three parameters. By straightforward derivation of (3.3) we obtain

$$\begin{gathered}
\frac{\partial tr(a,b,c;x)}{\partial a}=\begin{cases} 0 & if \;\; x<a \vee x\geq b \\ \dfrac{x-b}{(b-a)^2} & if \;\; a<x<b \end{cases} \\
\frac{\partial tr(a,b,c;x)}{\partial b}=\begin{cases} \dfrac{a-x}{(b-a)^2} & if \;\; a\leq x<b \\ 0 & if \;\; x<a \vee x\geq c \\ \dfrac{c-x}{(b-c)^2} & if \;\; b<x<c \end{cases} \\
\frac{\partial tr(a,b,c;x)}{\partial c}=\begin{cases} 0 & if \;\; x\leq b \vee x>c \\ \dfrac{x-b}{(b-c)^2} & if \;\; b\leq x<c \end{cases}
\end{gathered} \tag{3.11}$$

The derivatives for the rightmost and leftmost membership functions μ_1 and μ_R can be obtained by the same equations (3.11) at the limit for either $a \rightarrow -\infty$ or $c \rightarrow +\infty$, respectively.

Equations (3.11) are still not operational for the application of the gradient descent algorithm because in this case we cannot apply a "true" gradient descent algorithm since the membership functions are not differentiable with respect to the adjustable parameters $\boldsymbol{c}$. We can now more clearly see the problem; in each of the equations (3.11) the derivative is not defined for some value of x (i.e., respectively, $x = a$, $x = b$, and $x = c$). In order to be able to apply some algorithm using these derivatives we need to define a value for them at the undefined points, but doing this we are not defining a derivative anymore. We will proceed, with the hope of making this example beneficial to explaining what are the risks in not considering the non-differentiability of FLSs. It seems reasonable to use either the right or the left derivative value. Moreover, following the rationale used in Nomura *et al.* [52], we will choose a 0 value over a non-zero one. There is only one case (second equation) where we have to choose between two nonzero values and we arbitrarily choose one of them (the right one). With these choices we can loosely rewrite the "derivatives" (not *true* derivatives) as

$$\frac{\partial tr(a,b,c;x)}{\partial a}=\begin{cases}0 & if \quad x\le a\vee x\ge b\\ \dfrac{x-b}{(b-a)^2} & if \quad a<x<b\end{cases}$$
$$\frac{\partial tr(a,b,c;x)}{\partial b}=\begin{cases}\dfrac{a-x}{(b-a)^2} & if \quad a\le x<b\\ 0 & if \quad x<a\vee x\ge c\\ \dfrac{c-x}{(b-c)^2} & if \quad b\le x<c\end{cases} \tag{3.12}$$
$$\frac{\partial tr(a,b,c;x)}{\partial c}=\begin{cases}0 & if \quad x\le b\vee x\ge c\\ \dfrac{x-b}{(b-c)^2} & if \quad b\le x<c\end{cases}$$

We can now state the pattern-by-pattern “gradient descent” tuning algorithm as follows:

1. Initialize antecedent and consequent parameters;
2. Start with the first data point, i = 1;
3. Compute the FLS output using (3.6) along with (3.3) and (3.4);
4. Compute the instantaneous approximation error and the corresponding updates using (3.7) along with (3.8), (3.9), (3.10) and (3.12);
5. If there are still some data points ($i < N$) increase i (= i + 1) and go back to step 3;
6. Otherwise ($i = N$), compute the mean square error (average of the instantaneous errors in (3.7)) and verify if a termination criterion is met, either stop or go to the next step;
7. *Shuffle* the data points (mix their order) and go back to step 2.

In the above algorithm the termination criterion could be an error goal or a maximum number of epochs or both.

3.2.2 Results and Discussion

We choose R = 5 (10 adjustable parameters) to approximate a simple parabola, $y = x^2$ in the interval [-1,1]. We fix the maximum number of epochs to 300 (large enough to yield convergence of the algorithm) in order to compare the performance of the algorithm in a few different situations at fixed computation time.

With these choices, the problem is nicely scaled and the parabola can be easily approximated by the FLS (it has many more parameters than needed to describe a parabola). For example, we pick a constant learning rate $\eta = 0.1$, the training data are 21 points evenly distributed in [-1,1], the c_i parameters are initially evenly spaced in [-1,1], and the δ_i parameters are initially all set equal to 0.5. After 300 epochs the algorithm converges to a mean square error of 4×10^{-4}. Figure 3.2 shows the convergence history for the mean square error (a), along with the approximation error (b) in [-1,1], and the final position of the antecedent membership functions (c). Both the final value of the MSE and the approximation error show the capability of the system to approximate the given function.

In order to highlight the problems due to the non-differentiability of triangular membership functions, we slightly modify the problem with an *ad-hoc* selection of the training data to stimulate the effects of the non-differentiability using training data that fall

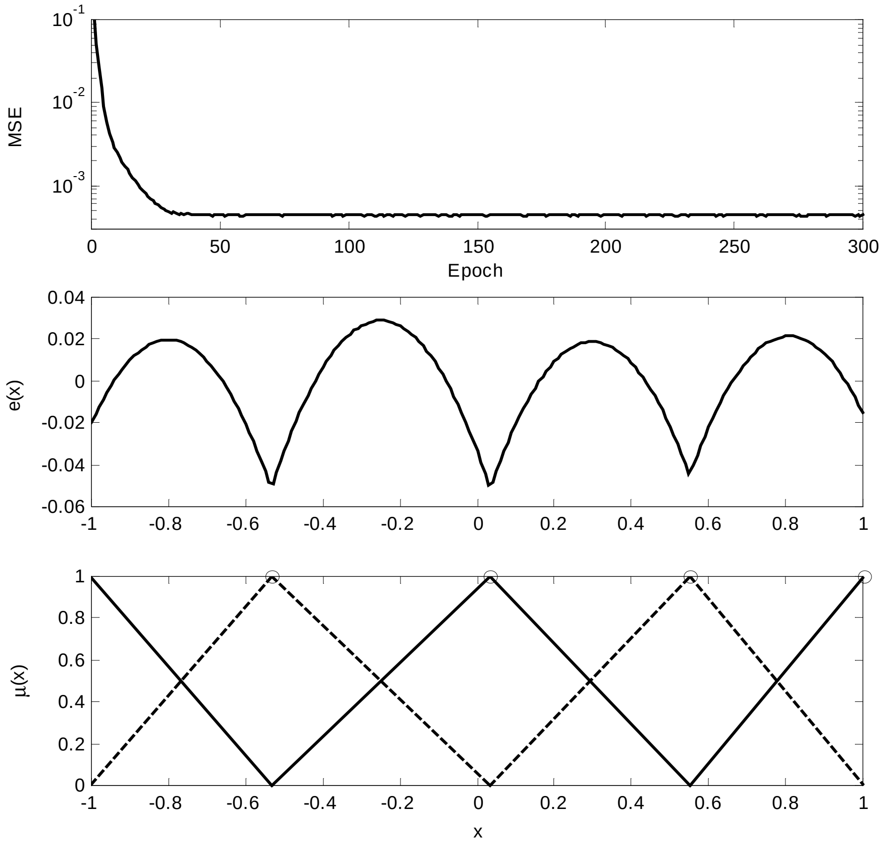

Figure 3.2. (a) Convergence history of the mean square error, (b) approximation error, (c) antecedent membership functions

exactly at the non-differentiability points. Thus, the training set is composed of eleven evenly spaced points in [-1,1]; those points do not change in each epoch. In each epoch we also introduce five additional training points x_i equal to the c_i values at the particular iteration. Therefore, we are using some points in the training data where the objective function is non-differentiable with respect to its parameters. If we were using a true gradient descent approach and a small enough step-size, we should always observe a decrease in *instantaneous* error at every step (not epoch) provided that the constant step size is small enough. For this reason $\eta = 0.01$ was chosen, in order to exclude (or limit) non-improving steps due to overstepping (i.e., a learning rate that is too big). Moreover, since this is not a true gradient descent, at some steps we might observe an increase in error function that is only due to the non-improving nature of the (non-gradient) direction that was taken, and not due to overstepping. Any non-improving step is checked using a very small step-size (10^{-10}), in order to verify the reason for the non-improvement. Note that the strange approach taken to generate training data that fall exactly at the non-differentiable points can be regarded as an on-line approach where the training points are randomly sampled, or in general, are generated by a process that is out of our control. Moreover, in principle, we should not be restricted by the training data that we use (assuming that they are informative enough).

Using a random initialization for antecedent and consequent parameters, we executed the algorithm obtaining the convergence history reported in Fig. 3.3 with the dashed line (b). The mean square error always improves at every epoch, and the final mean square error is 0.01. Five steps per epoch were taken along non-gradient directions, for 300 epochs, thus giving a total of 1500 possible non-improving steps. The nature of these directions (whether improving or not) depends of the particular problem at hand and of all its characteristics. Thus, it is not easy to predict whether a direction will be improving the (instantaneous) error, or not. On the other hand, we know that the direction of the negative gradient always yields an initial decrease in objective function value. In this case, of the 1500 total possibilities there were 145 non-improving steps (i.e., where the instantaneous error did not improve) of which 4 were due to overstepping (the direction was a descent direction). Thus, roughly 10% of the non-gradient directions turned out to be non-improving. This, added to

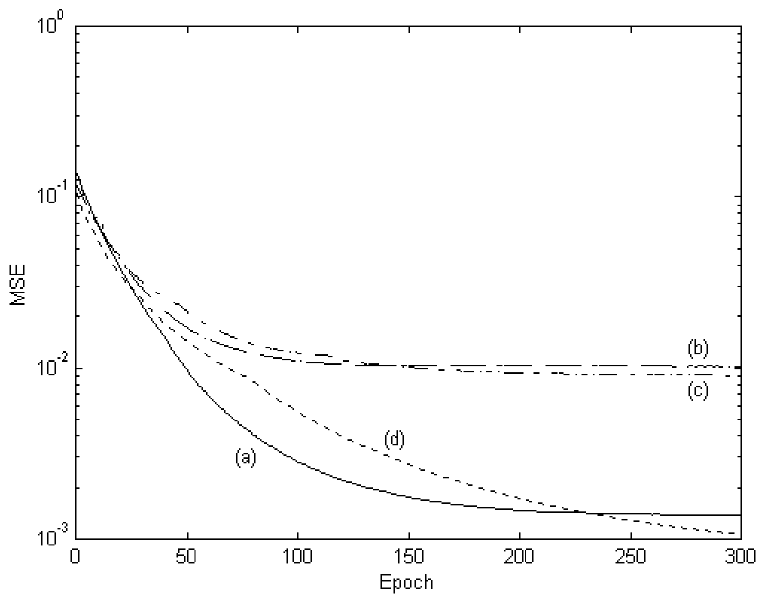

Figure 3.3. Convergence history for: (a) $\sigma = 0.1$, (b) non-differentiable case $\sigma = 0$, (c) $\sigma = 0.01$, and (d) differentiable case with same training data as (b)

the fact that hitting by accident one of these non-differentiable points is a very remote event, seems to make this case practically resilient to having big troubles due to the non-differentiability.

In order to understand what are the (practical) damages induced by the non-differentiability, a second experiment was performed. In this experiment the conditions are exactly the same as the previous case, with the only difference being that the dynamically generated training data that are not positioned exactly at $x = c_i$. They are positioned at $x = c_i + \sigma$, where σ is a small parameter that we will call a *detuning* parameter. Its effect is to introduce a small detuning from the non-differentiability points, thus making the problem differentiable, and the approach a true gradient descent. The value $\sigma = 0.1$ was chosen with exactly the same initialization (random with same seed) as in the previous case. Note that the previous non-differentiable approach used data generated in this same way with $\sigma = 0$. The rationale for this approach is to generate a case analogous to the previous experiment, trying to understand the problems caused by non-differentiability and also by proximity to it. The final mean square error for this case ($\sigma = 0.1$) was 0.0014; that is, one order of magnitude smaller than the error obtained in the previous case. The solid line (a) of

Fig. 3.3 shows the convergence history of the mean square error. In the entire evolution of the algorithm there was only one non-improving step and overstepping caused it.

Comparing the errors for these two approaches (with and without detuning), we cannot be entirely sure whether the different convergence history is caused by the non-differentiability or by the different training data sets because (some of) the training data are generated according to the evolution of the antecedent parameters. Since the parameters evolve differently in the two previous cases, the data that are generated are also different. To investigate this effect, another experiment with same initialization and same data points of the first (non-differentiable) case was conducted. The only difference is that the data set of the first epoch was exchanged with the last one, since if even the first data set were the same then the algorithm would evolve exactly the same way. Thus, besides one minor ordering modification for the first epoch, all the other conditions are the same. This case converges to a final mean square error of 10^{-3}, with a convergence history shown by the dotted line (d) of Fig. 3.3. Therefore, training with the same data as the non-differentiable case (b) leads to convergence that is as good as the detuned case (a), and even a little better. It has a smaller final error and still has a non-zero error rate. Things were not supposed to be exactly the same; the same qualitative behavior shows that the worse performance of the non-differentiable case shown in curve (b) is not due to non-rich training data, but it is due to updates in non-improving directions (combination of training data and parameter values) generated by non-differentiable conditions.

Another detuned case with *detuning* parameter $\sigma = 0.01$ (and same initialization) was also considered. The final mean square error was $9{\cdot}10^{-3}$ and the convergence history is shown by the dash-dot line (c) in Fig. 3.3. Note that, even though all the cases shown in Fig. 3.3 are initialized exactly the same way, there is a small difference in initial errors due to the slightly different (initial) training data. As we can see from Fig. 3.3, there is a difference from the differentiable (a) to the non-differentiable (b) case that is not due to the different training data as shown by the behavior of line (d), very similar to (a). The convergence curve (c) points to the fact that for $\sigma \rightarrow 0$ (a) approaches (b); thus, (c) is just an intermediate case. In the evolution of (c), obtained for $\sigma = 0.01$, the instantaneous error increases 44 times <u>because of overstepping</u>. Thus a possible explanation is that in non-differentiable conditions (i.e., σ = 0) there are some non-improving directions; approaching non-differentiable conditions the problem is differentiable and thus improving directions are generated, but the range of step-sizes that offer an improvement becomes smaller and smaller. Moreover, the amount of objective function decrease along the negative gradient direction becomes smaller and smaller as σ approaches zero, so that for σ = 0 there is a smooth transition to a non-improving direction.

Some experiments were conducted to ascertain the influence of σ (closeness to non-differentiable points) on convergence. Training of the FLS with several values of σ was attempted and the corresponding learning curves were compared as shown in Fig. 3.4 along with the corresponding values of σ = 0.001, σ = 0.01, σ = 0.02, σ = 0.05, σ = 0.08, and σ = 0.1. The convergence curves for the MSE vary smoothly from σ = 0.1 to σ = 0.001 orderly filling the space between the curves obtained for σ = 0.001 and σ = 0.1. Given the difference in training data for each case, the order is not exactly satisfied at every epoch, but the plot shows a nice and orderly qualitative behavior exhibiting a main effect due to the detuning parameter σ. The closer the training data are to the points of non-differentiability, the higher the final mean square error, and thus the slower the algorithm.

The motivation for, and the mechanism underlying this behavior were studied

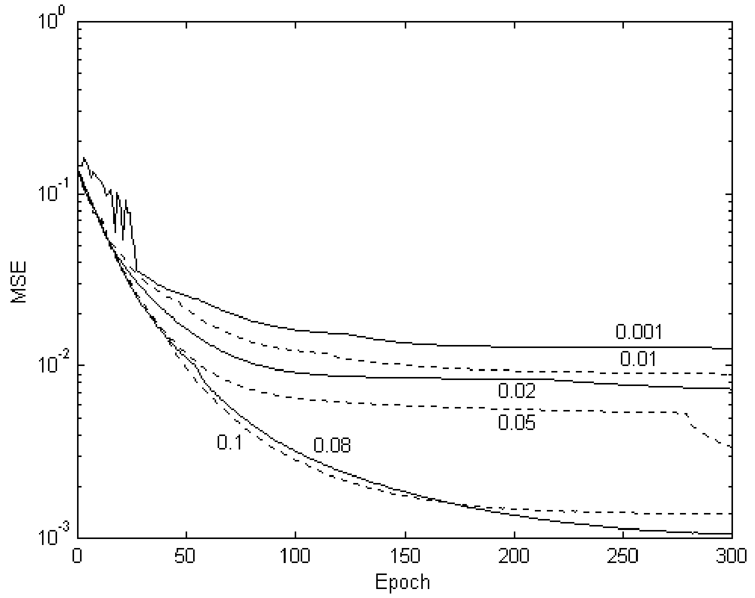

Figure 3.4 Convergence history for varying σ

examining the instantaneous error as a function of the step-size at arbitrary points in parameter space and for different values of the detuning parameter σ. The solid line of Fig. 3.5 shows the situation at one of the non-differentiable points (σ = 0). The instantaneous error $E_i(\boldsymbol{c},\delta)$ is shown as a function of the step size, η along the update (non-gradient) direction is depicted. The plot was generated using an initial step size equal to zero, and then 50 step sizes logarithmically spaced between 10^{-10} and 10^{-1}. The update direction is indeed a non-improving direction, the instantaneous error strictly increases at least for $\eta \in [0, 0.1]$. The same situation holds for all the other 141 non-improving directions in the non-differentiable case.

The directions generated for small non-zero values of σ are improving directions (since they are true gradient directions) but the interval for improvement might change with σ. In Fig. 3.5 we also show the behavior of the instantaneous error at one point in parameter space as a function of the step-size along the negative gradient direction for σ = 0.005, σ = 0.01, and σ = 0.02 (dotted lines). It can be seen that as the detuning parameter, σ, decreases, the interval of step-sizes corresponding to improvement in error shrinks, and the amount of decrease in error gets smaller. At the limit these curves converge to a non-improving direction corresponding to the non-differentiable case (solid line, σ = 0).

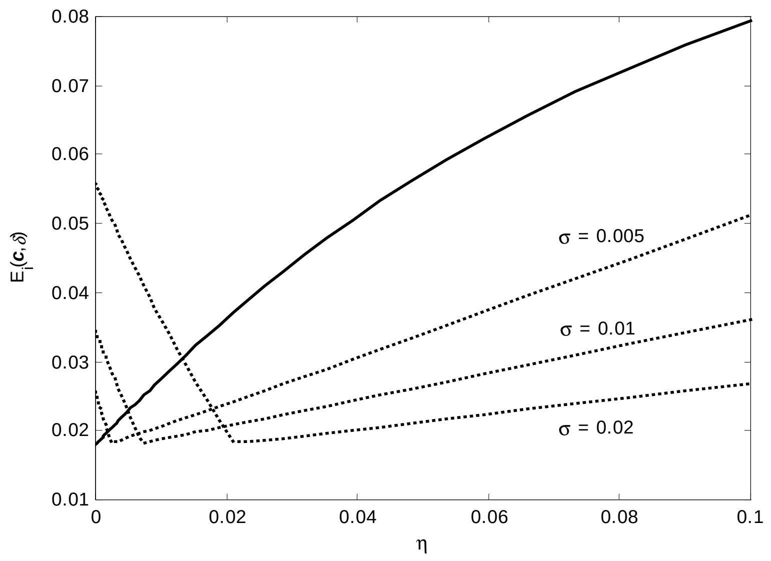

Figure 3.5. Behavior of the error function along the "gradient direction" for different values of σ

The optimal step size observed in the detuned cases of Fig. 3.5 (as well as in other analogous cases) corresponds to moving one parameter almost exactly on to the point of non-differentiability. This corresponds to moving the adjustable parameter (i.e., center of the triangular membership function) on top of one of the training data. This is a natural tendency of the training algorithm since with a suitable value of the consequents the approximation error at this point will be zero (see for example the "notches" in the approximation error in Fig. 3.2b, corresponding to the position of the centers of the membership functions). The optimal step-size is not such that one of the parameters will fall exactly at the point of non-differentiability since there is probably a trade-off with the error decrease obtained by further changes in the remaining parameters; one of the parameters turns out to be very close to a point of non-differentiability when an optimal step-size is used. This is also intuitive due to the fact that the gradient is offering a linear local approximation to the objective function. At a point of non-differentiability the objective function radically changes, and the linear local approximation breaks. This is the real problem that the mathematical abstraction of non-differentiability raises. There is a sudden variation in first derivative in a very small (i.e., infinitesimal) interval. Even if the membership functions were made differentiable with respect to the adjustable parameters by replacing the non-differentiable "tip" with a differentiable one, but still retaining a sudden change in derivative value over a small (finite) interval, we would have similar problems due to possible overstepping of the algorithm. Indeed, the abrupt variation in first derivative value emphasizes the shortsighted nature of gradient descent that is based on a local approximation of the objective function.

The non-differentiability raises theoretical concerns regarding the possible divergence of the algorithm, but it also creates a pragmatic concern since it might slow down the algorithm due to the overtaking of non-improving steps (due to overstepping) caused by excessive closeness to the points of non-differentiability. Moreover, in any type of application, regardless of the type of training data used (random or fixed) the membership functions move along the universe of discourse during the tuning, thus making it possible for them to get arbitrarily close to points of non-differentiability.

3.3 Example 2: A Mamdani FLS for MISO function approximation

In this second example we employ a Mamdani FLS for a simple function approximation task. The FLS is a two-input one-output system with singleton consequents, triangular antecedent membership functions, and product inference. We will derive the gradient descent equations and show some simulation results illustrating the divergence of the algorithm. This is an expanded version of the work presented in [11].

3.3.1 Problem formulation

The input space is partitioned by three membership functions for each input, while there are five consequent membership functions (singletons) defined on the output space. The leftmost and rightmost input membership functions are shouldered, and all the linear parts have unit slope. The input membership functions are shown in Fig. 3.6. The membership functions can be expressed by:

$$\mu_{i1}(x,\theta_i)=\begin{cases} 1 & if \;\; x\le\theta_i-1 \\ \theta_i-x & if \;\; \theta_i-1<x<\theta_i \\ 0 & if \;\; x\ge\theta_i \end{cases}$$
$$\mu_{i2}(x,\theta_i)=\begin{cases} 0 & if \;\; x\le\theta_i-1\vee x\ge\theta_i+1 \\ x-\theta_i+1 & if \;\; \theta_i-1<x\le\theta_i \\ \theta_i+1-x & if \;\; \theta_i<x\le\theta_i+1 \end{cases} \tag{3.13}$$
$$\mu_{i3}(x,\theta_i)=\begin{cases} 0 & if \;\; x\le\theta_i \\ x-\theta_i & if \;\; \theta_i<x<\theta_i+1 \\ 1 & if \;\; x\ge\theta_i+1 \end{cases}$$

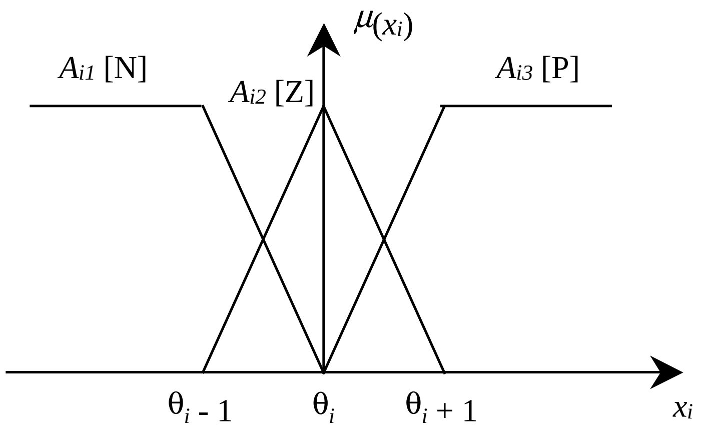

Figure 3.6. Membership functions for Example 2 (i = 1,2)

This choice ensures a constant completeness level of 0.5, moreover, at any point in the universe of discourse the sum of membership values is always 1. That is:

$$\sum_{j=1}^{3} \mu_{ij}(x) = 1 \quad \forall x \in \Re \quad i = 1,2 \tag{3.14}$$

The rule table for this fuzzy system is predefined by an expert (the author) and is given in Table 3.1. By definition of the proper terms and application of (1.25) the output of the corresponding fuzzy system is

Table 3.1. Rule base for Example 2

$x_1 \rightarrow$ / $x_2 \downarrow$	N [A_{11}]	Z [A_{12}]	P [A_{13}]
N [A_{21}]	NL [δ_1]	NS [δ_2]	ZE [δ_3]
Z [A_{22}]	NS [δ_2]	ZE [δ_3]	PS [δ_4]
P [A_{23}]	ZE [δ_3]	PS [δ_4]	PL [δ_5]

$$y(\boldsymbol{x},\boldsymbol{\theta},\boldsymbol{\delta}) = \frac{\sum_{i=1}^{3}\mu_{1i}(x_1)\sum_{j=1}^{3}\mu_{2j}(x_2)\delta_{j+i-1}}{\sum_{i=1}^{3}\mu_{1i}(x_1)\sum_{j=1}^{3}\mu_{2j}(x_2)} \tag{3.15}$$

Substituting (3.14) in (3.15) the denominator adds up to unity and thus disappears. We can thus rewrite (3.15) also exchanging the summations:

$$y(\boldsymbol{x},\boldsymbol{\theta},\boldsymbol{\delta}) = \sum_{i=1}^{3}\mu_{1i}(x_1)\sum_{j=1}^{3}\mu_{2j}(x_2)\delta_{j+i-1} \tag{3.16a}$$

$$y(\boldsymbol{x},\boldsymbol{\theta},\boldsymbol{\delta}) = \sum_{j=1}^{3}\mu_{2j}(x_2)\sum_{i=1}^{3}\mu_{1i}(x_1)\delta_{j+i-1} \tag{3.16b}$$

We fix the consequent parameters at: δ = [-2 –1 0 1 2]. To tune the antecedents (two parameters; this way we will be able to visualize the data) a pattern-by-pattern approach is used, in which the update of the two adjustable parameters (θ_1 and θ_2) is given by (3.8). In order to completely formulate the training algorithm we need to compute the derivative of the fuzzy output with respect to the adjustable parameters. Using the property that θ_j defines only the membership functions $\mu_{ji}(x)$, j = 1,2 and i = 1,2,3, we can use Equations (3.16a) and (3.16b) to compute the partial derivatives of y with respect to θ_1 and θ_2, respectively. We obtain:

$$\frac{\partial y(\boldsymbol{x},\boldsymbol{\theta})}{\partial\theta_1} = \sum_{i=1}^{3}\frac{\partial\mu_{1i}(x_1)}{\partial\theta_1}\sum_{j=1}^{3}\mu_{2j}(x_2)\delta_{j+i-1} \tag{3.17a}$$

$$\frac{\partial y(\boldsymbol{x},\boldsymbol{\theta})}{\partial\theta_2} = \sum_{j=1}^{3}\frac{\partial\mu_{2j}(x_2)}{\partial\theta_2}\sum_{i=1}^{3}\mu_{1i}(x_1)\delta_{j+i-1} \tag{3.17b}$$

The partial derivatives of the membership functions with respect to the θ parameters can be obtained by straightforward derivation of (3.13):

$$\begin{aligned}
\frac{\partial\mu_{i1}(x,\theta_i)}{\partial\theta_i} &= \begin{cases} 0 & if \;\; x<\theta_i-1 \\ 1 & if \;\; \theta_i-1<x<\theta_i \\ 0 & if \;\; x>\theta_i \end{cases} \\
\frac{\partial\mu_{i2}(x,\theta_i)}{\partial\theta_i} &= \begin{cases} 0 & if \;\; x<\theta_i-1 \vee x>\theta_i+1 \\ -1 & if \;\; \theta_i-1<x<\theta_i \\ 1 & if \;\; \theta_i<x<\theta_i+1 \end{cases} \\
\frac{\partial\mu_{i3}(x,\theta_i)}{\partial\theta_i} &= \begin{cases} 0 & if \;\; x<\theta_i \\ -1 & if \;\; \theta_i<x<\theta_i+1 \\ 0 & if \;\; x>\theta_i+1 \end{cases}
\end{aligned} \tag{3.18}$$

Equation (3.18) is probably the best example to illustrate the discontinuity in the derivative since in this case the derivative is piecewise constant. We are faced with the same problem as in the previous Example 1: we need to define what happens at the points of non-differentiability, that is, we need to decide what value we will use to determine the update direction at these points. Guided by the same approach as in the previous case (this is not the only possibility), we can choose null values over non-zero ones, thus revising (3.18) in (3.19) as follows[1]:

$$\begin{aligned}
\frac{\partial\mu_{i1}(x,\theta_i)}{\partial\theta_i} &= \begin{cases} 0 & if \;\; x\le\theta_i-1 \\ 1 & if \;\; \theta_i-1<x<\theta_i \\ 0 & if \;\; x\ge\theta_i \end{cases} \\
\frac{\partial\mu_{i2}(x,\theta_i)}{\partial\theta_i} &= \begin{cases} 0 & if \;\; x\le\theta_i-1 \vee x\ge\theta_i+1 \\ -1 & if \;\; \theta_i-1<x<\theta_i \\ 1 & if \;\; \theta_i\le x<\theta_i+1 \end{cases} \\
\frac{\partial\mu_{i3}(x,\theta_i)}{\partial\theta_i} &= \begin{cases} 0 & if \;\; x\le\theta_i \\ -1 & if \;\; \theta_i<x<\theta_i+1 \\ 0 & if \;\; x\ge\theta_i+1 \end{cases}
\end{aligned} \tag{3.19}$$

Given an input to the FLS we can use (3.13) and (3.16) to determine the corresponding output. Moreover, given some training data we can compute pattern-by-

[1] Note that in Equation (3.19) the *derivative* symbol is loosely meant since we know the derivative does *not* exist at the points we are arbitrarily defining it.

pattern "gradient-descent" updates for the two parameters (θ_1 and θ_2) using (3.8) along with (3.17) and (3.19).

3.3.2 Results and Discussion

As a very simple example we will try to approximate a plane described by the equation $y = x_1 + x_2$ in the square [-1,1]×[-1,1]. The consequent parameters are fixed and their value is: δ = [-2 –1 0 1 2]. With the given choice of rule base and δ, a (global) optimal solution to this problem is $\theta_1 = \theta_2 = 0$. The FLS surface in this case is illustrated in Fig. 3.7. The surface is identical to the given plane in the desired approximation range.

From (3.18) we can see that for a fixed value of $\boldsymbol{x}$ the non-differentiable points are given by $\theta = \boldsymbol{x}$ and all the possible combinations obtained by adding or subtracting 1 from each component of θ. Thus, we can see in this case that there is a more dense grid of non-differentiable points. In the same fashion of the previous example we tried to stimulate the effects of the non-differentiability by using training points that fall exactly at the non-differentiability points. The training data were chosen such that one of the two components of $\boldsymbol{x}$ (which one is chosen randomly) is equal to the corresponding component of θ at that iteration, while the other component of $\boldsymbol{x}$ is chosen randomly in [-1,1]. A constant learning

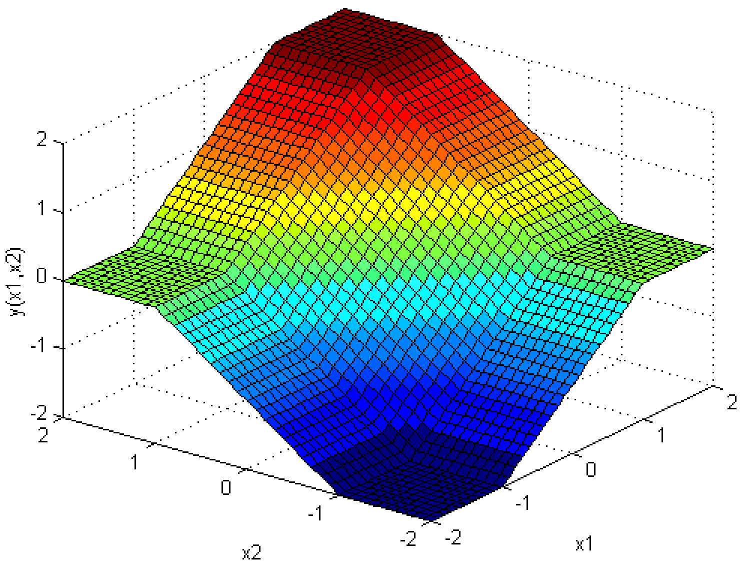

Figure 3.7. FLS surface for the optimal setting of parameters ($\theta_1 = \theta_2 = 0$)

rate η = 0.1 was used along with a maximum number of 200 iterations. The initial value of the parameters θ is chosen randomly. In the simulation we tracked the instantaneous error at each iteration (i.e., presentation of a datum) and we computed the effect of the update on the instantaneous error.

In many cases the algorithm does not converge smoothly and the parameters have an erratic behavior. Moreover, in some cases the parameters as well as the total error diverge! One of such cases is shown in Fig. 3.8 where we show the history of instantaneous errors, mean square error, and the evolution of the algorithm in parameter space. The full triangle in Fig. 3.8c indicates the initial point, while the full square indicates the optimal solution: the MSE diverges as well as the parameters. The algorithm is diverging because it is (almost) always following non-improving directions (Fig. 3.8a). The construction of the

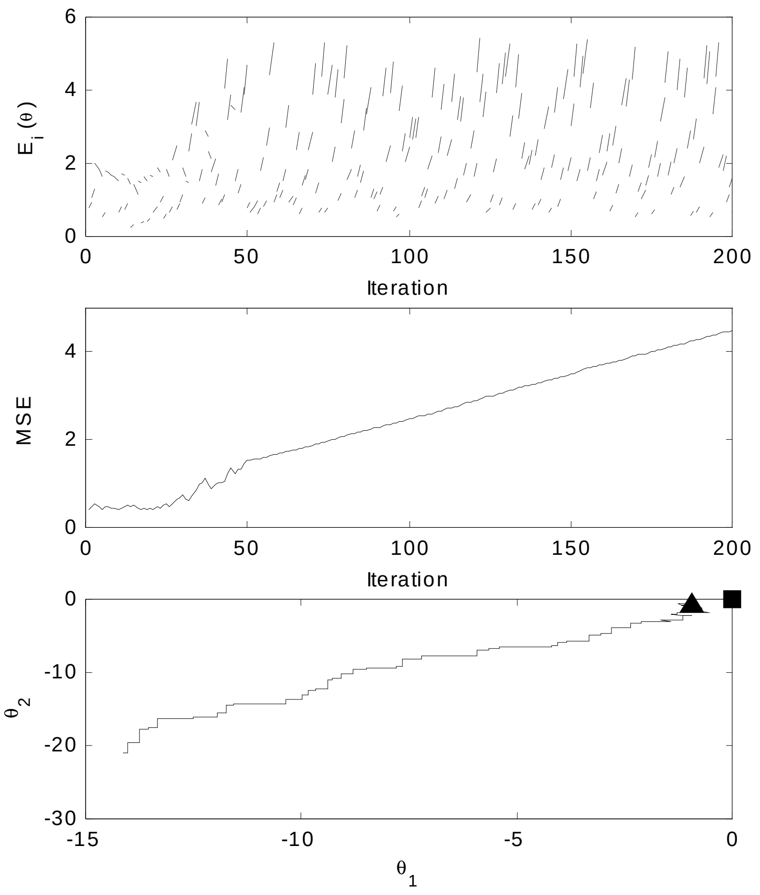

Figure 3.8. (a) Instantaneous and (b) total errors for a divergent training case and (c) evolution in parameter space

problem is somewhat strange, but if the hypothesis of gradient descent were correct, then we should have at least descent in terms of instantaneous errors. We still would not be able to conclude much in terms of total error regarding a random data set, but we would have descent at least in instantaneous error; here this is not the case.

We can take the same approach as in the first example, that is, to move away from the non-differentiability points and observe what happens to the algorithm. Thus, we introduce a detuning parameter σ that is added to the component of $\boldsymbol{x}$ that was chosen equal to the corresponding component of θ. As in the previous case, this detuning allows for differentiability and thus generally gives better results showing what were the effects of hitting non-differentiable points. Note that the same considerations regarding a different training data set hold in this case as well.

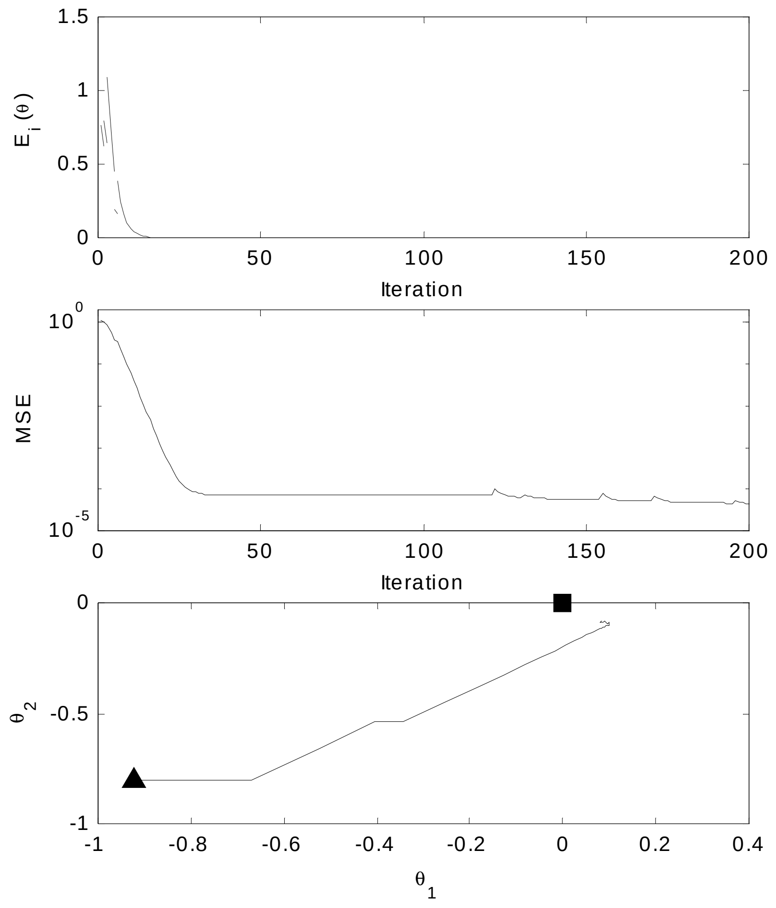

Figure 3.9. (a) Instantaneous and (b) total errors for a convergent training case (detuning σ = 0.1) and (c) evolution in parameter space

Choosing σ = 0.1 and exactly the same initialization of the previous case, the tuning algorithm manages to converge to a solution very close to the optimal solution as illustrated in Fig. 3.9. The exact same results are obtained for any smaller value of σ (σ = 0.001, σ = 0.00001). This seems to contradict the hypothesis formulated in the previous example of a continuum between differentiability and non-differentiability. Indeed, in this particular example we can only observe sudden changes reaching the non-differentiable points, but no gradual changes are noted. This might be due to the different (bipolar) nature of the derivatives in this case.

In Fig. 3.10 we show the instantaneous error versus the step size along the negative "gradient" direction at the starting point of the algorithm. The solid line indicates the non-improving behavior observed for σ = 0, that is in the non-differentiable case. The other two lines show the behavior of the instantaneous error for σ = 0.1 and σ = 0.00001. There is not too much difference in directional behavior for these two last cases. This seems to confirm the idea of a sudden change in the algorithm for σ = 0 as a consequence of *hitting* a non-differentiable point. Indeed, for σ = 0 the update direction is no longer an improving direction and may become (as it happens in this case) a non-improving direction.

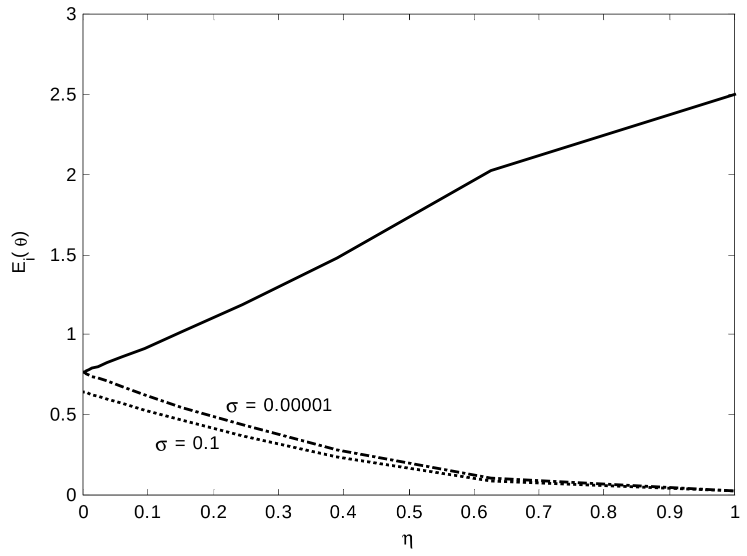

Figure 3.10. Directional behavior for the non-differentiable case (solid line) and two detuned cases (σ = 0.00001 dash-dot line, σ = 0.1 dotted line)

The same situation is depicted in a three dimensional plot of the instantaneous error function shown in Fig. 3.11. The instantaneous error function is a well-behaved function that would not seem too hard to optimize. The solid lines all along the surface indicate some of the points of discontinuity for the derivatives. Point **A** is the random initial point for the algorithm, and **B** is the point reached taking a small step in the "gradient" direction in the non-differentiable case (σ = 0); but, using a small detuning σ = 0.00001, the algorithm updates point **A** to point **C**. We are following two opposite directions, and obviously the second one (**AC**) is an improving one. In the first case we move along a non-improving direction from **A** to **B**, thus, increasing the instantaneous approximation error.

This second example is beneficial in illustrating the risks posed by the non-differentiability of triangular membership functions. Indeed, a divergent case was illustrated, along with its convergent counterpart obtained through the use of a small detuning factor. Unlike the first example, in this case the influence of the magnitude of the detuning factor seems null. This shows a sudden variation in the behavior of the algorithm.

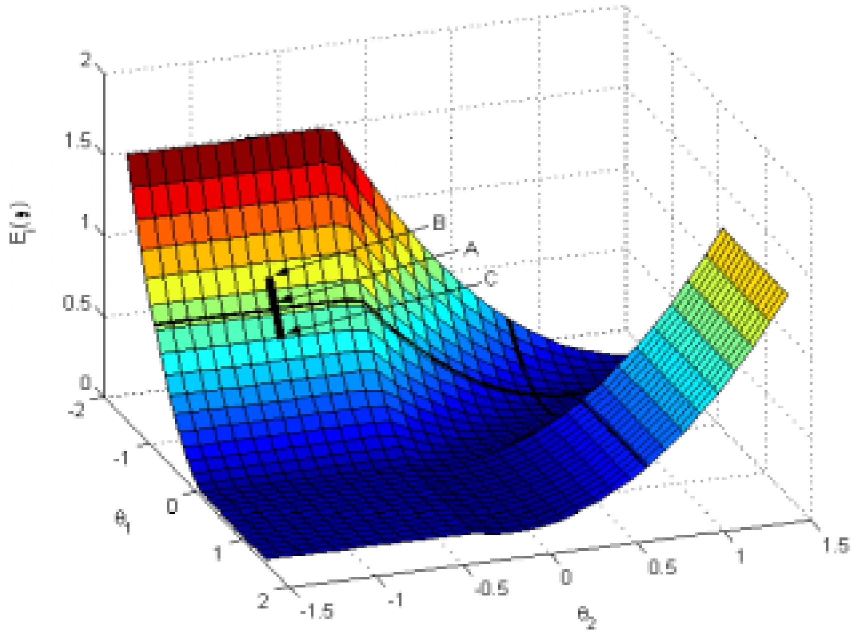

Figure 3.11. Instantaneous error function and one algorithmic step for the non-differentiable case (AB) and for the detuned case (AC, σ = 0.00001)

3.4 Conclusions

This chapter highlighted and discussed the threats offered by the non-differentiable nature of the optimization problem in supervised learning of certain FLSs. A simple SISO function approximation example was formulated and discussed in Section 3.2. It was shown that non-differentiability could practically affect the evolution of the algorithm by slowing it down. Indeed, only getting close to the points of non-differentiability deteriorates the performance (speed) of the algorithm by increasing the probability of overstepping. This seems to point to the fact that the non-differentiability problem is not only a mathematical abstraction after all, but in fact a pragmatic concern that in this case materializes due to the sudden variation in first derivative.

In order to attack this problem, we considered revising the pattern-by-pattern training algorithm. When the algorithm is close to points of non-differentiability the optimal step-size in the gradient direction is such that the corresponding parameter will be moved almost on top of the point of non-differentiability. Therefore, whenever an adjustable parameter is about to cross a training point (i.e., a point of non-differentiability), we can clip its update and stop it just a little over the corresponding training point. This heuristic approach does not affect the efficiency of the learning algorithm but might yield an advantage in terms of convergence rate. The implementation and testing of the approach revealed that sometimes there is a very marginal improvement in final error when comparing this approach to a pattern-by-pattern training. In this comparison a fixed set of training data was used. Moreover, other times there is no improvement offered by this heuristic. Therefore, this heuristic was discarded and is not discussed further in this work.

In the second example of Section 3.3 a simple two-input one-output function approximation example with two adjustable parameters was formulated and discussed. A divergent case was illustrated, along with its convergent counterpart obtained through the use of a small detuning factor. With the latter example we showed some of the problems involved with the non-differentiable nature of piecewise linear membership functions. Triangular membership functions are non-differentiable at a finite number of points, whenever the corresponding parameters assume any of these values, the gradient ceases to exist, and the resulting direction (obtained by an arbitrary choice of “derivative” values) is not necessarily an improving direction. Pragmatism would suggest that this case is not too important since the probability of hitting a point of non-differentiability is zero. The theoretical problem, however, still exists: the approach is not a gradient descent and is not theoretically convergent; and, thus, it seems very unsuitable especially for online applications. Moreover, there is also a practical concern as illustrated by the first example in which even only closeness to non-differentiable conditions slowed the convergence of the algorithm.

Chapter 4
Problem Reformulation

4.1 Introduction

The supervised learning problem formulated in Chapter 2 is a nonlinear programming problem that, as we saw, has been approached in several ways in the technical literature. One of the main limitations of the existing approaches is their local nature. No global optimization approach to the supervised learning of fuzzy logic systems has been successfully introduced (besides stochastic search methods). Some attempts at global learning algorithms [9,36,87] or data processing techniques [15] have been presented, but no hard proof of their global convergence has been presented. This is obviously a major problem hindering the analysis capacity of researchers in the FLS field.

This chapter takes a first step in the direction of global optimization of FLSs. The use of a newly developed *reformulation-linearization technique* (RLT) for nonconvex global optimization [71] is proposed. The idea underlying this technique is to reformulate the original problem in order to generate successive tight linear programming relaxations. The essence of this RLT approach is explained in the following Section 4.2. This is obviously not enough to give justice to the RLT approach, fully explaining its details and intricacies, and for the reader more interested in it we suggest [71].

The problem in its current form is not amenable to be approached with this RLT technique, and thus it is reformulated in a suitable new way that we called the *equation error* formulation (by direct analogy with the system identification literature). In essence in this formulation, instead of trying to minimize the sum-of-squares of errors between the output of the FLS and the desired output, we manipulate the equations in order to create an error that is polynomial in the antecedent membership degrees and consequent constants. The details of this new approach are illustrated in Section 4.3. The corresponding optimization problem is still affected, however, by the non-differentiability of membership functions. Thus, in Section 4.4 we propose a different formulation for piecewise linear membership functions that applies as well to minimum and maximum operators. In this formulation, adding suitable integer variables hides the discontinuity of the first derivative. Unfortunately, this formulation expands the number of variables as well as the degree of the problem; this, coupled with the expansion of dimensionality inherent to the RLT technique, causes the problem to become very big and thus intractable. This shortcoming, as well as the possibility of lower order polynomial approximations to the higher order problem (as well as to Gaussian or bell-shaped membership functions) is discussed with other possible alternatives in Section 4.5. The use of a quadratic fitting heuristic presented in the next Chapter 5 was preferred. Therefore, this chapter is closed by some concluding remarks in Section 4.6.

4.2 A Reformulation-Linearization Technique

This section loosely describes a newly developed reformulation-linearization technique proposed by Sherali and Adams [71]. This technique provides a mean of automatically generating tight linear programming representations to solve discrete and continuous nonconvex programming problems to global optimality. We will focus on its application to solving (continuous) polynomial programming problems having integral exponents. Its applicability to discrete programming problems is very important though, since the full design of a FLS (i.e., including the design of the structure of a FLS) has discrete components. Thus, the RLT approach could also be used for this larger design problem.

Consider a polynomial programming problem (i.e., a problem where the objective function and the constraints are polynomial in its variables) $PP(\Omega)$ with integral exponents and in m variables, each having lower and upper bounds. Moreover, let us call M the maximum degree of any polynomial term in this problem. The reformulation approach starts by adding to the problem new constraints obtained by lower and upper bounds on the problem variables. Considering the *j*-th problem variable ($j = 1,2,\ldots,m$) x_j with lower and upper bounds l_j and u_j respectively, let us call *bounding factors* the terms $(x_j - l_j) \geq 0$ and $(u_j - x_j) \geq 0$. New implied constraints are generated by taking all the distinct products of bounding factors (i.e., *bound-factor products*) *M* at a time. The number of distinct constraints generated this way is given by

$$\binom{2m+M-1}{M} \tag{4.1}$$

Once the problem is augmented with the bound-factor products we can linearize it by substituting each nonlinear product (e.g., $x_1x_2^2x_3$) in the augmented optimization problem with a new RLT variable (e.g., X_{1223}). The number of new RLT variables generated at this stage is

$$\binom{m+M}{M} - (m+1) \tag{4.2}$$

We now have the RLT linear programming relaxation $LP(\Omega)$ to the original problem. We can solve $LP(\Omega)$ with any (efficient) linear programming solver. Solution of the linear program offers a lower bound to the solution of the original problem (in the region where it is solved); any solution of $PP(\Omega)$ is also a solution of $LP(\Omega)$, while the opposite does not necessary hold. Moreover, if the nonlinear products are equal to the value of the new RLT variables (e.g., the values of X_{1223} and $x_1x_2^2x_3$ obtained by solution of the linear program are equal), then the solution to $LP(\Omega)$ is also feasible to $PP(\Omega)$ and thus solves it in Ω.

The problem at this point is to be able to convert the solution obtained by the linear program into a suitable solution to the original polynomial problem, as this is not generally possible unless the discrepancy between the new RLT variable and the nonlinear products is zero. Therefore, the solution of the linear programming problems is embedded in a branch-and-bound algorithm aiming at "facilitating" this conversion by trying to decrease these discrepancies to zero. Starting from the initial search region (a hypercube given by the lower and upper bounds), the linear programming relaxation is generated and solved. Unless an optimal feasible solution to the polynomial problem is found, this region is partitioned, choosing a suitable branching variable. This variable is the one that created the biggest difference between nonlinear products and new RLT variables. Any region corresponds to a node in the branch-and-bound tree and to a solution of a linear programming problem. Every linear programming problem will always give a lower bound to the optimal solution of the original problem, as well as sometimes an upper bound (i.e., incumbent solution) to the solution. If the lower bound of a node is higher than the current incumbent solution, then the node is fathomed (i.e., excluded). If in the process the minimum lower bound on the solution becomes equal to the incumbent solution, then the incumbent solution is the sought after optimal solution to the original polynomial problem. Moreover, it was proven [71] that if finite termination is not achieved, thanks to the augmentation with bound-factors products and to the branching rule, the problem will be solved asymptotically.

Figure 4.1 explains a branch-and-bound procedure through a pictorial example. Let us assume we want to minimize the objective function represented in Fig. 4.1 (solid line) in the interval Ω_1. The dash-dot line in Fig. 4.1 represents a relaxation of the objective function in Ω_1. The solution of the relaxed problem leads to a solution with value ν_1. Therefore, we can start building the branch-and-bound tree (Fig. 4.1, on the right) by putting a first node Ω_1 with lower bound (*LB*) ν_1 and incumbent solution (ν^*) that is infinity

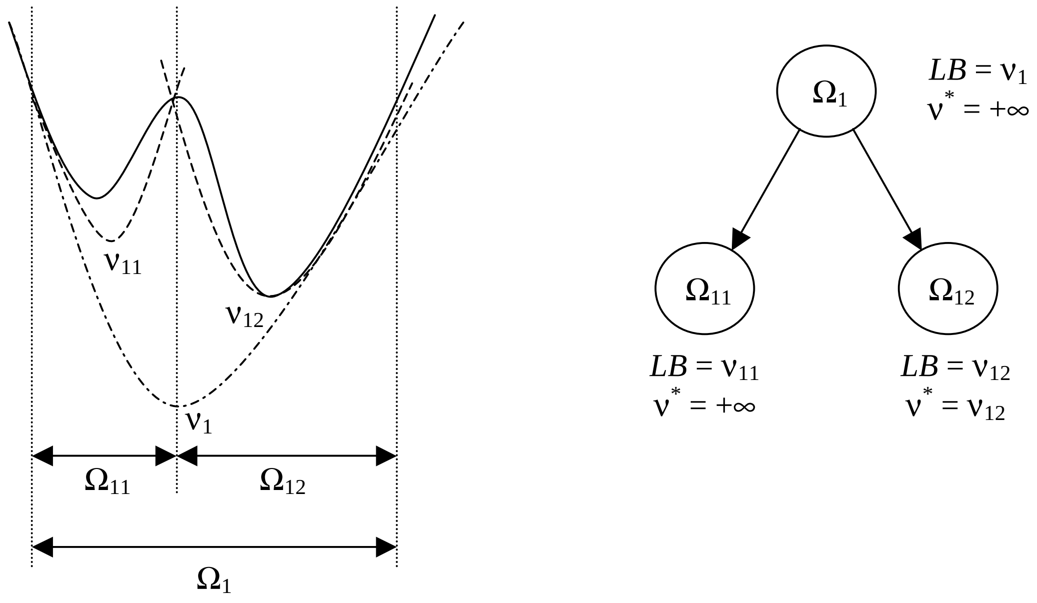

Figure 4.1. Example of branch-and-bound; objective function and relaxations (left), branch-and-bound tree (right)

(since ν_1 does not correspond to a solution of the original problem). The algorithm will now branch, dividing the search space Ω_1 into Ω_{11} and Ω_{12}. The two corresponding relaxations are shown in Fig. 4.1 with dashed lines. For the node Ω_{11} the solution of the relaxed problem leads to the solution value $\nu_{11} < \nu_1$, thus, the lower bound is updated ($LB = \nu_{11}$). The solution of the relaxed problem at the node Ω_{12} has value $\nu_{12} < \nu_{11}$ and it is also a solution of the original problem in Ω_{12}. Therefore, the lower bound is updated ($LB = \nu_{12}$) and also the incumbent solution is ($\nu^* = \nu_{12}$). Moreover, node Ω_{11} is fathomed since its solution is higher than the current incumbent solution. Finally, the current lower bound and incumbent solution are equal, and therefore the algorithm terminates, having found the optimal solution ν_{12} to the original problem.

We now need to formulate the supervised learning problem of FLSs. In the following Section 4.3 we introduce a novel formulation that is polynomial in the membership functions and consequent constants. In Section 4.4 we will present a polynomial reformulation of triangular membership functions, therefore, yielding a polynomial optimization problem to which the RLT can be applied.

4.3 The Equation Error Approach

Most, if not all, the supervised learning approaches employ a problem formulation like the one in Equation (2.1). This problem formulation is aimed at minimizing the mean square approximation error to the given training data. Obviously, this is not the only possible problem formulation. Different formulations can be introduced and might be helpful by offering an easier or more suitable problem structure. Borrowing the name from the system identification literature, we can call the error formulation in (2.1) an *output error* formulation. In the following we propose a different problem formulation and show some of its advantages in terms of problem structure. Namely, the problem becomes polynomial in the membership degrees as well as in the consequent parameters.

Given the i-th training datum ($\boldsymbol{x_i}$, y_{di}) the output of the FLS correspondent to a given set of parameters is given by (1.25). We desire this output to be as close as possible to y_{di}, thus, we would like:

$$y_{di} = \frac{\sum_{l=1}^{R} \delta_{h(l)}(\boldsymbol{w_c}) \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}, \boldsymbol{w_a}) \right]}{\sum_{l=1}^{R} \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}, \boldsymbol{w_a}) \right]} \tag{4.1}$$

Equation (4.1) can be rewritten as

$$\sum_{l=1}^{R} \left[\delta_{h(l)}(\boldsymbol{w_c}) - y_{di} \right] \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}, \boldsymbol{w_a}) \right] = 0 \tag{4.2}$$

with the realistic assumption that at least one of the rules is activated, that is

$$\sum_{l=1}^{R} \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}, \boldsymbol{w_a}) \right] \neq 0 \tag{4.3}$$

Obviously, in the absence of exact approximation of the data, the equality in (4.2) will not perfectly hold. Thus, on the right hand side of (4.2) there will be a nonzero error term e_i, that is

$$\sum_{l=1}^{R} \left[\delta_{h(l)}(\boldsymbol{w_c}) - y_{di} \right] \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}, \boldsymbol{w_a}) \right] = e_i \tag{4.4}$$

We can write one equation like (4.4) for each of the N training data. Thus, we can reformulate our optimization problem as one that minimizes the sum of these error terms squared:

$$\min_{\boldsymbol{w}} E(\boldsymbol{w}) = \frac{1}{2N} \sum_{i=1}^{N} e_i^2 \tag{4.5}$$

Writing this out explicitly, we obtain:

$$\min_{\boldsymbol{w}} E(\boldsymbol{w}) = \frac{1}{2N} \sum_{i=1}^{N} \left\{ \sum_{l=1}^{R} \left[\delta_{h(l)}(\boldsymbol{w_c}) - y_{di} \right] \left[\bigotimes_{j=1}^{n} \mu_{jk(j,l)}(x_{ij}, \boldsymbol{w_a}) \right] \right\}^2 \tag{4.6}$$

The problem formulation we are proposing through (4.6) could be called an *equation error* problem formulation as opposed to the *output error* of (2.1). The equation error formulation has the advantage of being polynomial in the membership degrees μ and in the consequents δ, regardless of the use of minimum or product *t-norm*. Obviously, this problem formulation can be approached with both pattern-by-pattern and batch training.

The equation error approach is equivalent to the output error approach whenever the right hand side of (4.3) is unity (or constant) for every point in the training data set; this simplifying assumption is sometime used in FLSs, but does not correspond to the most general case. In a general case the error in the equation error approach is a scaled version of the error in the output error approach since (4.4) can be rewritten as

$$\frac{\sum_{l=1}^{R}\delta_{h(l)}(\boldsymbol{w_c})\left[\bigotimes_{j=1}^{n}\mu_{jk(j,l)}(x_{ij},\boldsymbol{w_a})\right]}{\sum_{l=1}^{R}\left[\bigotimes_{j=1}^{n}\mu_{jk(j,l)}(x_{ij},\boldsymbol{w_a})\right]}-y_{di}=\frac{e_i}{\sum_{l=1}^{R}\left[\bigotimes_{j=1}^{n}\mu_{jk(j,l)}(x_{ij},\boldsymbol{w_a})\right]} \tag{4.7}$$

In Equation (4.7) the term on the left hand side is the output error, while the right hand side contains the equation error divided by the firing strengths summed over all the rules; the two errors are related through this scaling factor. In a well-structured FLS the scaling factor should always be around unity since there should always be at least one rule that fires with a higher strength (still less than unity though), while all the other rules should be dominated by this one, but still contribute to the sum. In general though, this factor can be bound between 0 and the number of rules *R*. Indeed,

$$0\le\mu_{jk(j,l)}(x_{ij},\boldsymbol{w}_a)\le 1\;\Rightarrow\;0\le\bigotimes_{j=1}^{n}\mu_{jk(j,l)}(x_{ij},\boldsymbol{w}_a)\le 1\;\Rightarrow\;0\le\sum_{l=1}^{R}\left[\bigotimes_{j=1}^{n}\mu_{jk(j,l)}(x_{ij},\boldsymbol{w}_a)\right]\le R$$

Depending on the minimum and maximum amount of overlap of the rules, tighter lower and upper bounds can be found. In general, the upper bound will be significantly lower than *R* since it is very unrealistic that all the rules fire with a unity activation strength. Likewise, realistically assuming a minimum amount of overlap the lower bound will be larger than zero. A very particular FLS configuration with no overlap of membership functions could lead to a trivial case of 0 degree of membership in any fuzzy set and thus the lower bound could actually be attained. Moreover, this could be one of the solutions that the optimization seeks, since whenever the second factor on the left hand side of (4.4) is zero e_i is exactly zero as well. Thus, this factor has to be "monitored" and/or eventually constrained to a nonzero value. This can be assured by imposing a minimum amount of overlap among fuzzy sets; i.e., we always need

$$\sum_{l=1}^{R}\left[\bigotimes_{j=1}^{n}\mu_{jk(j,l)}\left(x_{ij},\boldsymbol{w_a}\right)\right] > 0 \tag{4.8}$$

Equation (4.6) can be used to formulate a gradient descent approach in analogy to the output error approach. Moreover, even in this case LS, RLS and hybrid approaches can be used. The different objective function and the elimination of the denominator in the derivative calculation might help the convergence properties of the problem.

In the following subsections we will further explain the different problem formulations and discuss their properties, for different choices of *t-norm* (min or product) and membership functions (triangular, trapezoidal, Gaussian, bell shaped). Moreover, we will see that in the case of triangular membership functions we obtain a piecewise polynomial problem in the membership function parameters, while Gaussian and bell shaped membership functions yield a factorable optimization problem. Therefore, in the following Section 4.4 a polynomial formulation for triangular membership functions is introduced, leading to a polynomial optimization problem amenable to the RLT approach.

4.3.1 Optimization problem with min *t-norm*

Using the minimum *t-norm* in (4.4) we obtain:

$$\sum_{l=1}^{R}\left[\delta_{h(l)}\left(\boldsymbol{w}_c\right)-y_{di}\right]\left[\min_{j=\{1,2,\dots,n\}}\left\{\mu_{jk(j,l)}\left(x_{ij},\boldsymbol{w}_a\right)\right\}\right]=e_i \tag{4.9}$$

Denoting with $j_o(l)$ the index j for which the minimum is achieved for the l-th rule we can rewrite (4.9) as

$$\sum_{l=1}^{R}\left[\delta_{h(l)}\left(\boldsymbol{w}_c\right)-y_{di}\right]\mu_{j_o(l)k(j_o(l),l)}\left(x_{ij_o(l)},\boldsymbol{w}_a\right)=e_i \tag{4.10}$$

This problem is linear in the consequent parameters and linear in the antecedent membership functions. The number of adjustable parameters contributing to the total system output will always be a subset of the total number of adjustable parameters. Thus, each input will *activate* fewer parameters than the total, and only these will be available for adjustment based on this input. This greatly simplifies the problem structure, as we will shortly see, especially when piecewise linear membership functions are used. On the other hand, this also poses a risk for the procedure because some parameters might end up never being updated as a result of either a non-informative data set or a particular evolution of the optimization algorithm used. In the following we will specialize the problem formulation for the case of the most common membership functions: piecewise linear, Gaussian or bell shaped.

4.3.1.1 Piecewise linear membership functions

The particular structure of (4.10) can be very useful in case of triangular, trapezoidal or general piecewise linear membership functions. If the membership function term in (4.10) is linear in the adjustable parameter(s), the overall problem becomes (piecewise) quadratic in the (antecedent and consequent) adjustable parameters. More precisely, a piecewise linear membership function can always be described by $\mu(x) = \alpha^{(1)}x + \alpha^{(2)}$ where $\alpha^{(1)}$ and $\alpha^{(2)}$ are either constants or adjustable parameters. In what I would call a *proper* linear segment the two parameters (or at least one of them) are adjustable. There are exceptions though, when the membership value is constant (e.g., 0 or 1 or any constant in]0,1[) $\alpha^{(1)}$ is a constant ($\alpha^{(1)} = 0$) and $\alpha^{(2)}$ is also a constant 0 or 1 or any intermediate number. The most common cases are $\alpha^{(2)} = 0$ for triangular and trapezoidal membership functions, $\alpha^{(2)} = 1$ for trapezoidal membership functions and *saturated* triangular functions. The third case $0 < \alpha^{(2)} < 1$ is almost never observed, but is still considered for sake of generality. Moreover, we could have parameterizations in which either the slope or the intercept (center of the triangular function) are adjusted.

Of all the R rules in the rule-base only some of them will be *activated*, that is will fire with a nonzero strength. Let us partition the rules between those that fire and those that do not, as follows:

$$\begin{aligned} L_f &= \left\{ l \in \aleph \middle| l \leq R \wedge \bigotimes_{j=1}^{n} \mu_{jk(j,l)}\left(x_{ij}, \boldsymbol{w}_a\right) > 0 \right\} \\ L_{nf} &= \left\{ l \in \aleph \middle| l \leq R \wedge \bigotimes_{j=1}^{n} \mu_{jk(j,l)}\left(x_{ij}, \boldsymbol{w}_a\right) = 0 \right\} \end{aligned} \tag{4.11}$$

It is obvious that $L_f \cap L_{nf} = \varnothing$ and that $L_f \cup L_{nf} = \{1,2, \ldots, R\}$. Moreover, if (4.8) holds we also obtain the important result that $L_f \neq \varnothing$. In such a case, of all the firing rules, some will correspond to *proper* line segments, and some will correspond to non-zero constants ($0 < \alpha^{(2)} \leq 1$). Thus, we can partition L_f into

$$\begin{aligned} L_{fp} &= \left\{ l \in L_f \middle| \mu_{j_o(l)k(j_o(l),l)}\left(x_{ij_o(l)}, \boldsymbol{w}_a\right) = \alpha^{(1)}_{j_o(l)k(j_o(l),l)} x_{ij_o(l)} + \alpha^{(2)}_{j_o(l)k(j_o(l),l)} \wedge \alpha^{(1\vee 2)}_{j_o(l)k(j_o(l),l)} \text{ free parameters} \right\} \\ L_{fnp} &= \left\{ l \in L_f \middle| \mu_{j_o(l)k(j_o(l),l)}\left(x_{ij_o(l)}, \boldsymbol{w}_a\right) = c_l \wedge 0 < c_l \leq 1 \text{ constant} \right\} \end{aligned}$$

Using this notation, we can rewrite (4.10) as

$$\sum_{l \in L_{fp}} \left[\delta_{h(l)}\left(\boldsymbol{w}_c\right) - y_{di}\right]\left[\alpha^{(1)}_{j_o(l)k(j_o(l),l)} x_{ij_o(l)} + \alpha^{(2)}_{j_o(l)k(j_o(l),l)}\right] + \sum_{l \in L_{fnp}} c_l \left[\delta_{h(l)}\left(\boldsymbol{w}_c\right) - y_{di}\right] = e_i \tag{4.12}$$

This problem is piecewise quadratic in the antecedent and consequent parameters. The piecewise nature stems from the different linear segment that is selected according to the position of the training datum. In the rare case of only non-proper segments active (i.e., $L_{fp} = \varnothing$) then the problem becomes linear in the consequent parameters.

An intuitive approach to solving this problem as expressed by (4.12) is an iterative least square approach. In this approach we would fix a set of parameters (say the consequent parameters) and adopt a least square approach to determine the other set of parameters, since now the problem would be linear in those parameters. Then, the other set of parameters is fixed and another least square problem is solved. This process is repeated in an alternating fashion until (local) convergence is reached. The major advantages of such an approach are simplicity, robust algorithmic properties, and the possibility of recursive implementation.

4.3.1.2 Gaussian and bell shaped membership functions

The discussion for Gaussian and bell shaped membership functions is similar to the one above. This case is simplified by the fact that those functions (unlike piecewise linear functions) always have an infinite support. Thus, (4.8) always holds since there will always be a function contributing (even minimally) to the output. Using Gaussian membership functions (4.10) becomes

$$\sum_{l=1}^{R}\left[\delta_{h(l)}(\boldsymbol{w}_c)-y_{di}\right]\exp\left[-\left(\frac{x_{ij_o(l)}-m_{j_o(l)k(j_o(l),l)}}{\sigma_{j_o(l)k(j_o(l),l)}}\right)^2\right]=e_i \tag{4.13}$$

while with bell shaped membership functions we obtain

$$\sum_{l=1}^{R}\left[\delta_{h(l)}(\boldsymbol{w}_c)-y_{di}\right]\frac{1}{1+\left[\left(\frac{x_{ij_o(l)}-m_{j_o(l)k(j_o(l),l)}}{\sigma_{j_o(l)k(j_o(l),l)}}\right)^2\right]^{b_{j_o(l)k(j_o(l),l)}}}=e_i \tag{4.14}$$

Equations (4.13) and (4.14) show the easier mathematical formulation offered by Gaussian and bell-shaped, along with the simpler piecewise polynomial property of piecewise linear membership functions.

4.3.2 Optimization problem with product t-norm

Using the product *t-norm* in (4.4) we have:

$$\sum_{l=1}^{R}\left[\delta_{h(l)}(\boldsymbol{w}_c)-y_{di}\right]\prod_{j=1}^{n}\mu_{jk(j,l)}\left(x_{ij},\boldsymbol{w}_a\right)=e_i \tag{4.15}$$

This problem is linear in the consequent parameters and linear in each of the antecedent membership functions. The number of adjustable parameters contributing to the total system output will always be a subset of the total number of adjustable parameters. The scenario is the same as for the minimum *t-norm*, with the difference that the number of parameters activated by each data point will be higher than in the case of minimum *t-norm* since the output comprises a product of all the membership functions. The biggest advantage of using a product *t-norm* is that it is continuously differentiable (unlike its minimum counterpart), and thus it does not pose any problem for gradient descent based approaches. In the following we will specialize the problem formulation for the case of the most common membership functions: piecewise linear, Gaussian or bell shaped.

4.3.2.1 Piecewise linear membership functions

The particular structure of (4.15) can be really useful in case triangular, trapezoidal or in general piecewise linear membership functions are used. Assuming the same parameterization and discussion of the corresponding Section 4.3.1.1, the overall problem becomes polynomial (degree $n + 1$) in the (antecedent and consequent) adjustable parameters. Assuming that (4.8) holds, there will always be at least one rule firing, thus once again we have $L_f \neq \varnothing$. Concentrating only on the firing rules, let us consider rule l ($l \in L_f$): of all the terms in the product some of them will be proper line segments and some will be constant terms. We can describe this situation by partitioning the set of the first n integers corresponding to the inputs into two sets, one corresponding to inputs producing proper line segments (S_l) and the second corresponding to inputs producing non-zero constant terms (S_{ol}). More formally, we can introduce the sets

$$\begin{aligned} S_l &= \left\{ j \in \aleph \middle| j \leq n \wedge \mu_{jk(j,l)}\left(x_{ij}\right) = \alpha^{(1)}_{jk(j,l)} x_{ij} + \alpha^{(2)}_{jk(j,l)} \right\} \\ S_{ol} &= \left\{ j \in \aleph \middle| j \leq n \wedge \mu_{jk(j,l)}\left(x_{ij}\right) = c_{jk(j,l)} > 0 \right\} \end{aligned} \tag{4.16}$$

Let C_l denote the cardinality of S_l, obviously the cardinality of S_{ol} is the complement of C_l to n. Thus, we can rewrite (4.15) as

$$\sum_{l=1}^{R} \left[\delta_{h(l)}\left(\boldsymbol{w}_c\right) - y_{di}\right] \prod_{j \in S_l} \left[\alpha^{(1)}_{jk(j,l)} x_{ij} + \alpha^{(2)}_{jk(j,l)}\right] \prod_{j \in S_{ol}} c_{jk(j,l)} = e_i \tag{4.17}$$

The l-th term in the summation in (4.17) is a polynomial of degree ($C_l + 1$) in the antecedent and consequent parameters altogether. In the rare case of only non-proper segments active (i.e., $C_l = 0 \ \forall l \in \{1,2, \ldots, R\}$), then the problem becomes linear in the consequent parameters. Otherwise, the problem is piecewise polynomial of degree at most ($n + 1$) in the antecedent and consequent parameters. Moreover, the problem is linear in the consequent parameters with fixed antecedent parameters and is linear in each of the antecedent parameters while the other antecedent parameters as well as consequent parameters are held constant. Therefore, even in this case an alternating least square approach could be adopted.

4.3.2.2 Gaussian and bell shaped membership functions

The discussion for Gaussian and bell shaped membership functions is similar to the one above because those functions (unlike piecewise linear functions) always have an infinite support. Thus (4.8) always holds and there will always be a function contributing (even minimally) to the output. With Gaussian membership functions (4.15) becomes

$$\sum_{l=1}^{R}\left[\delta_{h(l)}(\boldsymbol{w}_c)-y_{di}\right]\prod_{j=1}^{n}\exp\left[-\left(\frac{x_{ij}-m_{jk(j,l)}}{\sigma_{jk(j,l)}}\right)^2\right]=e_i \tag{4.18}$$

while bell shaped membership functions yield

$$\sum_{l=1}^{R}\left[\delta_{h(l)}(\boldsymbol{w}_c)-y_{di}\right]\prod_{j=1}^{n}\frac{1}{1+\left[\left(\frac{x_{ij}-m_{jk(j,l)}}{\sigma_{jk(j,l)}}\right)^2\right]^{b_{jk(j,l)}}}=e_i \tag{4.19}$$

The problem with Gaussian or bell-shaped membership functions is factorable, that is, its objective function can be expressed as the sum of products of univariate functions.

4.4 Polynomial Formulation of Triangular Membership Functions

We have seen in the previous section that the supervised learning problem becomes piecewise polynomial when using piecewise linear membership functions. Therefore, we are still facing the non-differentiability of the triangular membership functions, which in this case appears through the piecewise nature of the problem. In this section we show how any triangular membership function can be formulated as a constrained polynomial through the addition of suitable new integer variables and constraints. Moreover, this approach can be extended to piecewise linear membership functions in general, as well as minimum and maximum operators. A generic triangular membership function μ can be defined as

$$\mu(x;c,m_1,m_2)=\begin{cases} m_1(x-c)+1 & c-\dfrac{1}{m_1}\le x\le c \\ 0 & x<c-\dfrac{1}{m_1}\vee x>c+\dfrac{1}{m_2} \\ -m_2(x-c)+1 & c\le x\le c+\dfrac{1}{m_2} \end{cases} \tag{4.20}$$

where c is its center, and m_1 and $-m_2$ are its left and right slopes. The center is constrained by a lower and upper bound imposed by the range of the corresponding input in the given problem. Moreover, the m coefficients have to be positive and not excessively large. Thus, we can constrain the parameters as follows:

$$l\le c\le u \quad 0<m_1<M \quad 0<m_2<M \tag{4.21}$$

where M is a large positive number. Equation (4.20) can also be written as:

$$\begin{gathered}\mu(x;c,m_1,m_2)=\max(z,0)\\ z=\min[m_1(x-c),-m_2(x-c)]+1\end{gathered} \tag{4.22}$$

We will first try to express z as a polynomial function of its variables. Depending on the position of x with respect to c, we will select either the first or the second argument of the minimum defining z. Therefore, we can introduce some new "switching" variables, that opportunely constrained will perform this selection. Let us introduce two new integer (0,1) variables I_1 and I_2; their unit values indicate the selection of the corresponding argument of the minimum. With the addition of these new variables we can describe z as

$$\begin{aligned}&z=(I_1m_1-I_2m_2)(x-c)+1\\ &\textit{subject to}:\\ &I_1+I_2=1\\ &I_1(I_1-1)=0\\ &I_2(I_2-1)=0\\ &I_1(x-c)\le 0\\ &I_2(x-c)\ge 0\end{aligned} \tag{4.23}$$

The two integer variables act as a switch between the first and second terms of z. The first constraint makes sure that only one of them has unit value while the other is zero. Their (0,1) nature is ensured by the following two constraints. Finally, the last two constraints decide on the switch position based on the value of x. For example, if $x > c$ then I_1 has to be zero while I_2 is 1, thus selecting the second term for z.

Analogously, we can introduce two new integer (0,1) variables I_3 and I_4 to describe the membership function μ. Therefore, from Equation (4.22) we have:

$$\begin{aligned}
&\mu = I_3 z \\
&\textit{subject to}: \\
&I_3 + I_4 = 1 \\
&I_3(I_3 - 1) = 0 \\
&I_4(I_4 - 1) = 0 \\
&I_3[x - (c - 1/m_1)][x - (c + 1/m_2)] \le 0 \\
&I_4[x - (c - 1/m_1)][x - (c + 1/m_2)] \ge 0
\end{aligned} \tag{4.24}$$

The interpretation of this formulation is in perfect analogy to the one of (4.23). We can finally formulate the triangular membership function as

$$\begin{aligned}
&\mu = I_3[(I_1 m_1 - I_2 m_2)(x - c) + 1] \\
&\textit{subject to}: \\
&I_1 + I_2 = 1 \\
&I_3 + I_4 = 1 \\
&I_1(I_1 - 1) = 0 \\
&I_2(I_2 - 1) = 0 \\
&I_3(I_3 - 1) = 0 \\
&I_4(I_4 - 1) = 0 \\
&I_1(x - c) \le 0 \\
&I_2(x - c) \ge 0 \\
&I_3[x - (c - 1/m_1)][x - (c + 1/m_2)] \le 0 \\
&I_4[x - (c - 1/m_1)][x - (c + 1/m_2)] \ge 0
\end{aligned} \tag{4.25}$$

The membership function is now described as a 4th order polynomial involving four additional (0,1 integer) variables subject to 10 constraints. The value of these additional variables, as well as the evaluation of the constraints, depends on *x*. Thus, in the context of supervised learning we need to introduce 4 new variables and 10 constraints for every training point (x_i) we consider. This makes sense, since the value of the additional integer variable is the one that acts as a switch between linear parts. Therefore, it is dependent (through the last constraints) from the value of *x*.

In the same fashion we can develop a general polynomial formulation for the min and max operators. Therefore, this approach is useful in generating higher dimensional polynomial representations of minimum, maximum, and piecewise linear membership functions. Thus, it can also be used to attack the non-differentiability problems caused by the minimum *t-norm*. In the next Section 4.5 we discuss this problem reformulation starting from the simple one-dimensional example of Section 3.2.

4.5 Discussion

The equation error approach of Section 4.3 reformulated the problem as polynomial in the membership degrees and consequents. The following Section 4.4 reformulated the triangular membership functions as higher dimensional polynomials. Therefore, the supervised learning problem becomes polynomial in the adjustable parameters as well, and the RLT technique can be applied. Let us see how this is possible in the one-dimensional example of Section 3.2.

In this problem the objective function (i.e., mean square error) is:

$$f(\boldsymbol{c},\boldsymbol{\delta}) = \frac{1}{2N}\sum_{i=1}^{N}\left[\sum_{l=1}^{R}\mu_l(x_i,\boldsymbol{c})\delta_l - y_i^d\right]^2 \tag{4.26}$$

The optimization problem becomes to minimize *f* given by Equation (4.26) subject to all the necessary constraints, that is the problem constraints and all the ones deriving from (4.25) (remembering that we need to include one of these last constraints per training point). Thus, the set of constraints is:

$$
\begin{aligned}
&\mu_j(x_i) = I_{3ij}\left[\left(I_{1ij}m_{1j} - I_{2ij}m_{2j}\right)\left(x - c_j\right) + 1\right] \quad j = 1,2,\ldots,R \\
&c_j < c_{j+1} \qquad\qquad j = 1:(R-1) \\
&\left.\begin{aligned}
&I_{1ij} + I_{2ij} = 1 \\
&I_{3ij} + I_{4ij} = 1 \\
&I_{1ij}\left(I_{1ij} - 1\right) = 0 \\
&I_{2ij}\left(I_{2ij} - 1\right) = 0 \\
&I_{3ij}\left(I_{3ij} - 1\right) = 0 \\
&I_{4ij}\left(I_{4ij} - 1\right) = 0 \\
&I_{1ij}\left(x_i - c_j\right) \le 0 \\
&I_{2ij}\left(x_i - c_j\right) \ge 0 \\
&I_{3ij}\left[x_i - \left(c_j - 1/m_{1j}\right)\right]\left[x_i - \left(c_j + 1/m_{2j}\right)\right] \le 0 \\
&I_{4ij}\left[x_i - \left(c_j - 1/m_{1j}\right)\right]\left[x_i - \left(c_j + 1/m_{2j}\right)\right] \ge 0
\end{aligned}\right\} \begin{aligned} &i = 1:N \\ &j = 1:R \end{aligned} \\
&0 < m_{1j} < M, \quad 0 < m_{2j} < M, \quad l_j \le c_j \le u_j \qquad j = 1:R
\end{aligned}
\tag{4.27}
$$

Equations (4.26) and (4.27) define a polynomial problem of the 10th order in ($3R$ + $4NR$) variables. The problem that this formulation raises is the excessive order of the polynomial; consider the specific case R = 5 and N = 21. The problem becomes polynomial of 10th degree in 420 variables! Moreover, the number of additional bound-factor products to add for the RLT approach is given by (4.1) and is

$$\binom{849}{10} = O\left(10^{22}\right)$$

Obviously the RLT approach cannot be applied to this problem formulation. The two main problems are the excessive number of variables introduced by the reformulation of the triangular membership function, as well as the high order of the polynomial involved. The latter problem could be addressed by fitting a lower order polynomial (third or fourth order) to the higher order objective function, similarly to the approach in [73]. Moreover, a first RLT solution of the lower order polynomial problem could then be used as the starting point for a classical gradient based approach. The high dimensionality of the problem remains and it is a byproduct of the triangular membership functions reformulation. Thus, a different (lower dimensional) reformulation (if possible) might help in this respect since the heart of the problem is the fact that the problem ends up being scaled with N (i.e., the number of training points), that is, a very undesirable feature.

A different and interesting approach could be to forget the membership functions and compute the optimal membership degrees at the training points. This problem would be independent of the type of membership functions used, and in a second stage the best type of membership function to approximate the given optimal membership degrees could be identified and fitted to those data. The issue with this approach is that the number of variables involved would be again scaled with the number of training points. In order to entertain such an approach, we would need to define new variables corresponding to the membership degrees at the training points, that is, introduce the variables

$$\mu_{jk(j,l)}^{(i)} = \mu_{jk(j,l)}\left(x_{ij}, \boldsymbol{w_a}\right) \tag{4.28}$$

We will need to define N of such variables for each input, thus yielding a total of

$$N\sum_{j=1}^{n} K_j$$

variables, where K_j is the number of fuzzy sets on the j-th input and n is the number of inputs. The objective function will thus become

$$E(\boldsymbol{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left\{\sum_{l=1}^{R}\left[\delta_{h(l)}(\boldsymbol{w_c}) - y_{di}\right]\prod_{j=1}^{n}\mu_{jk(j,l)}^{(i)}\right\}^2 \tag{4.29}$$

In this case the problem is polynomial in these new membership degrees, and has order $2(n + 1)$. Therefore, once again, things easily blow up for big problems. For small problems they might still be manageable in terms of order of the polynomial, but not in terms of number of variables. Let us look back at the example we considered before. Consider N = 21, n = 1, and K_1 = 5; the number of membership degrees variables is 105 while the number of consequent variables is 5. The order of the polynomial objective function is 4 and the number of RLT constraints is once again given by (4.1)

$$\binom{223}{4} = O\left(10^8\right)$$

This approach yields an advantage in terms of decreasing the number of variables and of RLT constraints, but the problem size is still intractable since the number of variables scales like the number of training points. For these reasons we decided to concentrate on a different approach, namely the limited memory quadratic fit described in the following Chapter 5.

A final note regards the use of the equation error approach with RLT in presence of Gaussian and bell-shaped membership functions; which is possible since the corresponding problem described in Equations (4.18) and (4.19) is factorable. Therefore, an approach like that of [73] could be devised. The Gaussian and bell-shaped membership functions could be approximated by a lower order polynomial and the RLT approach could be applied.

4.6 Conclusions

In this chapter we introduced the RLT technique that we proposed to use for global optimization of a FLS design. The supervised learning problem was reformulated through the equation error approach and the polynomial reformulation of the triangular membership functions. This led to a polynomial formulation of the supervised learning problem, suitable for the application of an RLT technique. Unfortunately, the high dimensionality of the problem reformulation, along with the expansion of dimensionality of the RLT technique, generated a problem that is prohibitive to solve. Since the RLT technique is viable in principle, perhaps some other formulations of the problem along with some RLT variations might help in applying it to solve the supervised learning problem to global optimality.

Chapter 5
Step Size Selection by Limited Memory Quadratic Fit

5.1 Introduction

In the extensive literature review presented in Chapter 2 we have seen that the most popular supervised learning approach is the pattern-by-pattern training, mostly because it offers few computational problems and is easy to implement. Moreover, people generally use this method with a small constant step-size, taking advantage of the elevated number of updates it performs. A more exact approach to the problem (not often used in practice) consists of using batch mode training, that is, freezing the adjustable parameters, accumulating the same type of corrections as for the pattern-by-pattern method, and applying them at each epoch (i.e., entire presentation of the data set). The direction generated by this method is the true gradient (i.e., whenever it exists) of the objective function (the mean square error); therefore, this is a more "reliable" direction for descent, than the ones generated by the pattern-by-pattern method. Nonetheless, batch training is not used too often because once an update direction (i.e., the negative gradient) is generated at each epoch, it is computationally expensive to know how far to move along this direction in order to take full advantage of it.

From an optimization perspective one would want to perform a line search in the update direction in order to find the minimum of the objective function along this direction. There are a few methods to perform this task; the interested reader can see [3]. One of the most popular line searches is the quadratic search [3]. In a quadratic search three suitable points along the gradient direction are used to fit a (univariate) parabola, whose interpolated vertex position will give an estimate of the optimal step-size along the negative gradient direction. Repeating this process a few times (while updating the three points) will lead to a good estimate of the optimal step-size in the update direction; this procedure is simple to implement, but unfortunately it carries a burden in terms of objective function evaluations. In the supervised learning of fuzzy logic systems in batch mode, an objective function evaluation corresponds to the calculation of the mean square error in approximating the training data. Thus, any objective function evaluation is scaled with N (i.e., the number of training data) that is generally a large number in practical applications. Furthermore, if the algorithm incurs a point of non-differentiability (or very close to it), the optimal step-size might be zero and the algorithm would terminate at a point of non-zero gradient. A fixed small step-size or a step-size modification approach based on the increase or decrease of past errors could be used instead of a line search. The problem in this approach is that since the update of the parameters is performed only once per epoch, if the algorithm does not take full advantage of the objective function decrease in the update direction, then it risks a slow convergence.

In order to effectively use the batch gradient direction, we propose a method that judiciously increases the computational time and whose complexity does not scale with the number of training points, but with the number of parameters. This method is explained in detail in the following Section 5.2. In essence the method attempts to select the step-size in the batch negative gradient direction by using a quadratic fit performed on some past errors stored during the convergence history of the algorithm. For this reason we call this a limited memory quadratic fit; at every epoch we store the value of the parameters as well as the corresponding value of the mean square error. This “history” is then used to fit a reduced quadratic model for the mean square error as a function of the adjustable parameters. To limit the number of variables in the model and to ease the computational burden, we do not consider cross terms (e.g., such as x_1x_2), thus, limiting the fit to a paraboloid with contours being ellipses with axes parallel to the axes of the frame of reference.

The use of linear fits to objective function evaluations in the context of optimization has been widely used in the past in the context of response surface methodologies (RSM) [8]. Generally, a linear model is fitted to some objective function evaluations obtained according to an experimental design, and is then used to estimate the gradient direction.

Closing in to optimality, a second order model can be used in order to estimate the position of the optimum [29]. This approach is particularly useful and widely used for problems in which a closed form expression for the objective function and its derivatives are not available, thereby requiring an estimate of the gradient. For example, problems like discrete-event simulation response optimization [25,10] or shape design problems in computational fluid dynamics [48] adopt these techniques. One of the differences in our case is the fact that an analytical expression for both the objective function and its gradient are readily available. Moreover, in RSM, in order to obtain a reliable estimate of the model parameters, the model is fitted to data that are obtained through a judicious sampling of the objective function, generally by some experimental design. This process is generally costly in terms of objective function evaluations. Here we propose to fit our model to past objective function evaluations (i.e., mean square errors) that are readily available. Thus, the proposed methodology does not bear a cost in terms of ulterior mean square error computation; its only cost is the computation of the model parameters (i.e., a pseudoinverse), besides an obvious storage requirement.

We fit a quadratic model to the data, trying to estimate the position of the optimum as the vertex of the paraboloid; this is easily estimated from the model parameters and gives an indication of a possible position for the optimum to the problem. Thus, we select the step-size along the negative gradient direction corresponding to the point along the negative gradient bearing the minimum distance from this vertex. This step-size selection choice is in close analogy to what is employed in non-differentiable optimization problems [3,2,21,62,63] where it is not always required that the algorithm achieve an improvement in objective function value, but to decrease the distance from the optimum. The optimum obtained by the vertex of the paraboloid is not directly used to update the parameters since it is not considered very reliable information due to the fact that it is obtained by fitting the model to data that are not obtained from a judicious experimental design but from the convergence history of the algorithm. Nonetheless, the convergence history data contain some information on the objective function, which we exploit by selecting the step-size along the update direction. The negative gradient of the mean square error is more reliable and is thus used as an update direction.

The use of the quadratic fit is helpful in that it contains some global information about the objective function to be optimized. Moreover, it prevents the algorithm from being trapped at a point of non-differentiability. This approach was tested on some function approximation problems, always yielding superior and very consistent results in terms of final error and oftentimes convergence speed as well. A two-dimensional supervised learning problem formulation in a new and interesting matrix form (very efficient for MATLAB implementation) is presented in Section 5.3. Some computational experiments along with a related discussion are presented in Section 5.4.

The proposed approach is not limited to using the negative gradient direction; it is, indeed, an approach for step-size selection along any arbitrary update direction. Therefore, inexpensive gradient deflection approaches such as the generalized delta rule [20], very popular in the field of neural networks but not very used in the field of FLSs, could be employed as well in conjunction with the limited memory quadratic fit. The generalized delta rule is a gradient deflection approach without restarts, where the past update direction is multiplied by a small constant called the *momentum coefficient*. One of the issues in using such a rule is selecting the proper momentum coefficient; this could be achieved by using an established conjugate gradient approach such as Fletcher and Reeves' [3] or a strategy like the average direction strategy (ADS) proposed by Sherali and Ulular [68]. This technique has been successfully employed to speed up convergence in the context of differentiable and non-differentiable problems as well as RSM [29]. The common use of different learning rates (i.e., gradient pre-multiplication by a suitable diagonal matrix) suggests the use of variable metric methods (e.g., quasi-Newton methods), where some memoryless variations can be employed in order to alleviate storage requirements. In Section 5.5 we briefly introduce the generalized delta rule, as well as gradient deflection strategies and memoryless space dilation and reduction algorithm. The proposed limited memory quadratic fit is used in conjunction with these approaches and some sample results are presented. Finally, Section 5.6 offers some concluding remarks.

5.2 Step Size Selection by Limited Memory Quadratic Fit

Given some past evaluations of the mean square error $E(\boldsymbol{w})$, stored along with the value of the corresponding P adjustable parameters $\boldsymbol{w}$, we want to fit these data to a quadratic model of the following type

$$E(\boldsymbol{w}) \cong \beta_0 + \sum_{i=1}^{P} \beta_i^{(1)} w_i + \sum_{i=1}^{P} \beta_i^{(2)} w_i^2 \tag{5.1}$$

The model (5.1) requires the estimation of $O(P)$ parameters, more precisely, $(2P + 1)$ parameters; considering also the cross terms $w_i w_j$ would increase the number of the parameters by $P(P-1)/2$, i.e., $O(P^2)$. Moreover, with this parameterization the vertex $\boldsymbol{w}^*$ of the paraboloid is simply obtained as

$$\boldsymbol{w}^* = -\frac{1}{2}\left[\frac{\beta_1^{(1)}}{\beta_1^{(2)}} \quad \frac{\beta_2^{(1)}}{\beta_2^{(2)}} \quad \cdots \quad \frac{\beta_P^{(1)}}{\beta_P^{(2)}} \right]^T \tag{5.2}$$

This shows yet another advantage of excluding cross-terms from our quadratic model, since otherwise the coordinates of $\boldsymbol{w}^*$ would have to be obtained through the solution of P linear equations.

Given a point $\boldsymbol{w}(j)$ in parameter space corresponding to the position of the algorithm at the j-th iteration, and given an update direction $\boldsymbol{d}_j$ (e.g., the negative gradient direction), we are going to move along this direction using a step-size η_j to the next point $\boldsymbol{w}(j + 1)$ given by

$$\boldsymbol{w}(j+1) = \boldsymbol{w}(j) + \eta_j \boldsymbol{d}_j \tag{5.3}$$

We can find a suitable non-negative step-size η_j along this direction by stopping at the point along $\boldsymbol{d}_j$ yielding minimum distance from the vertex $\boldsymbol{w}^*$ of the paraboloid (in concept similar to what is used in subgradient optimization approaches [21,62,63]). Thus, η_j has to satisfy

$$\boldsymbol{d}_j^T \left[\boldsymbol{w}^* - \boldsymbol{w}(j+1)\right] = 0 \tag{5.4}$$

Substituting (5.3) in (5.4) and solving for η_j we obtain

$$\eta_j = \frac{\boldsymbol{d}_j^T \left[\boldsymbol{w}^* - \boldsymbol{w}(j)\right]}{\boldsymbol{d}_j^T \boldsymbol{d}_j} \tag{5.5}$$

Instead of trying to fit an approximated quadratic model like (5.1) that also bears the approximation of the data that are being used to build it, we can simplify computations by choosing a reduced quadratic model

$$E(\boldsymbol{w}) \cong \beta_0 + \sum_{i=1}^{P} \beta_i^{(1)} w_i + \beta^{(2)} \sum_{i=1}^{P} w_i^2 \tag{5.6}$$

Model (5.6) represents a paraboloid with circular contours instead of ellipses. We will call the method based on model (5.1) a limited memory quadratic fit (LMQ), while the one based on (5.6) a limited memory reduced quadratic fit (LMRQ). In the LMRQ fit, the vertex of the paraboloid is obtained according to

$$\boldsymbol{w}^* = -\frac{1}{2}\begin{bmatrix} \frac{\beta_1^{(1)}}{\beta^{(2)}} & \frac{\beta_2^{(1)}}{\beta^{(2)}} & \cdots & \frac{\beta_P^{(1)}}{\beta^{(2)}} \end{bmatrix}^T \tag{5.7}$$

The step-size is then selected according to Equation (5.5).

In the implementation of this method we will not store the entire evolution of the algorithm, but only a certain number of past points, hence its limited memory. We need to define the number of past data points (Q) that we want to store for performing these quadratic fits; obviously, we should have $Q > P$. The choice of Q influences the accuracy of the estimates as well as the computational and storage overhead, thus it needs to be defined according to this trade-off. Moreover, the algorithm needs some initial data to start performing the quadratic fit. Thus, it will either go through a warm-up period (in which either an optimal or a constant step-size or variations thereof are employed) or it will start by randomly sampling the search space. Let us call $\boldsymbol{B}$ the buffer $(P+1)\times Q$ where the past data are stored

$$\boldsymbol{B} = \begin{bmatrix} E(\boldsymbol{w}_1) & E(\boldsymbol{w}_2) & \cdots & E(\boldsymbol{w}_Q) \\ \boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots & \boldsymbol{w}_Q \end{bmatrix} \tag{5.8}$$

Once the LMQ or LMRQ fit is started and estimates of the β parameters are obtained, a check is run to verify that the quadratic approximation is indeed suitable, since especially at the beginning of the search, the objective function might only look linear. Thus, given a small tolerance ε, we verify that

$$\min_i \left|\beta_i^{(2)}\right| \geq \varepsilon \tag{5.9}$$

If the test is not successful, we move only a minimum step-size η_{min} along the update direction, otherwise we go on determining $\boldsymbol{w}^*$ as in (5.2) or (5.7) and compute the step-size according to (5.5).

The step-size obtained applying (5.5) is not necessarily non-negative; it will indeed be such if and only if the update direction $\boldsymbol{d}_j$ and the direction from the current point to the vertex of the parabola (i.e., $\boldsymbol{w}^*$-$\boldsymbol{w}(j)$) form an acute angle. This is not always necessarily the case, therefore, we impose a minimum limit η_{min} on the allowable step size. Analogously, to avoid excessive step-sizes we impose a maximum limit η_{max} on the allowable step-size.

We can now state the LMQ (and LMRQ) algorithms as follows:

1. Define Q, η_{min}, η_{max} and ε;
2. Initialize the buffer $\boldsymbol{B}$ with Q measurements obtained either by random sampling of the MSE or by the execution of any other algorithm;
3. Set the epoch number $j = 1$ and initialize $\boldsymbol{w}(1)$;
4. Compute $E[\boldsymbol{w}(j)]$ and $\boldsymbol{d}_j$;
5. Using $\boldsymbol{B}$ compute the parameters β for (5.1) (or (5.6));
6. If (5.9) does not hold set $\eta_j = -\infty$ and go to step 8;
7. Compute $\boldsymbol{w}^*$ by (5.2) (or (5.7)) and compute η_j using (5.5);
8. Set $\eta_j = \max(\min(\eta_j, \eta_{max}), \eta_{min})$;
9. Update $\boldsymbol{w}$ according to (5.3);
10. Set $k = \mathrm{mod}(j\text{-}1,Q) + 1$, replace the k-th column of $\boldsymbol{B}$ with $[E[\boldsymbol{w}(j)], \boldsymbol{w}(j)^{\mathrm{T}}]^{\mathrm{T}}$;
11. If a termination criterion is not met set $j = j+1$ and go back to step 4.

The termination criterion could be a maximum number of iterations, relative change in objective function or relative parameter change smaller than a given threshold, or any combination of the above. The operator mod(*a*,*b*) in step 10 indicates the remainder of the division of *a* by *b*, and is used in order to implement a sliding window on the past data.

In the above algorithm the estimated step-size could oscillate from minimum to maximum values very quickly, therefore we employ averaging (i.e., discrete low-pass filtering) using a sliding window that effectively produces a step-size that is the average of the newly calculated step-size with some past values. Therefore, given a width W of this sliding window, the effective step-size used in step 9 instead of η_j would be

$$\overline{\eta}_j = \frac{1}{W}\sum_{i=j-W+1}^{j}\eta_j \tag{5.10}$$

During the execution of the algorithm, the objective function value could tend to oscillate, especially depending on its nature and the values of the minimum and maximum step-sizes. Generally the algorithm can recover from these oscillations, but if a maximum number of iterations is used, it may terminate at a high value of the mean square error, before having the time to actually decrease again. Therefore, at every epoch we can store the best solution achieved and consider this the solution of the algorithm at that epoch.

The next Section 5.3 illustrates a novel matrix formulation for the supervised learning of a two-input and one-output TS-FLS that will be used in two test cases for the computational experiments of Section 5.4, and in the experiments of Section 5.5.

5.3 Matrix Formulation for a Two-Dimensional Problem

Consider a two-input single-output FLS with a different constant consequent per rule, that is, a TS-FLS with constant local models. The corresponding rule base is shown in Table 5.1 with the usual meaning of the symbols; K_1 and K_2 are the numbers of partitions on the first and second input, respectively. Using (1.25) the output of the FLS is given by

$$y(x_1, x_2, \boldsymbol{w}) = \frac{\sum_{i=1}^{K_1}\sum_{j=1}^{K_2}\mu_{1i}(x_1,\boldsymbol{w})\mu_{2j}(x_2,\boldsymbol{w})\delta_{i,j}}{\sum_{i=1}^{K_1}\sum_{j=1}^{K_2}\mu_{1i}(x_1,\boldsymbol{w})\mu_{2j}(x_2,\boldsymbol{w})} \tag{5.11}$$

Table 5.1. Rule base of a two-input FLS

$x_1 \downarrow x_2 \rightarrow$	μ_{21}	μ_{22}	$\cdots$	μ_{2K_2}
μ_{11}	$\delta_{1,1}$	$\delta_{1,2}$		δ_{1,K_2}
μ_{12}	$\delta_{2,1}$	$\delta_{2,2}$		δ_{2,K_2}
$\cdots$				
μ_{1K_1}	$\delta_{K_1,1}$	$\delta_{K_1,2}$		δ_{K_1,K_2}

From now on we will omit the dependence from ***x*** and ***w***, unless necessary, for sake of simplicity of the formulation. Let us introduce the vectors of membership degrees, collecting the membership degrees on the different fuzzy sets of a given input

$$\mu_{\boldsymbol{i}} = \left[\mu_{i1} \quad \mu_{i2} \quad \dots \quad \mu_{iK_i}\right]^T \quad i = 1,2 \tag{5.12}$$

The consequent constants are grouped in a matrix Δ of consequents

$$\Delta = \begin{bmatrix} \delta_{1,1} & \delta_{1,2} & \dots & \delta_{1,K_2} \\ \delta_{2,1} & \delta_{2,2} & \dots & \delta_{2,K_2} \\ \dots & \dots & \dots & \dots \\ \delta_{K_1,1} & \delta_{K_1,2} & \dots & \delta_{K_1,K_2} \end{bmatrix} \tag{5.13}$$

Let us also recall the norm 1 of a vector $\boldsymbol{v} \in \Re^n$ defined as

$$\|\boldsymbol{v}\|_1 = \sum_{i=1}^{n} |v_i| \tag{5.14}$$

Using the definitions above, we can rewrite the output of the FLS (5.11) as

$$y = \frac{\mu_1^T \Delta \mu_2}{\|\mu_1\|\mu_2\|_1\|_1} \tag{5.15}$$

Let us now consider the supervised learning problem where we desire to learn only one point. Equivalently, we can consider it to be a pattern-by-pattern mode of training. In essence, let us consider the update equations due to a single training point: if we are using the pattern-by-pattern training mode we will update the parameters accordingly, otherwise, in the batch training we will accrue the updates and apply their average only once per epoch. We define the usual squared error

$$E = \frac{1}{2}(y - y_d)^2 \tag{5.16}$$

The update of the consequent parameters can be grouped in a matrix of updates with elements

$$\left(\frac{\partial E}{\partial \Delta}\right)_{ij} = \frac{\partial E}{\partial \delta_{ij}} \quad \begin{array}{l} i = 1,2,\ldots,K_1 \\ j = 1,2,\ldots,K_2 \end{array} \tag{5.17}$$

By inspection of (5.15) and (5.16) we see that the consequent parameters updates are readily obtained as

$$\frac{\partial E}{\partial \Delta} = \frac{(y - y_d)}{\left\| \mu_1 \|\mu_2\|_1 \right\|_1} \mu_1 \mu_2^T \tag{5.18}$$

Introducing the scalar

$$k_e = \frac{(y - y_d)}{\left\| \mu_1 \|\mu_2\|_1 \right\|_1} \tag{5.19}$$

Equation (5.18) becomes

$$\frac{\partial E}{\partial \Delta} = k_e \mu_1 \mu_2^T \tag{5.20}$$

Let us now (reasonably) assume that each membership function μ_{ij} is parameterized by two parameters c_{ij} and b_{ij} representing its center and width, respectively, and that the parameters defining μ_{ij} do not affect any other membership function. The following approach can be easily extended to more than two parameters per membership function. In a fashion analogous to (5.12) we define the following vectors of parameters

$$\begin{array}{l} \boldsymbol{c_i} = \begin{bmatrix} c_{i1} & c_{i2} & \ldots & c_{iK_i} \end{bmatrix}^T \\ \boldsymbol{b_i} = \begin{bmatrix} b_{i1} & b_{i2} & \ldots & b_{iK_i} \end{bmatrix}^T \end{array} \quad i = 1,2 \tag{5.21}$$

The Jacobian matrices of the membership degree vector with respect to the parameters vectors are diagonal and are given by

$$\begin{aligned}\frac{\partial \mu_i}{\partial \boldsymbol{c}_i} &= \mathrm{diagonal}\begin{pmatrix}\frac{\partial \mu_{i1}}{\partial c_{i1}} & \frac{\partial \mu_{i2}}{\partial c_{i2}} & \cdots & \frac{\partial \mu_{iK_i}}{\partial c_{iK_i}}\end{pmatrix} \\ \frac{\partial \mu_i}{\partial \boldsymbol{b}_i} &= \mathrm{diagonal}\begin{pmatrix}\frac{\partial \mu_{i1}}{\partial b_{i1}} & \frac{\partial \mu_{i2}}{\partial b_{i2}} & \cdots & \frac{\partial \mu_{iK_i}}{\partial b_{iK_i}}\end{pmatrix}\end{aligned} \quad i = 1,2 \tag{5.22}$$

Let us also introduce the *m*×*n* unity matrix

$$\mathbf{1}_{m\times n} = \underbrace{\begin{bmatrix}1 & \dots & 1 \\ \dots & \dots & \dots \\ 1 & \dots & 1\end{bmatrix}}_{n}\left.\vphantom{\begin{bmatrix}1\\ \dots \\ 1\end{bmatrix}}\right\}m \tag{5.23}$$

By inspection of (5.15) and (5.16) we can express the antecedent parameters updates as

$$\begin{aligned}\frac{\partial E}{\partial \boldsymbol{c}_1} &= k_e \frac{\partial \mu_1}{\partial \boldsymbol{c}_1}\left(\Delta - y\mathbf{1}_{K_1\times K_2}\right)\mu_2 & \frac{\partial E}{\partial \boldsymbol{b}_1} &= k_e \frac{\partial \mu_1}{\partial \boldsymbol{b}_1}\left(\Delta - y\mathbf{1}_{K_1\times K_2}\right)\mu_2 \\ \frac{\partial E}{\partial \boldsymbol{c}_2} &= k_e \frac{\partial \mu_2}{\partial \boldsymbol{c}_2}\left(\Delta - y\mathbf{1}_{K_1\times K_2}\right)^T \mu_1 & \frac{\partial E}{\partial \boldsymbol{b}_2} &= k_e \frac{\partial \mu_2}{\partial \boldsymbol{b}_2}\left(\Delta - y\mathbf{1}_{K_1\times K_2}\right)^T \mu_1\end{aligned} \tag{5.24}$$

If the membership functions are triangular with center c_{ij} and width b_{ij}, they can be represented as

$$\mu_{ij}(x_i) = \max\left(1 - 2\frac{|x_i - c_{ij}|}{b_{ij}}, 0\right) \quad \begin{aligned}i &= 1,2 \\ j &= 1,2,\dots,K_i\end{aligned} \tag{5.25}$$

Introducing the signum function

$$sign(x) = \begin{cases}+1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0\end{cases} \tag{5.26}$$

The "derivative" of μ_{ij} with respect to the center c_{ij} and width b_{ij} is readily obtained as

$$\begin{aligned}\frac{\partial \mu_{ij}}{\partial c_{ij}} &= \frac{2}{b_{ij}} sign(x_i - c_{ij}) sign(\mu_{ij}) \\ \frac{\partial \mu_{ij}}{\partial b_{ij}} &= \frac{2}{b_{ij}^2}|x_i - c_{ij}| sign(\mu_{ij})\end{aligned} \quad \begin{aligned}i &= 1,2 \\ j &= 1,2,\dots,K_i\end{aligned} \tag{5.27}$$

Note that the term derivative in this context is loosely meant since we know that triangular membership functions are non-differentiable at some points. The reader can easily verify that at these points the "derivative" is set to zero.

The equations described in this section are all matrix equations, they can be implemented with computational advantage in specific software packages like MATLAB, for example, making this formulation both compact and efficient for this type software. Also taking advantage of multidimensional vector functions, vector multiplication, division, and power offered by MATLAB, the learning algorithm (either pattern-by-pattern or batch) can be implemented very efficiently.

5.4 Results and Discussion

In this section we present and discuss some results of the testing of the LMQ and LMRQ algorithms proposed in Section 5.2, along with the results obtained using a batch algorithm employing optimal step-size (BOS) computed through quadratic search. These results are compared to those obtained using pattern-by-pattern (PBP) training. We first consider the one-dimensional example introduced in Section 3.2, then we move to two bi-dimensional examples taken from the literature and attacked with the FLS discussed in Section 5.3.

The necessary algorithmic parameters were chosen according to what is customarily done in the literature, and trying to improve convergence by observing its behavior in a few experiments. The order of presentation of the data in the PBP approach is randomized as is customarily done to improve convergence [20]. Neither one of LMQ, LMRQ, or BOS requires (and thus uses) randomized presentation of the training data. In the LMQ and LMRQ algorithms the buffer is initially filled using Q randomly generated data points; a sliding window size of W = 3 samples is used for averaging of the step-size. All the algorithms are started from the same initial random point in parameter space, but even so the initial MSE for BOS, LMQ, and LMRQ is always the same, while that for PBP might be slightly different, due to the nature of the updates in the PBP (i.e., the parameters are not fixed during an epoch).

The algorithms are all stopped after a maximum number of epochs has elapsed. Moreover, the convergence histories depict the MSE as a function of execution time. This execution time should be taken as an approximate (rather than exact) measure of the speed of the algorithm since it depends on implementation details as well as other processes eventually interfering with the execution of the algorithms (even though an effort was made to not have any other foreground processes running). The learning curves are obtained by equally dividing the total execution time by the number of iterations. The computational cost for PBP, LMQ, and LMRQ is always the same at each epoch, whereas the cost of the BOS will be higher in the initial epochs (where it might be more difficult to find a three-point pattern and to converge to a suitable step-size) and lower in the final ones. Therefore, the convergence history for the BOS will show it a little faster in the first epochs than it actually is. Finally, since we are initially filling a buffer for the LMQ and LMRQ algorithms, their execution time per iteration is actually smaller since the buffer filling process is an overhead that is divided over all iterations. Even though affected by all these approximations, it is interesting to visually present the results as a function of execution time, rather than epoch, since the execution time can give us a feel for the speed and convergence characteristics of the algorithms.

5.4.1 Example 1

We use the SISO FLS described in Section 3.2 to approximate a parabola $y = x^2$, of which $N = 21$ points equally spaced in [-1,1] are given. The FLS is characterized by $R = 5$ partitions on the input and consequent membership functions; this corresponds to a total of 10 adjustable parameters. In the PBP algorithm we choose a constant step-size $\eta = 0.1$. In both the LMQ and LMRQ algorithms we select $\eta_{max} = 6$, $\eta_{min} = 0.1$, $Q = 50$, and $\varepsilon = 10^{-4}$. The algorithms are all stopped after 200 epochs.

The initial random values for the antecedent parameters $\boldsymbol{c}$ are obtained by independent samples from a uniform distribution in [-1,1], and the consequent parameters are obtained by independent samples from a uniform distribution in [0,1]. The $\boldsymbol{c}$ parameters are also initially sorted in ascending order as required for the application of (3.4). Moreover, every run leading to a different order of these parameters (i.e., inversion) was excluded from the experiment. No significant difference in the number of inversions per algorithm was noted, an inversion was noted whenever the parameters were initially too close to each other.

A first convergence history is presented in Fig. 5.1. The four algorithms all seem to reach a similar final MSE, even though both LMQ and LMRQ have a slightly smaller one. Both BOS and LMQ seem equally fast and faster than the other algorithms, with LMRQ being initially the slowest, but also the one that leads to the smallest MSE. The overhead associated with the BOS can be noted even in a small example such as this one (N = 21); that is, the BOS requires a longer total execution time (about 27 s) which is about 50% larger than that required for the other three algorithms (about 18 s). However, the BOS compensates for this longer execution time by faster convergence. It can also be noted that the LMRQ indeed offers a slight computational advantage with respect to the LMQ, and, amazingly it is also faster than PBP. This can be explained by both the approximation and the fact that the PBP requires randomization of the order of presentation of the data points at each epoch (an operation implemented as a cycle, opposed to the matrix inversion

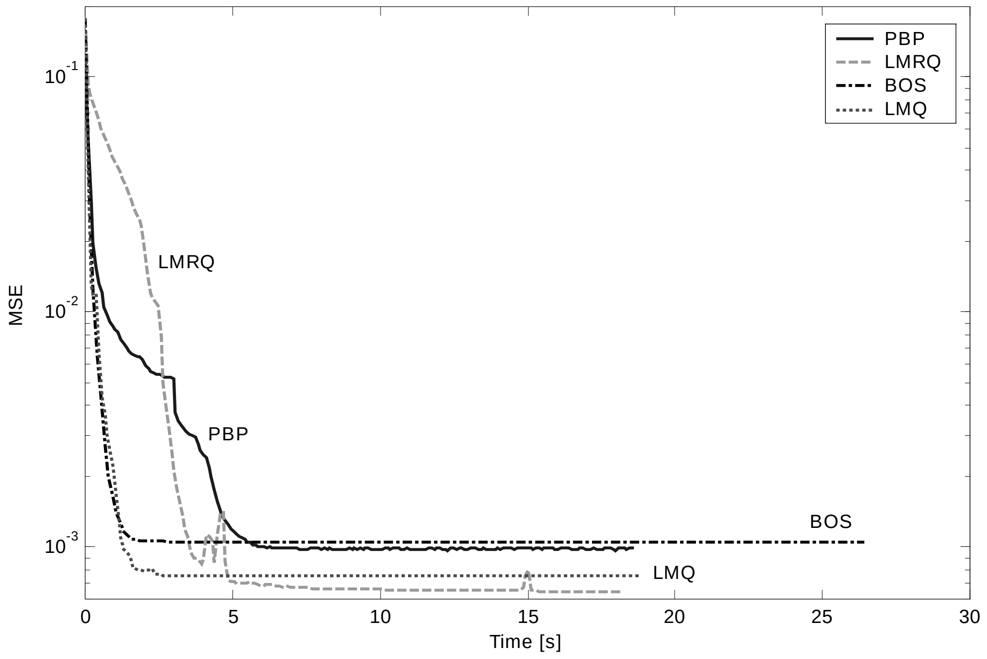

Figure 5.1. Convergence history for PBP, LMRQ, LMQ, and BOS; Example 1

necessary for LMQ and LMRQ, for which MATLAB is very efficient).

A final interesting note regarding the BOS is the fact that in the extensive experimentation that was performed, the BOS often stopped at (or very close to) a point of non-differentiability (showing once again a practical issue with non-differentiability) and did not move from there yielding a zero optimal step-size. Conversely, PBP, LMQ, and LMRQ did not suffer from this problem. Using PBP, there is only one point of non-differentiability per update (since a single training point at a time is used), this effectively reduces the chances of feeling an effect of non-differentiability. Both the LMQ and LMRQ algorithms do not feel an effect of non-differentiability since their step-size is chosen according to a limited memory quadratic fit and not according to a line search along the negative gradient direction. Obviously, these algorithms can still encounter a non-improving direction, but this would result in a non-improving step, that, due to the criterion chosen to determine the step-size, should at least result in a decreased distance from the sought optimum.

Another representative test is shown in Fig. 5.2. In this case the PBP algorithm gets trapped at a high value of the MSE (about 0.02) while all of LMQ, LMRQ, and BOS

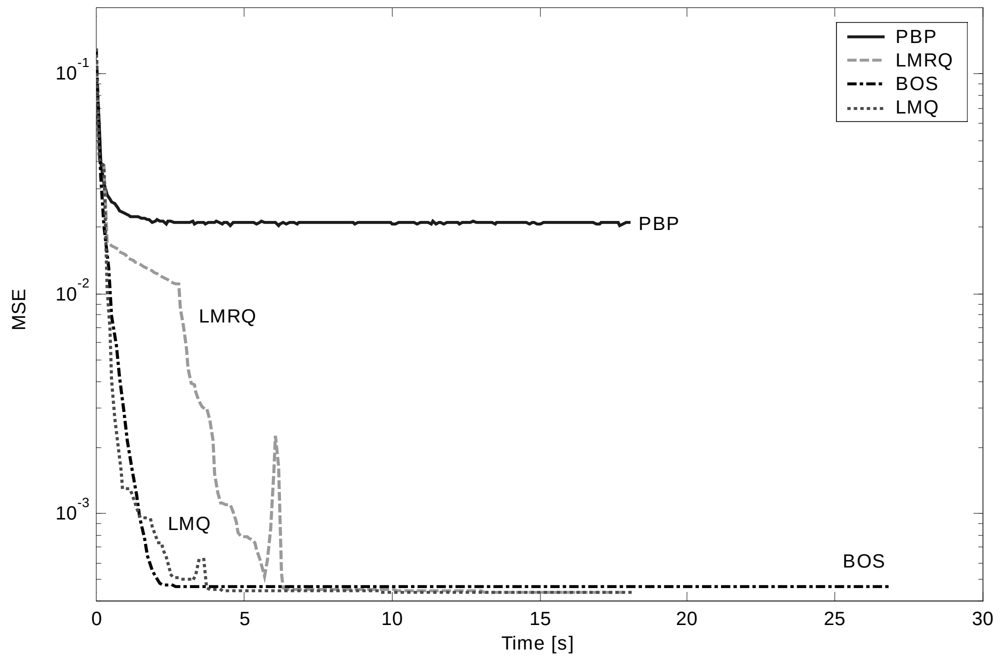

Figure 5.2. Convergence history for PBP, LMRQ, LMQ, and BOS; Example 1

manage to achieve a final MSE that is about one and one half orders of magnitude smaller. The abrupt stop of the BOS shows the presence of the same non-differentiability problem discussed for the previous case. Moreover, the speed characteristics of the three algorithms are similar to what was observed in the previous case. All three algorithms reach the same final MSE.

These two examples are representative of what was observed in the extensive experimentation on the algorithms. Most of the times all the algorithms manage to converge to satisfactory MSE values. The PBP algorithm and the BOS sometimes get trapped at sub-optimal points while LMQ and LMRQ consistently reach lower MSE values. In order to show these differences we performed a Monte-Carlo analysis executing five hundred runs of each algorithm using different random initializations were performed and collecting the final MSE. The histogram analog of a probability density function and a cumulative distribution of the errors were obtained for each algorithm. Figure 5.3 shows a bar plot of the number of times each algorithm produced a final MSE into certain error bins, while Fig. 5.4 shows the same information in the form of a cumulative distribution of the errors. Observation of Fig. 5.3 shows that most of the time the final MSE with any of the

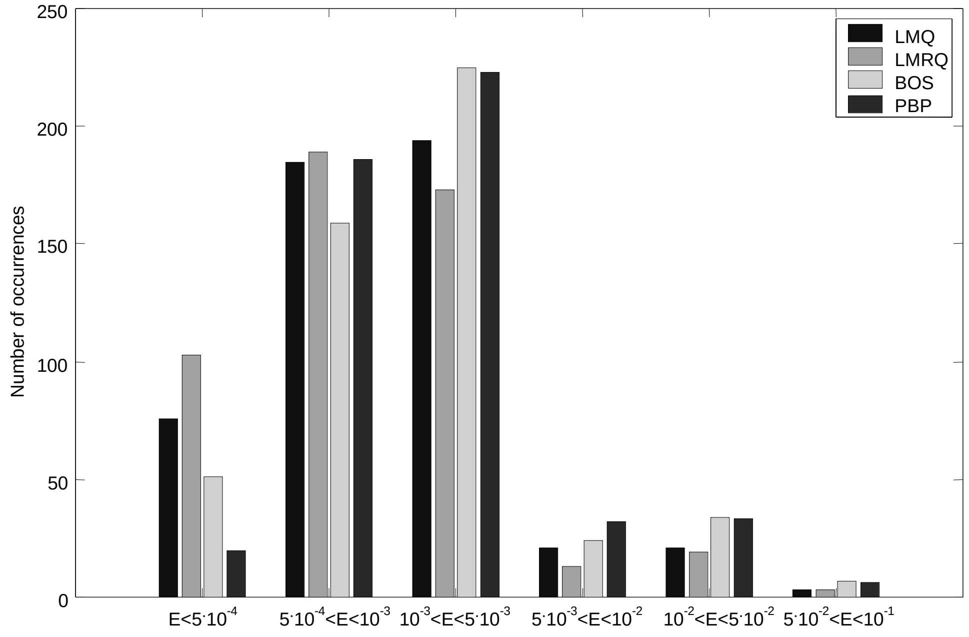

Figure 5.3. Density of final MSE for PBP, LMRQ, LMQ, and BOS; Example 1

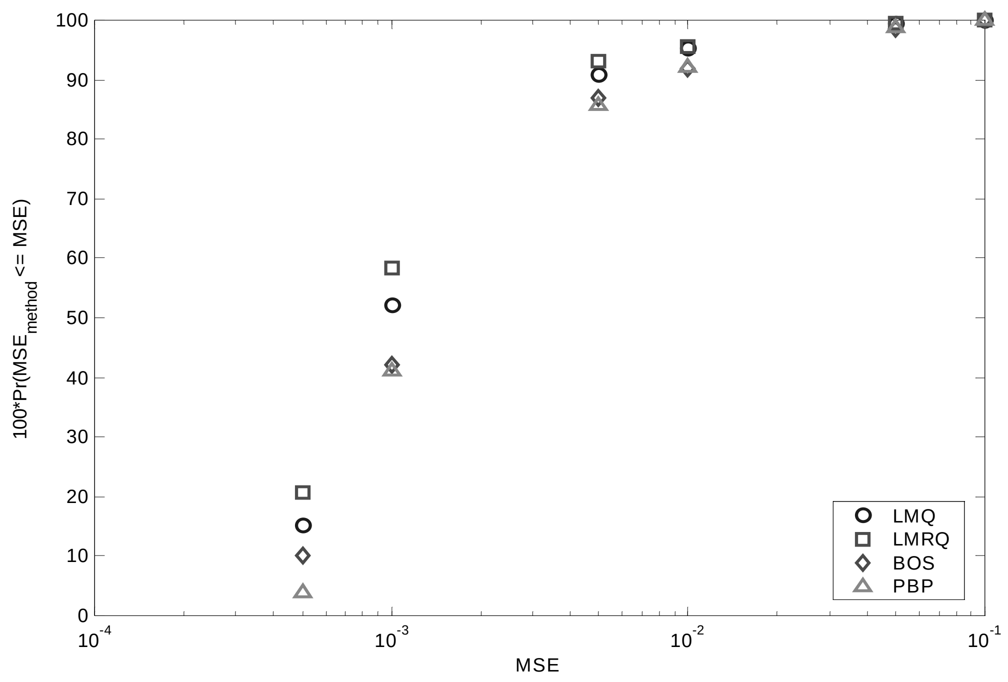

Figure 5.4. Cumulative distribution of final MSE for PBP, LMRQ, LMQ, and BOS; Example 1

algorithms is between $5{\cdot}10^{-4}$ and $5{\cdot}10^{-3}$. The LMRQ algorithm shows a final MSE smaller than $5{\cdot}10^{-4}$ in about 100 runs (20% of the time), immediately followed by the LMQ (76 runs), BOS (51 runs) and PBP (20 runs). The LMRQ and LMQ algorithms obviously outperform the PBP. Moreover, LMRQ also performs somewhat better than LMQ. This is strange since LMRQ is an approximation to LMQ, it could be motivated by the nature of the objective function in this case (its contours are perhaps circular). This could also be due to the fact that in LMRQ we are reducing the number of parameters describing the quadratic nonlinearities to only one, but by doing this we are increasing the quantity of data used to estimate this parameter, thus, yielding a more reliable estimate.

A final interesting observation is that BOS, not only offers smaller errors than PBP, but also shows a slightly higher number of larger MSEs (i.e., larger than 0.01). In other words BOS yields a flatter density of errors. This can be also observed more clearly in Fig. 5.4. Both LMQ and LMRQ yield a higher cumulative distribution of final MSEs (for low MSE values), and the gap with BOS and PBP decreases with increasing error levels.

Conversely, even though BOS starts with a higher probability than PBP for very small errors, it immediately gets very close to PBP, and finally flattens for large errors.

On average, the LMRQ, LMQ, and BOS algorithms performed better than PBP. The average ratio of final MSE for PBP and LMQ, LMRQ, or BOS was computed yielding the values 3.0, 3.6 and 2.4, respectively. Thus, on average the LMQ and LMRQ algorithms converge to errors that are 3 times lower than the one obtained by application of PBP. The BOS algorithm performs similarly. Obviously, this is an average measure that explains less than what was shown with the distribution of final errors. Indeed, we saw that on average the algorithms yield similar final MSEs, but both LMQ and LMRQ produce lower errors for a significantly higher number of times than with PBP. Therefore, the advantage of using LMQ and LMRQ is not in average terms but in terms of consistency in achieving low errors.

5.4.2 Example 2

In this example we use the two-input single-output FLS described in Section 5.3 to approximate the function

$$f(x_1, x_2) = \frac{1}{3}\sin(\pi x_1) + \frac{1}{6}\cos(\pi x_2) + 0.5 \tag{5.28}$$

This function is shown in Fig. 5.5. Nomura *et al.* [52] used this example to test their pattern-by-pattern learning approach for an IMF-FLS using triangular membership functions. They achieved a training error of $O(10^{-3})$ with an IMF-FLS containing 80 adjustable parameters, a similar error, at the expense of largely increased computation, was also achieved with a neural network containing a comparable number of weights. The same example was also used by Shi and Mizumoto [75] in comparing an IMF-FLS and a FLS like the one described in Section 5.2. In the case of an IMF-FLS with 45 parameters they report training errors of $O(10^{-2}\text{-}10^{-3})$, while in the case of an FLS with 31 parameters they report errors of $O(10^{-3})$. They report better training times and generalization with the FLS.

We chose N = 49 training points equally spaced in $[-1,1]\times[-1,1]$. The FLS is characterized by K_1 = 4 and K_2 = 4 partitions on each of the two inputs. A full rule base with R = 16 rules and a constant consequent per rule was chosen. This yields a total of 16 antecedent adjustable parameters and 16 consequent adjustable parameters. In the PBP algorithm we chose different constant step-sizes (as in [75]) for training of centers and widths of the triangular membership functions and for the consequents. The antecedent parameters were trained using $\eta_{c,b} = 0.01$ and the consequents using $\eta_\delta = 0.1$. Note that this corresponds to using the same constant step-size of 0.01 for all parameters along with a pre-scaling of the gradient in the direction of the consequent parameters (i.e., pre-multiplication of the gradient by a diagonal matrix having unit values in correspondence of the antecedent parameters and value of 10 in correspondence of the consequents). In the LMQ algorithm we selected $\eta_{max} = 4$ and $\eta_{min} = 0.1$, while in the LMRQ we chose $\eta_{max} = 6$ and $\eta_{min} = 0.1$. For both LMQ and LMRQ we also selected $Q = 5\times32$, and $\varepsilon = 10^{-4}$. The algorithms are all stopped after 400 epochs.

Contrary to [75], where the initial values for the antecedents are set in such a way that the membership functions equally partition the input space and the consequents are all set equal to 0.5, we chose some random initial conditions obtained by independent and identically distributed perturbations of the ones in [75]. Namely, indicating with $U[a,b]$ a

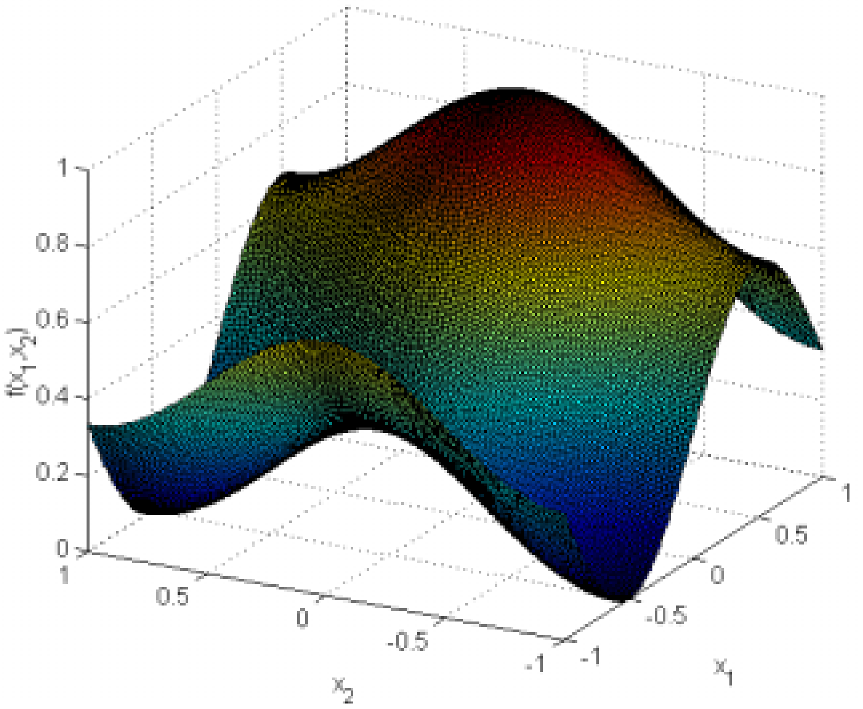

Figure 5.5. Sinusoidal test function for Example 2

uniform distribution in [a,b], the initial values of the adjustable parameters are set as follows

$$\begin{array}{lll} c_{ij} = -1 + \dfrac{2}{R_i - 1}(j-1) + \varepsilon_{ij}^c & & i = 1,2 \\ & \varepsilon_{ij}^c \sim U[-0.4, 0.4] & j = 1,2,\ldots,R_i \\ b_{ij} = \dfrac{4}{R_i - 1} + \varepsilon_{ij}^b & \varepsilon_{ij}^b \sim U[-0.4, 0.4] & k = 1,2,\ldots,R_1 \\ \delta_{kl} = 0.5 + \varepsilon_{kl}^\delta & \varepsilon_{kl}^\delta \sim U[-0.2, 0.2] & l = 1,2,\ldots,R_2 \end{array} \tag{5.29}$$

During the evolution of the algorithm the ***c*** and Δ parameters are unconstrained, while the ***b*** parameters are lower bounded by a minimum width (we chose 0.2), but this constraint never became active in any of the runs.

A sample run of the algorithms yields the convergence history shown in Fig. 5.6. We can see that BOS, PBP and LMQ are all equally fast in the first stages of the optimization. Both BOS and PBP get trapped at a MSE of $O(10^{-3})$, while LMQ undergoes some oscillations and manages to find a very steep descent of more than 4 orders of magnitude (and the MSE is still slowly decreasing). The LMRQ is slower than all the other algorithms in the initial stages, but it manages to achieve a final MSE slightly smaller than the one of PBP and BOS. The increased number of points with respect to the previous case shows

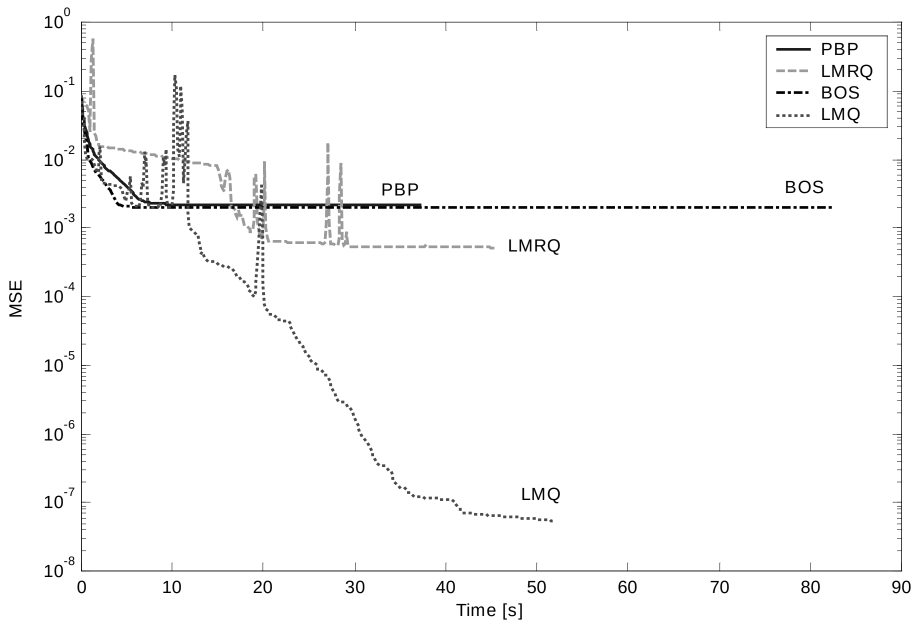

Figure 5.6. Convergence history for PBP, LMRQ, LMQ, and BOS; Example 2

more clearly the overhead of the BOS in executing the 400 iterations; this is compensated by convergence speed. The BOS has an increased execution time with respect to the PBP by about 120% (82 s versus 37 s). Both LMQ and LMRQ also present smaller overheads of 40% and 20%, respectively (52 s and 45 s). The increased overhead of LMQ and LMRQ for this case is due to the increased number of adjustable parameters. This increase does not depend on the number of training points though, and it decreases (in a relative sense) with an increased number of data points. For example, with 121 training points the overhead of the BOS with respect to the PBP increases to 150% (218 s versus 89 s), while those of LMQ and LMRQ decrease to 19% and 12%, respectively (106 s and 100 s).

The cases in Fig. 5.6 are representative of some of the situations that were encountered during the experimentation. In other situations all the algorithms converged to solutions of $O(10^{-3})$. In analogy to the Monte-Carlo analysis performed in the previous example, we tested the behavior of PBP, LMQ, and LMRQ (BOS was excluded in order to reduce the computational time) for 500 runs starting from random initial conditions obtained by (5.29). A histogram of the final MSEs is shown in Fig. 5.7. Both LMQ and LMRQ perform in a similar way, yielding a significantly higher number of times where the

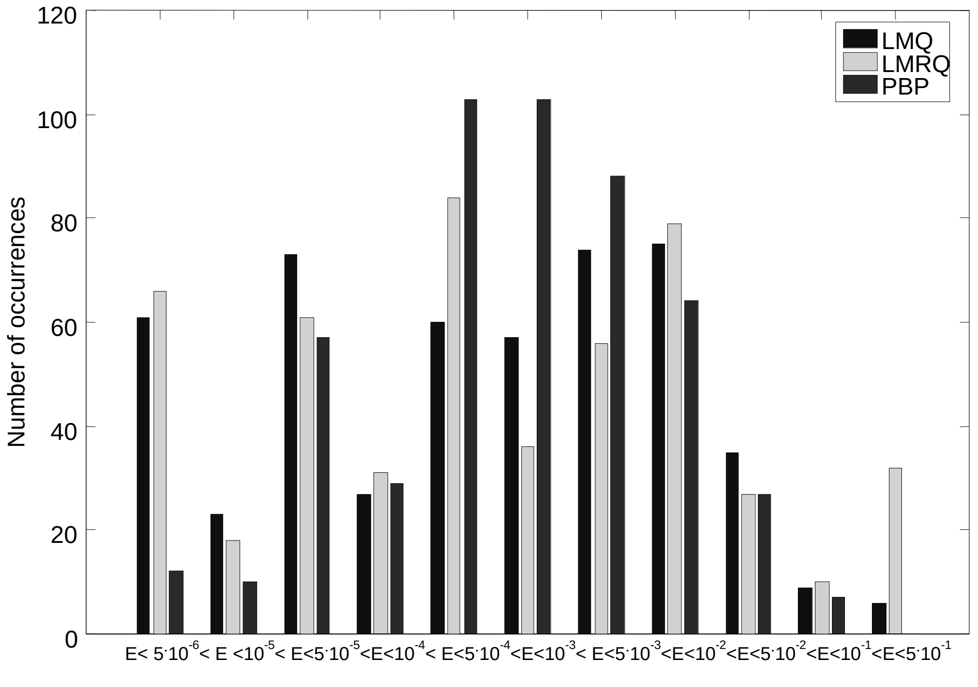

Figure 5.7. Density of final MSE for PBP, LMRQ, and LMQ; Example 2

algorithm achieved very small errors. For example, both LMQ and LMRQ achieve final MSEs smaller than $5 \cdot 10^{-6}$ about 60 times, that is, about 5 times than with the PBP algorithm. Conversely, both LMQ and LMRQ (especially LMRQ) terminate with a high MSE (between 0.1 and 0.5) about 5 and 30 times, respectively. In this particular experiment all the training algorithms presented some oscillations, especially the LMQ and LMRQ. The oscillations could be limited by decreasing the minimum and maximum allowable step-sizes, but for LMQ and LMRQ this implied sometimes converging to a higher MSE. Indeed, these oscillations are often useful for the algorithm to leave a flat error area for an initially higher error, but then settling to a lower error. Therefore, a high final MSE is due to the fact that the maximum number of epochs was reached when the algorithm was on a high MSE value in one of the oscillations.

Using the results from the same Monte-Carlo simulations, we plotted the density of the minimum MSE in the evolution of each of PBP, LMQ, and LMRQ. Note that this is one of the algorithmic variations described in the previous Section 5.2. The corresponding plot is shown in Fig. 5.8. The high values of the MSE for LMQ and LMRQ have indeed disappeared; there are a few occurrences of MSE between 0.05 and 0.1 but they are

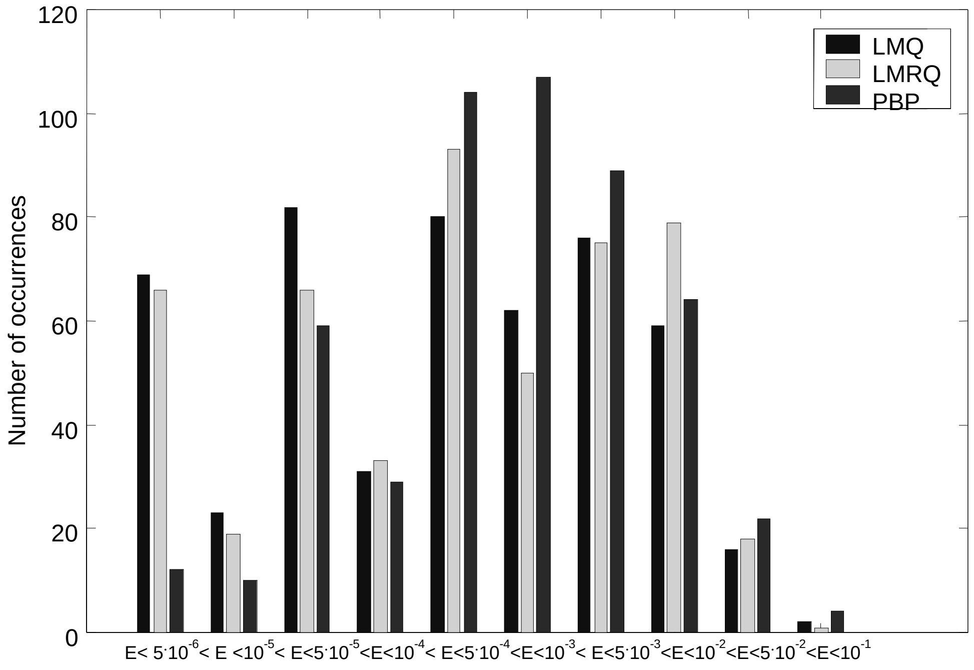

Figure 5.8. Density of minimum MSE for PBP, LMRQ, and LMQ; Example 2

negligible, and also higher for PBP than for LMQ or LMRQ. Both LMQ and LMRQ offer a very low MSE for a significant number of times more than the PBP. Errors smaller than $5 \cdot 10^{-6}$ are now encountered about 70 times with both LMQ and LMRQ, as opposed to about 10 times with PBP. Most of the errors achieved with the PBP are between 10^{-4} and $5 \cdot 10^{-3}$. Both LMQ and LMRQ exhibit similar performances, with the LMRQ algorithm being outperformed by the LMQ. In this case, the LMQ algorithm always offers a slightly higher number of low errors than the LMRQ. Moreover, the LMRQ presents the highest number of errors between 0.005 and 0.01. The advantage offered by both LMQ and LMRQ over PBP is also measured in an average sense by the ratio between the minimum error achieved with PBP and by LMQ or LMRQ; this average ratio is 225 and 133 for LMQ and LMRQ, respectively. Therefore, on average, both quadratic fit algorithms offer a minimum MSE that is two orders of magnitude smaller than that of the PBP. This is an average measure and, as discussed in the previous section, the advantages of LMQ and LMRQ are more obvious in a distribution sense rather than on the average. In this case though, the average ratio is also substantially high. Finally, the higher values obtained with the LMQ shows the better performances offered in this case by using the LMQ algorithm.

Some of these considerations and differences are better illustrated by the cumulative distribution of minimum MSE for PBP, LMQ, and LMRQ shown in Fig. 5.9. It is easily noted that both LMQ and LMRQ largely dominate the solutions obtained with PBP up to levels of the MSE of $5 \cdot 10^{-4}$. The difference starts decreasing after those levels, and the spike in MSE with LMRQ noted above, is apparent with the inversion between LMRQ and PBP for errors smaller than 0.05. This difference is minimal though. The difference between the three algorithms further decreases for increasing errors.

In this example we saw how the LMQ and LMRQ approaches can significantly outperform the PBP method. The MSE found with the PBP is on average of the same order of magnitude as that found in the literature. The LMQ and LMRQ approaches are beneficial in that they manage to offer errors that are on average two orders of magnitude smaller, and in about 15% of the cases, three order of magnitude smaller. Moreover, the computational overhead due to both LMQ and LMRQ is contained and becomes negligible for a large

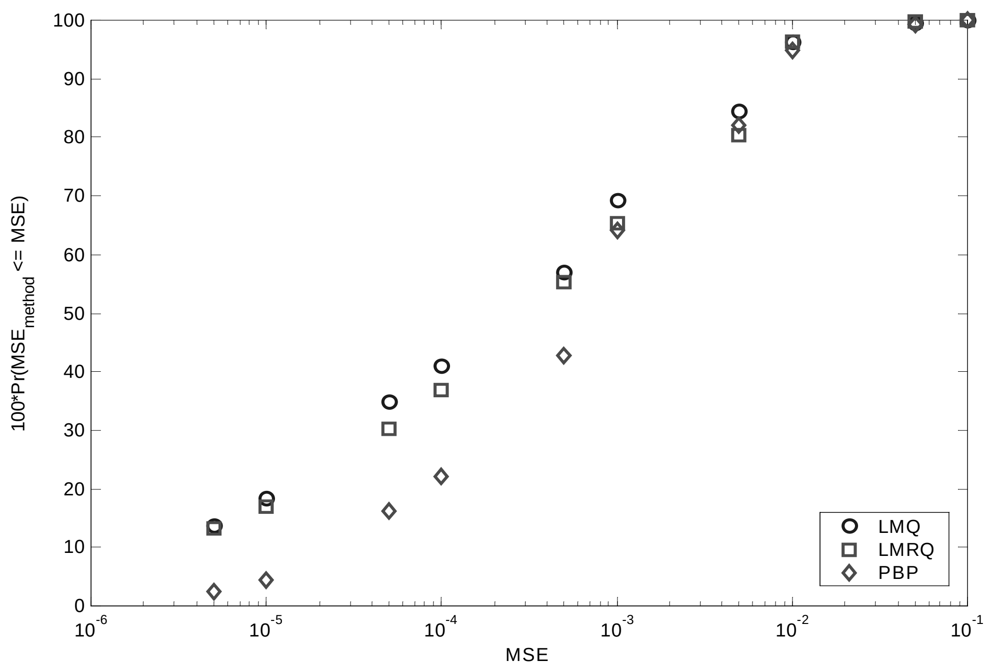

Figure 5.9. Cumulative distribution of minimum MSE for PBP, LMRQ, and LMQ; Example 2

number of training data. Finally, in this case the results confirm the intuition that the LMQ algorithm should perform slightly better than the LMRQ.

5.4.3 Example 3

As a final example we consider the two inputs single output FLS described in Section 5.3 (the same one used in the previous example) to approximate the function

$$f(x_1, x_2) = \frac{1}{2.2419\sqrt{3e^{3x_1} + 2e^{-4x_2}}} - 0.0343 \qquad (5.30)$$

The function in (5.30) is shown in Fig. 5.10. Nomura *et al.* [52] used this example to test their pattern-by-pattern learning approach for an IMF-FLS using triangular membership functions. They achieved a training error of $O(10^{-2}\text{-}10^{-3})$ with an IMF-FLS containing 80 adjustable parameters. A similar error, at the expense of significantly increased computation, was also achieved with a neural network containing a comparable number of weights. Observation of Fig. 5.10 shows that this is a simple problem. The function to approximate is monotone along the x_1 and x_2 directions, moreover, it does not have sudden jumps or extreme nonlinearities.

In analogy to [52] we chose $N = 25$ training points equally spaced in $[-1,1]\times[-1,1]$. We decided to employ a much smaller FLS though, one characterized by $K_1 = 3$ and $K_2 = 3$ partitions on each of the two inputs. A full rule base with $R = 9$ rules and a constant consequent per rule was chosen. This yields a total of 12 antecedent adjustable parameters and 9 consequent adjustable parameters. In the PBP algorithm (exactly as in the previous case) we chose different constant step-sizes for training of centers and widths of the triangular membership functions and for the consequent constants. The antecedent parameters are trained using $\eta_{c,b} = 0.01$ and the consequent using $\eta_\delta = 0.1$. In both the LMQ and LMRQ algorithms we select $\eta_{max} = 2$, $\eta_{min} = 0.1$, $Q = 5\times21$, and $\varepsilon = 10^{-4}$. All algorithms are stopped after 400 epochs. The initial values for the adjustable parameters are set exactly the same way as in the previous case, that is they are randomly initialized according to (5.29). During the evolution of the algorithm the $\boldsymbol{c}$ and Δ parameters are unconstrained, while the $\boldsymbol{b}$ parameters are lower bounded by a minimum width (we chose 0.2). This constraint never became active in any of the runs.

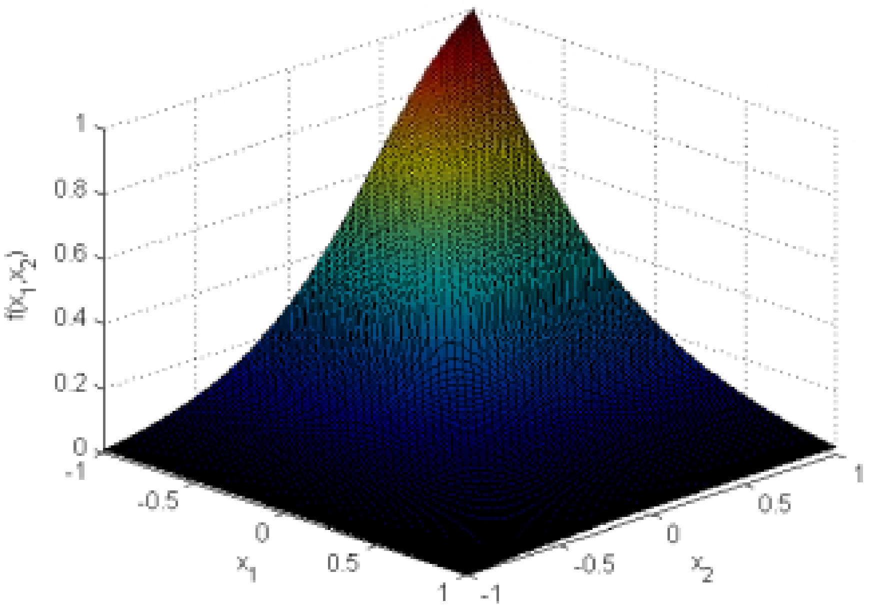

Figure 5.10. Exponential test function for Example 3

A sample convergence history for PBP, BOS, LMQ, and LMRQ is shown in Fig. 5.11. We can see that there are no major differences among the algorithms. Both BOS and LMQ are slightly faster than the other algorithms, with LMQ settling at a MSE of about $2{\cdot}10^{-4}$ and BOS still slightly decreasing. The LMRQ approach is initially fast but then slows down and later recuperates settling to a slightly smaller MSE than PBP. In general, this case did not show any significant differences for the four approaches. Approximating this function is a relatively “simple” task and there is not a big advantage in using any “sophisticated” optimization approach. Indeed, the function is smooth and monotone along the axes directions, and the number of parameters we use is comparable to the number of training points.

Observing Fig. 5.11 we can see that even in this case, with a small number of training data, the overhead of the BOS over the PBP is significant, as it takes BOS about 116% more time than PBP to execute the 400 epochs (41 s versus 19 s). This, increased execution time is compensated by the faster convergence of the BOS. Moreover, the overhead for LMQ and LMRQ is contained to 32% and 16%, respectively (25 s and 22 s). These differences shrink for LMQ and LMRQ and expand for BOS when increasing the number

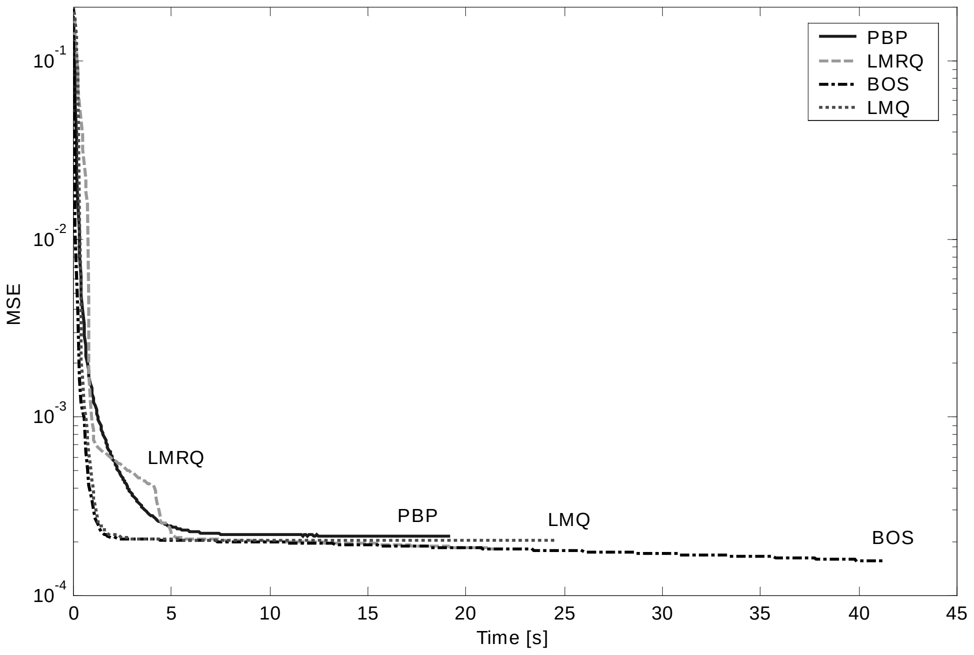

Figure 5.11. Convergence history for PBP, LMRQ, LMQ, and BOS; Example 3

of training points. Using 121 training points BOS has an overhead of 124% (186 s versus 83 s), while LMQ and LMRQ have a small overhead of 7% and 3.5% respectively (89 s and 86 s).

In order to compare the final MSE for PBP, BOS, LMQ, and LMRQ, we executed a Monte-Carlo simulation of these algorithms starting from the described random initial conditions. The errors are collected and a density of final MSEs is plotted for the four algorithms. This bar graph is shown in Fig. 5.12. All the algorithms besides PBP converge most of the time to an MSE between 10^{-4} and $5{\cdot}10^{-4}$. Conversely PBP produces errors that are mostly between 10^{-4} and 10^{-3}. Moreover, BOS, LMRQ, and LMQ yield errors smaller than 10^{-4}, respectively 41, 27, and 9 times, while PBP reaches this level only 3 times.

In this case both BOS and LMRQ are the best algorithms in terms of final MSE, with BOS having a small advantage for the smallest errors. Moreover, LMQ is still competitive with both BOS and LMRQ. The advantages of BOS, LMRQ, and LMQ over PBP are apparent even though in this simple case they are not as obvious as in the previous example. An average measure of the advantage yielded by these algorithms over the PBP is the average of the ratio of MSEs obtained by PBP over BOS, LMQ, and LMRQ, it is 2.2, 1.5,

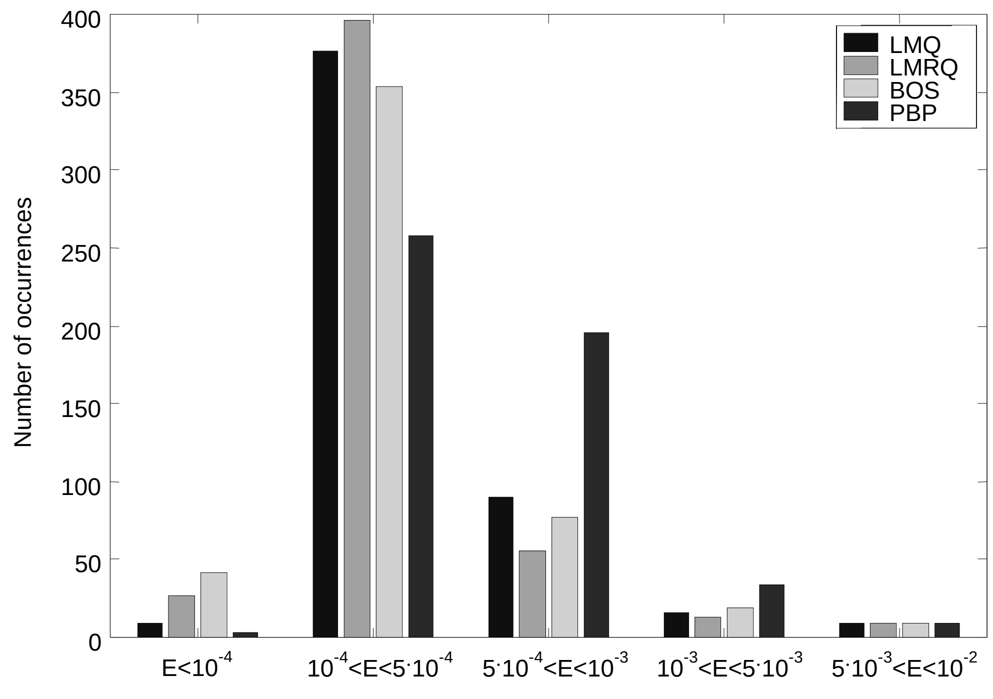

Figure 5.12. Density of final MSE for PBP, LMRQ, LMQ, and BOS; Example 3

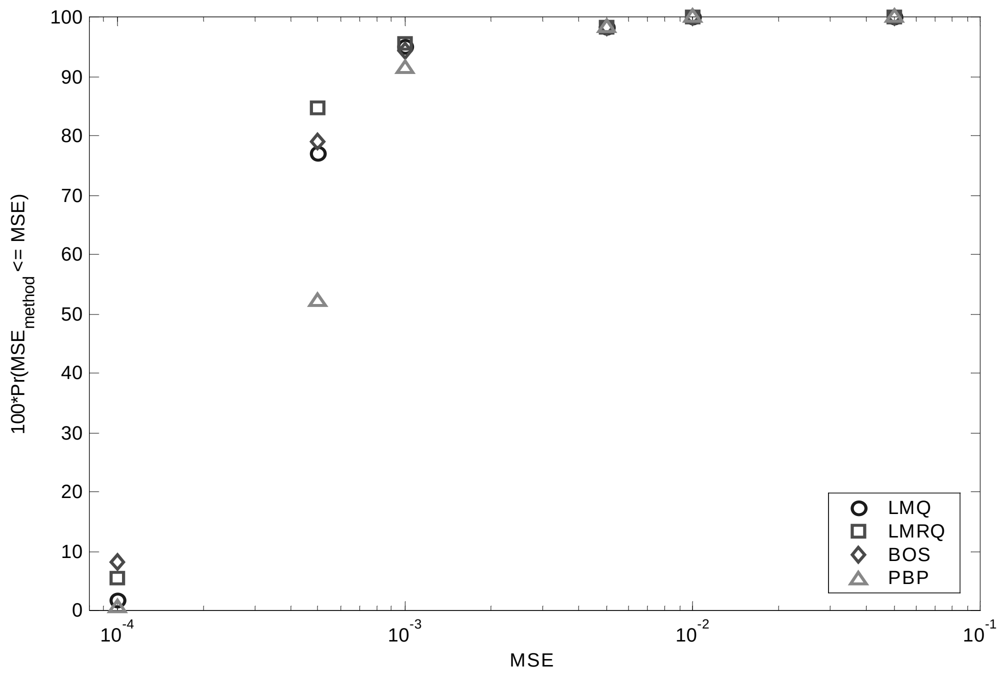

Figure 5.13. Cumulative distribution of final MSE for PBP, LMRQ, LMQ, and BOS; Example 3

and 2.2, respectively. Thus, on average BOS, LMQ, and LMRQ yield MSEs that are about half of the MSEs obtained with PBP. These ratios also show the close performances of BOS and LMRQ and slightly worse performance of LMQ. As we saw in the test case though, LMQ often yields faster convergence.

A comparison of the three algorithms can also be seen in Fig. 5.13, where a cumulative distribution of the MSE for PBP, BOS, LMQ, and LMRQ is shown. From this plot it is easily seen that BOS is initially better than LMQ and LMRQ, with LMQ and LMRQ immediately recuperating and LMRQ becoming the best algorithm.

The errors achieved by the PBP are smaller than those in [52], obtained with an IMF-FLS and a larger number of adjustable parameters. Moreover, the newly introduced LMQ and LMRQ also yield a modest improvement in MSE, even for this easy example. Both LMQ and LMRQ are competitive with BOS, and present less computational overhead, which is scaled with the number of adjustable parameters and not with the number of training points.

Both LMQ and LMRQ were always competitive and offered higher probability to achieve smaller MSEs than the PBP. These methods were used along with a batch gradient descent approach. In the next Section 5.5 we will discuss the possibility of using LMQ and LMRQ with higher order methods, and test some strategies that we will propose to use for supervised learning of FLS.

5.5 Second Order Methods

5.5.1 Introduction

The limited memory quadratic fit that was presented in Section 5.2 is a procedure for prescribing the step-size once an update direction is fixed. This strategy is independent of the update direction as can be seen from the main defining Equation (5.5). In the previous Section 5.4 we tested the limited memory quadratic fit along with a reduced version on a few test cases where batch and pattern-by-pattern approaches were also used. The quadratic fit was employed to select the step-size along the negative gradient direction, or along a deflected version of the negative gradient direction. To illustrate the possibility of use of the limited memory quadratic fit algorithm with strategies other than gradient descent, we propose the use of some easy and efficient second order strategies, which are tested and compared in the following Section 5.5.2, yielding outstanding results on few sample cases. Besides the possibility of using the LMQ with gradient deflection strategies and the corresponding advantages, another interesting aspect of this section is the connection between some novel and well-established strategies in the optimization literature with some similar empirical approaches employed in the neural networks literature. Moreover, some of the strategies were originally proposed in the context of differentiable optimization problems, while others do not require differentiability of the objective function.

Since the first neuro-fuzzy analogies described in Chapter 2 (e.g., [26,27,28,37,80]) many approaches to the supervised learning problem of fuzzy logic systems were borrowed from the neural networks literature. Many of them though have not gained wide spread recognition, leaving the simple pattern-by-pattern approach with constant step-size as one of the most common methods used for supervised learning of FLS. A very common approach to neural network learning consists of using the generalized delta rule [20], whereby the update direction at the *k*-th iterate is given by

$$\boldsymbol{d}_k = -\boldsymbol{g}_k + \mu \boldsymbol{d}_{k-1} \tag{5.31}$$

where

$$\boldsymbol{g}_k = \left.\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\boldsymbol{w}(k)} \tag{5.32}$$

is the gradient of the objective function with respect to the adjustable parameters computed at the *k*-th epoch, and μ is called the *momentum coefficient* (e.g., think of a ball rolling in a valley and being able to pass a subsequent cliff and fall to a lower valley as if it has enough *momentum*). Considering this strategy from an optimization point of view, we can recognize (5.31) as defining a gradient deflection method, where the only difference with traditional strategies is that the momentum coefficient μ (called deflection parameter) is a constant. The strategy used in computing the deflection parameter defines the particular gradient deflection algorithm. Gradient deflection methods are a super-class of conjugate gradient methods, but in this context, due to the non-differentiability, it is improper to talk about conjugate gradient methods. A thorough discussion of conjugate gradient algorithms and their advantages and disadvantages is presented in Bazaraa *et al.* [3].

In the following we will consider the popular gradient deflection method of Fletcher and Reeves [3], in which the update direction is obtained as

$$\boldsymbol{d}_k = -\boldsymbol{g}_k + \frac{\|\boldsymbol{g}_k\|^2}{\|\boldsymbol{g}_{k-1}\|^2}\boldsymbol{d}_{k-1} \tag{5.33}$$

Note that $\|\cdot\|$ denotes a 2-norm. We also use a second gradient deflection strategy that is popular and efficient for both differentiable and non-differentiable problems; proposed by Sherali and Ulular [68], and called the average direction strategy (ADS), it computes the update direction as

$$\boldsymbol{d}_k = -\boldsymbol{g}_k + \frac{\|\boldsymbol{g}_k\|}{\|\boldsymbol{d}_{k-1}\|}\boldsymbol{d}_{k-1} \tag{5.34}$$

Using this choice of deflection parameter, the new update direction bisects the angle between the negative gradient direction and the past direction of movement, hence the name ADS. This strategy was also successfully used in Joshi *et al.* [29] in the context of RSM.

In gradient deflection algorithms the old direction of motion acts as a memory of the system; for this reason it is very often seen that these algorithms are subject to restarts (i.e., set $\boldsymbol{d}_k = -\boldsymbol{g}_k$). In this work, we use the second restarting criterion set (i.e., RSB) discussed in [29], that is, we will restart if any of the two following conditions are met:

1. k = number of variables
2. $\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{g}_k \leq -0.8\, \boldsymbol{g}_k^{\mathrm{T}}\boldsymbol{g}_k$

The first condition represents a regular restart every number of epochs equal to the number of variables (see [3] for more explanations); while the second condition verifies that there is enough descent along $\boldsymbol{d}_k$. Moreover, if prior second-order information is available the negative gradient in (5.31) can be pre-multiplied by a suitable diagonal pre-conditioning matrix that operates a re-scaling of the problem variables. The restarting criteria that were employed are not necessarily the best ones for this type of problem, therefore more investigation in this direction could be employed, although the conditions seemed to work well in our test cases. Moreover, even though in the optimization literature it has been observed that conjugate gradient type algorithms perform better with some restarting, in the neural network literature no restarts are used in conjunction with the use of the generalized delta rule.

In the context of second order algorithms quasi-Newton methods are also very popular. They are all based on trying to approximate the inverse of the Hessian in order to pre-multiply it by the negative gradient [3]. Therefore, in this type of methods the update direction at the k-th epoch is obtained as

$$\boldsymbol{d}_k = \boldsymbol{D}_k(-\boldsymbol{g}_k) \tag{5.35}$$

The difference in the way this matrix $\boldsymbol{D}_k$ is computed yields different quasi-Newton methods. In the context of non-differentiable functions it is improper to talk about Hessian and quasi-Newton methods. Nonetheless, approaches like (5.35) can yield improved algorithmic performances. These approaches correspond to using operators that stretch (i.e., dilate and reduce) the gradient, implementing a variable metric (thus also the name of *variable metric methods*) using present and past information. More details can be found in the excellent introduction to the subject presented by Sherali *et al.* [72]. Oftentimes these methods store the deflection matrix and operate by constantly updating it. To alleviate the storage requirements, *memoryless* methods have been proposed in which information at the current and previous step is used to compute $\boldsymbol{D}_k$. In the following we test a memoryless variant of the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [3] that computes the deflection matrix as

$$\boldsymbol{D}_k = \boldsymbol{I} + \frac{\boldsymbol{p}_k \boldsymbol{p}_k^T}{\boldsymbol{p}_k^T \boldsymbol{q}_k}\left(1 + \frac{\boldsymbol{q}_k^T \boldsymbol{q}_k}{\boldsymbol{p}_k^T \boldsymbol{q}_k}\right) - \frac{\boldsymbol{q}_k \boldsymbol{p}_k^T + \boldsymbol{p}_k \boldsymbol{q}_k^T}{\boldsymbol{p}_k^T \boldsymbol{q}_k} \tag{5.36}$$

where

$$\boldsymbol{p}_k = \boldsymbol{w}_k - \boldsymbol{w}_{k-1} \quad \boldsymbol{q}_k = \boldsymbol{g}_k - \boldsymbol{g}_{k-1} \tag{5.37}$$

In the context of non-differentiable optimization Sherali *et al.* [72] propose a memoryless space dilation and reduction strategy where $\boldsymbol{D}_k$ is obtained as

$$\boldsymbol{D}_k = \boldsymbol{I} - \left(1 - \alpha_k^2\right)\frac{\boldsymbol{q}_k \boldsymbol{q}_k^T}{\boldsymbol{q}_k^T \boldsymbol{q}_k} \tag{5.38}$$

and the coefficients α_k are computed according a proposed strategy (MSDR). For sake of simplicity in this case we use a variation of their approach that they found to be a second choice, but still competitive on some test cases. In this variation (MSD) a fixed dilation parameter $\alpha_k = 1/3$ is used. Moreover, if either the difference between two successive gradients ($\boldsymbol{q}_k$) or the new update direction ($\boldsymbol{d}_k$) are too small, than the negative gradient direction is used. Knowing some prior second-order problem information would also allow us to substitute a diagonal scaling matrix for the identity matrix in Equation (5.38). This, known as Oren-Luenberger scaling technique, has been found to enhance the performance of memoryless and limited memory variable metric methods [38].

5.5.2 Results

In this Section we test the pattern-by-pattern (PBP) and batch gradient using quadratic fit (GRAD) with some second-order methods, all using the LMQ for step-size selection. Namely, we compare the Fletcher-Reeves (FR), ADS (both with the restarting described above), BFGS, and MSD strategies. Moreover, two MSD strategies are considered. In the first (MSD_1) the deflection matrix is computed as in (5.38), while in the second (MSD_2) we used the Oren-Luenberger scaling technique where we employ the knowledge that using a higher learning rate for the FLS consequents improves performances. Therefore, in MSD_2 instead of the identity matrix in (5.38) we use a diagonal matrix, with all unit entries, and values of 10 corresponding to the consequent parameters. Moreover, the same scaling was also used for the two gradient deflection strategies FR and ADS. This same choice was used for all the following three test cases.

The test cases consist of the two-dimensional function approximation problems using the FLS described in Section 5.3 and an additional problem presented in [26]. The initial conditions for all the algorithms are the same and are set to equally partition the input space. That is, we use the initial conditions described in (5.29) with zero perturbations; this corresponds to using initial conditions that are intuitive and that are commonly considered as "good" ones. This choice reduces the randomness in the operation of the algorithms, even though some randomness is still present in the randomization of the order of training points for the PBP, and in the sampling of the objective function for filling the buffer for the LMQ. In all the cases we use N = 121 training points equally spaced in $[-1,1]\times[-1,1]$. The algorithms are stopped after 400 epochs. In the PBP algorithm the antecedent parameters are trained using $\eta_{c,b}$ = 0.01 and the consequents using η_δ = 0.1. In the LMQ algorithm we selected $\varepsilon = 10^{-4}$, W = 3, and the buffer size is always set to 5 times the number of adjustable parameters, that is

$$Q = 5\left[2\left(K_1 + K_2\right) + K_1 K_2\right] \tag{5.39}$$

As a first example, we consider Example 2 from the previous section. That is, we want to approximate the sinusoidal function given by Equation (5.28), using the same FLS, that is $K_1 = K_2 = 4$, thus using a total of 32 parameters (one more than Shi and Mizumoto [75] and significantly less than Nomura *et al.* [41]). In the LMQ algorithm for step-size selection we employ $\eta_{max} = 4$ and $\eta_{min} = 0.5$. A sample convergence history is shown in Fig. 5.14. The improvement in MSE with respect to the PBP can be of as much as three orders of magnitude. Indeed, the PBP converges to an MSE of $8 \cdot 10^{-4}$ (slightly less than what was reported in [52] and [75]), but all the other approaches decrease the MSE by a factor as little as 3 and as large as 1000.

All of the algorithms (besides MSD_1) initially quickly converge and get temporarily trapped at a sub-optimal value of the MSE of 0.002. This is exactly the final MSE found by Shi and Mizumoto [75] using a pattern-by-pattern approach. All of the algorithms manage to escape this plateau (PBP included); the PBP does so only to get trapped into another plateau very soon afterwards. The simple gradient approach (GRAD) takes a little longer to leave that plateau and achieves an MSE that is a little more than one third of that of the PBP, moreover the error is still decreasing, thus using a higher number of iterations might have helped. Both MSD algorithms perform similarly in terms of final MSE; they yield a

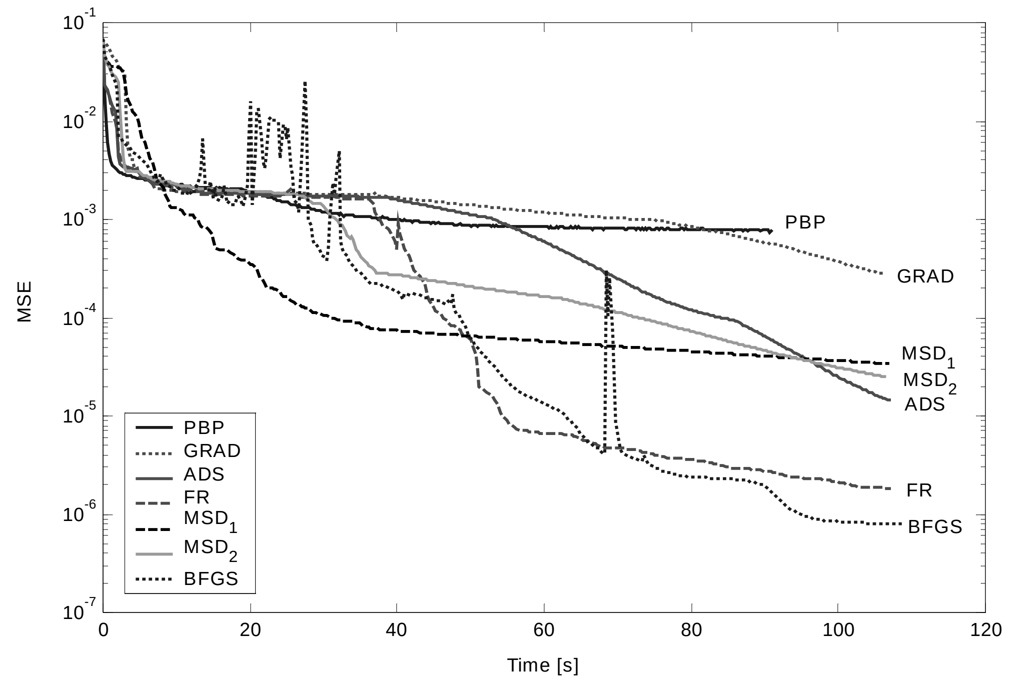

Figure 5.14. Convergence history for PBP, GRAD, FR, ADS, BFGS, and MSD; Case 1: sinusoidal function

final MSE that is about 1.3-1.5 orders of magnitude smaller than that of the PBP, with the error still decreasing. It is interesting to note the different path they follow to reach this error. Indeed, MSD_2, like most of the other algorithms, converges quickly and gets trapped at the first plateau, managing to escape it later. In contrast, MSD_1 starts convergence much slower (since it does not have the problem information contained in the pre-conditioning matrix), but does not get trapped at all in the plateau where all the other algorithms temporarily stop. Therefore, its initially slower rate of convergence makes it faster than all the other approaches in successive stages of the algorithm due to the fact that is does not get trapped in the plateau. The rate of decrease of error in the final stages of the algorithms seems to show a potentially lower MSE for the MSD_2 than for the MSD_1 in a longer run. This number of epochs was selected in order to try to contain the already high computational burden in comparing all the strategies. The ADS offers a final MSE that is about 1.7 orders of magnitude smaller than that obtained by PBP. Moreover, the rate of decrease of the error in the final epochs is still quite large. The best approaches for this case are the FR and the BFGS that yield improvements of about 2.5 and 3 orders of magnitude, respectively, with respect to the PBP. The BFGS exhibits some innocuous oscillations.

In terms of total execution time (rather than convergence speed) the overhead with respect to the PBP of all of the gradient and second-order algorithms employing the LMQ step-size selection, is not substantial (about 16%). Moreover, none of the second-order methods seems to impose an observable overhead on the learning time (total execution time less than 1% larger than for GRAD). Therefore, these second order strategies can take advantage of the LMQ step-size selection by further reducing the MSE and improving convergence, but without imposing any significant overhead.

As a second example we consider Example 3 from the previous section. That is, we want to approximate the exponential function given by Equation (5.30). We use the same FLS, with $K_1 = K_2 = 3$, thus using a total of 21 parameters (significantly less than Nomura *et al.* [52]). In the LMQ algorithm we employ $\eta_{max} = 2$ and $\eta_{min} = 0.5$.

A sample convergence history is shown in Fig. 5.15. The improvement in MSE with respect to the PBP can is very small. Indeed, as already discussed in the previous Section,

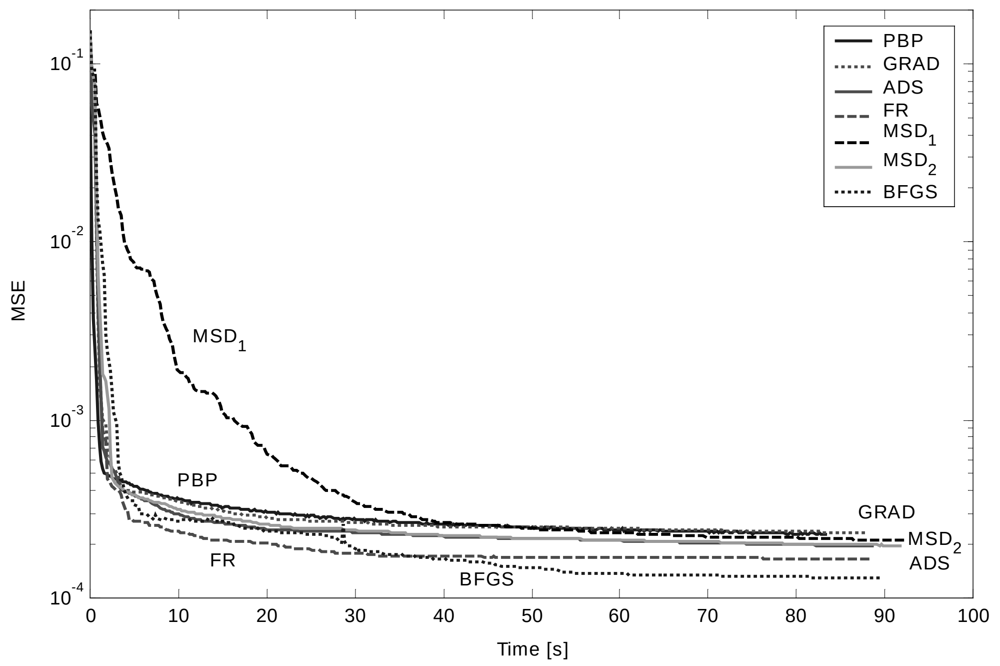

Figure 5.15. Convergence history for PBP, GRAD, FR, ADS, BFGS, and MSD; Case 2: exponential function

this is a simple example that does not seem to produce big differences between possible optimization approaches due to its simplicity. Both PBP and GRAD converge to the highest MSE. Slightly lower MSE values are obtained with MSD$_1$, MSD$_2$, and ADS. Once again, MSD$_1$ exhibits slower convergence than MSD$_2$, and slower than all the other algorithms. The best errors are obtained with FR and BFGS, which also offer fast convergence as well.

The overhead of the LMQ approach is as small as 6% for GRAD with respect to the PBP. The additional overhead of the second-order approaches with respect to the GRAD were: smaller than 0.5% for FR and ADS, smaller than 4% for the MSD algorithms and of about 1% for BFGS. Small numbers of this type are not significant since they might be strongly affected by the errors in discussed at the beginning of Section 5.4.

As a (final) third example we consider the problem of approximating a two-dimensional sinc function given by

$$f(x_1, x_2) = \frac{\sin(10x_1)}{10x_1}\frac{\sin(10x_2)}{10x_2} \tag{5.40}$$

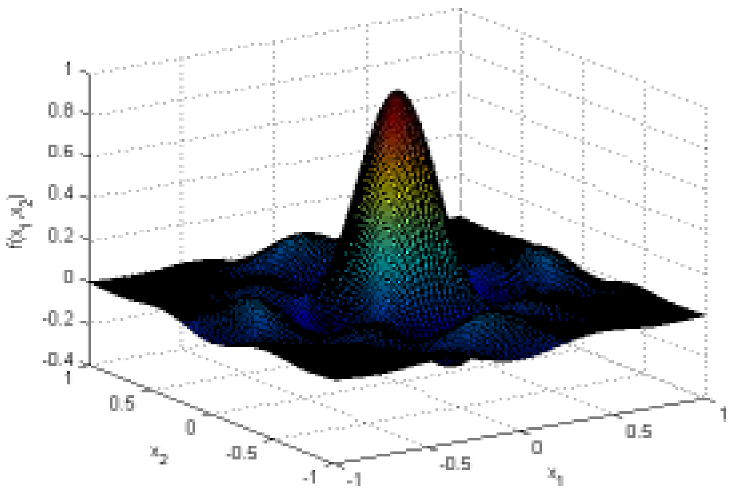

Figure 5.16. Sinc test function

A graphical representation of this function is shown in Fig. 5.16. This problem was used by Jang [26] to test his ANFIS. He achieves errors of $O(10^{-4})$ employing an ANFIS that uses 72 parameters. Moreover, he compares the performances of the ANFIS with a neural network with similar number of weights and trained with quick propagation and reports an error of $O(10^{-2})$. We use an FLS with $K_1 = K_2 = 5$, thus using a total of 45 parameters (significantly less than Jang [26]). In the LMQ algorithm for step-size selection we employ $\eta_{max} = 2$ and $\eta_{min} = 0.5$.

A sample convergence history is shown in Fig. 5.17. The PBP terminates at an error of about 10^{-3}, thus higher than the one obtained in [26], but achieved using a significantly smaller number of parameters. All of GRAD, ADS, and MSD$_2$ achieve a similar final MSE. The MSD$_1$ converges slower (as in the other cases) and to a slightly higher MSE. An excellent result is produced by both BFGS and FR. Using BFGS, we achieve a final error that is more than two orders of magnitude smaller than the one by PBP (and also smaller than the one reported in [26]) and there still seems to be some non-zero error decrease rate. The FR strategy rapidly converges to a final error that is 7 orders of magnitude smaller than the one obtained with PBP (and also much smaller than the one in [26] using a larger number of parameters)! The objective function seems to have many different solutions that yield a similar MSE of 10^{-3}. All of the algorithms get trapped in one of those sub-optimal

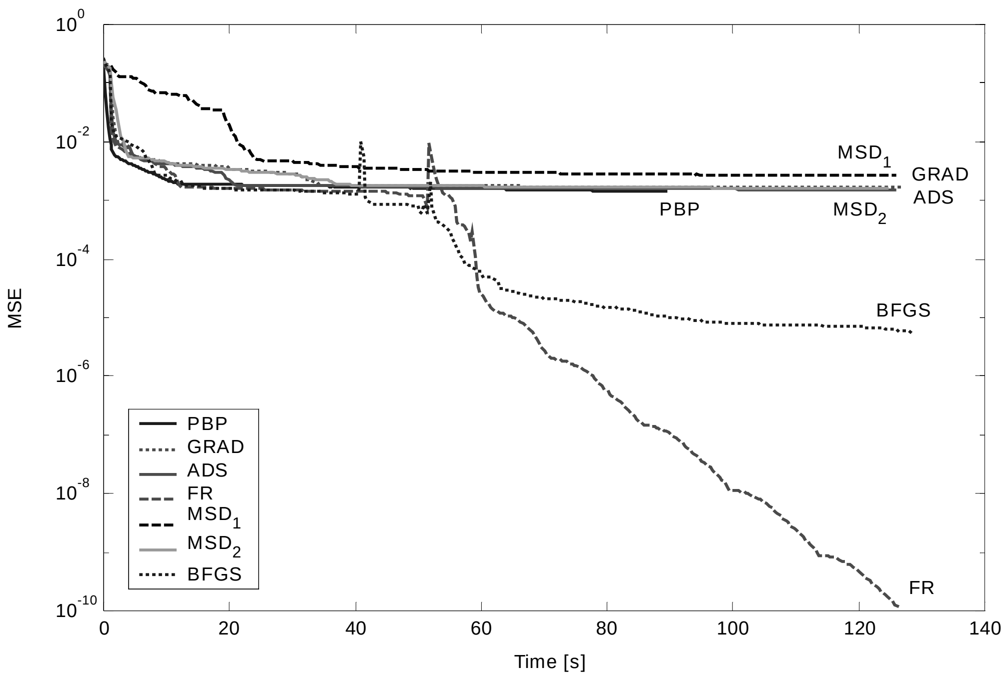

Figure 5.17. Convergence history for PBP, GRAD, FR, ADS, BFGS, and MSD; Case 3: sinc function

solutions, but the BFGS and FR manage to escape them. In the case of the BFGS it is one of the oscillations that pushes the algorithm away from the influence of one of the sub-optimal solutions. The shape of this objective function is probably due to the nature of the sinc function. Indeed, a good solution in terms of MSE will be achieved by a good approximation of the main lobe. Any approximation of the side lobes will affect the MSE to a lesser extent.

In this case the computational overhead due to using the LMQ algorithm is more pronounced, due to the increase in the number of adjustable parameters (while the number of training points is constant). However, this overhead is still contained to about 50%, and the additional cost of second-order methods is negligible.

5.6 Conclusions

In this chapter we proposed a novel step-size selection technique that uses successive quadratic fits of past values of the mean square error function we seek to minimize. A reduced variation of this technique was proposed as well. A FLS with two inputs and two outputs and one constant consequent per rule was introduced with both its defining and training equations. The novel aspect of this formulation is its matrix form that lends itself to an efficient implementation.

Both the limited memory quadratic fit and its reduced version were tested on some sample cases versus constant step-size pattern-by-pattern training and batch training with quadratic line search. The use of either one of the quadratic fit strategies showed a significant improvement in terms of the distribution of the final errors obtained by the algorithms when starting from random initial conditions. Moreover, the computational overhead imposed by the quadratic fit was also modest and always smaller than the one corresponding to a batch algorithm using optimal step-sizes, since in the latter case the number of computations is scaled with the number of training points, while in the former, the computational complexity ultimately depends on the number of adjustable parameters.

The proposed LMQ fit is a general step-size selection approach and, as such, it can be used in conjunction with any update direction. Therefore, we also tried to use it to determine a step-size, not only in the negative gradient direction, but also along several other directions modified according to strategies of the second-order type. Namely, we used the LMQ fit in conjunction with the Fletcher-Reeves and the average direction strategy gradient deflection algorithms, with the memoryless version of the Broyden-Fletcher-Goldfarb-Shanno method and with a simplified version of the memoryless space dilation and reduction algorithm by Sherali *et al.* [72]. The application of second-order approaches with the quadratic fit step-size selection enhanced convergence characteristics as well final MSE (by as much as 7 orders of magnitude)! In our experiments the Fletcher and Reeves conjugate gradient performed the best. However, this by far should not be considered an extensive and complete experimentation of the use of second order methods, but rather a proof of concept, of feasibility, and performances in using the proposed limited memory quadratic fit. The LMQ fit can, not only revitalize the use of batch training yielding improved performance, but it can also further improve performances using it in conjunction with second order approaches.

In our experimentation the choice of pre-conditioning the gradient by multiplication with a diagonal matrix was very successful. This is indeed a common approach, that is, to use different learning rates per logical group of parameters, with the highest learning rate generally assigned to the consequent parameters. This raises interesting questions regarding the scaling of the problem and the structure of the eigenspace of its Hessian (when it exists). A definite direction for future research consists of understanding the reasons for this common choice and its success; that is, analyzing the problem scaling in connection with its structure, and perhaps revealing some general properties. Moreover, Hessian information could also be used, not only to select an update direction, but also to determine a new basis for the adjustable parameters, in which the model of the limited memory reduced quadratic fit should always be adequate.

The effect of the size of the buffer for the quadratic fit, along with the frequency of update of the vertex might also be an interesting study to perform. Indeed, it should be possible to reduce the frequency of these updates rather than having them once per epoch. Moreover, the number of past data used in performing the quadratic fit could also be decreased, especially in the case of the reduced quadratic fit. This would obviously increase the efficiency and decrease the storage requirements of the algorithm.

Chapter 6
Conclusions

6.1 Conclusions

This dissertation considers the design optimization of fuzzy logic systems (FLS) when data, containing information on the desired behavior of the FLS, are available (i.e., supervised learning). This type of problem has widespread applications for control systems, system identification, and, in general, any application were fuzzy systems can be, and have been used with success.

The supervised learning problem consists of adjusting the FLS parameters in order to minimize a given error criterion in approximating the given data. Therefore, this is a nonlinear least squares problem of fuzzy type, a nonlinear optimization problem that, in some cases, is non-differentiable. In this work we examined the approaches presented in the literature and generated a taxonomy of approaches to which the proposed formulation can be applied by simply changing the value of some of its defining parameters. After discussing the potential problems and showing the ones connected with non-differentiability, we also proposed a new problem formulation and analyzed its properties. Finally, we proposed a new algorithm for design optimization of FLSs and tested it versus the commonly used method in several test cases, showing the advantages of the proposed approach.

In Chapter 1 we introduced fuzzy sets and fuzzy logic and, finally, formulated fuzzy logic systems. A formulation for the output of the system is given, and it is showed how this formulation can model Mamdani as well as Takagi-Sugeno fuzzy systems. This chapter mainly serves as the background for the reader as well as the introduction for the notation.

In Chapter 2 we introduced the problem of interest and showed that the supervised learning of FLS is merely a nonlinear programming problem. We extensively reviewed the literature on the subject and identified some significant issues with the reviewed approaches, namely:

- *Non-differentiability* of the objective function when using piecewise linear membership functions, minimum, or maximum operators;
- How to operate a suitable *step-size selection*, instead of using a constant step-size as customary;
- Strong dominance of *pattern-by-pattern* with respect to *batch training*;
- Existence of *local* approaches and weak attempts at finding *global* solutions;
- Complete lack of use of *higher order methods* or deflection techniques in general (e.g., conjugate gradient, quasi-Newton, etc.);
- Weak connection between the present approaches and considerations on the best *types of membership function*;
- Poor *readability* of the solutions generated by training;
- Lack of standardized *test cases*.

A subdivision between Mamdani FLSs, TS-FLSs, and IMF-FLSs was explicitly stated and a common general problem formulation was introduced. Moreover, equations for batch and pattern-by-pattern gradient descent approaches were derived from the general model; providing a novel common framework for the optimization of these three different types of fuzzy systems.

In the following Chapter 3 we focused on one of these problems, namely non-differentiability. Through two very simple examples we showed that in the case of non-differentiability of the objective function, we can observe divergence of the learning as well as a slowed convergence due to excessive proximity to points of differentiability. In these cases the gradient no longer exists at all points in the search space. Any arbitrary direction that replaces the gradient at the points of non-differentiability might not be a descent direction and might cause the algorithm to get trapped at a point of non-differentiability, or to diverge. Indeed, if non-improving steps are taken and their effects accumulate, divergence can be observed. Moreover, if an exact line search is conducted and the update direction for the parameters is not improving, then the optimal step-size will be zero. Finally, in Chapter 3 we also showed that the transition from a differentiable to a non-differentiable point happens smoothly, that is, approaching a point of non-differentiability that yields a non-improving direction, the optimal step-size on the gradient directions shrinks, and thus the chance of overstepping by using a constant step-size increases. This constitutes a practical issue of the non-differentiability, besides the obvious theoretical one.

In Chapter 4 we reformulated the supervised learning problem in a form possibly more suitable to global optimization approaches. Instead of trying to minimize the quadratic approximation error, we manipulated the equation in order to obtain another type of error that we called the *equation error*. Using this error as the minimizing function, the objective function becomes polynomial in both the membership degrees and the consequent parameters. Moreover, when piecewise linear membership functions are employed, the objective function becomes piecewise polynomial in the antecedent parameters. With Gaussian or bell-shaped membership functions the objective function becomes factorable in univariate functions. An expanded formulation for triangular membership functions, including suitable constraints and integer variables, is also proposed. Using this formulation, piecewise linear membership functions become polynomial in their adjustable parameters. Thus, the reformulated problem, which is piecewise polynomial in the membership functions' parameters, becomes polynomial in those parameters. The cost of eliminating the piecewise nature of the problem consists of an increased number of parameters, as well as additional constraints. The use of a new reformulation and linearization technique (RLT) [71] for polynomial or factorable problems was proposed. Unfortunately, the increase in the number of variables in the polynomial formulation of triangular membership functions, along with the expanded dimensionality of the RLT, makes its use not feasible. Nonetheless, the ideas presented can be eventually further developed in other directions.

A direct optimization of the membership degrees could be sought, and a determination of the parameters of some membership functions to approximate these optimal membership degrees could be successively performed. The advantage of such an approach would be to decouple the first optimization problem from the type of membership functions. The dimension of this problem would still be large however. Another possible approach would also be to use an alternate piecewise linear membership function reformulation that does not increase dimensions as much as the proposed one. These are open research directions for the future.

In Chapter 5 we proposed a more pragmatic approach to the problem, introducing a method that we call the limited memory quadratic fit. In this approach past values of the objective function in the convergence of the algorithm are used. A limited second order model is fitted to these data in a fashion similar to response surface methodologies. Unlike RSM though, the samples are not obtained by a careful experimental design since we use the ones that are already available, i.e., the points in parameter space that the algorithm already visited. The position of the vertex of the fitted paraboloid is used to determine the step-size along the negative gradient direction, in order to minimize the distance of the next point in the algorithm from the vertex itself. This choice of step-size is similar to what is used in non-differentiable optimization problems when moving along a subgradient direction; it is motivated by the higher reliability of the gradient than the quadratic fit, that offers qualitative information. This approach has a more global nature than other step-size selection approaches; moreover, it does not experience problems connected to the non-differentiability. Using this approach, the computation of the step-size along the negative (batch) gradient direction becomes scaled with the number of parameters and no longer with the number of training points. Therefore, our approach has the effect of making the use of batch training techniques more computationally attractive.

We formulate the output and training equations for a two-input one-output Takagi-Sugeno FLS with constant local models in an attractive and efficient matrix form. We use this FLS along with a SISO FLS to compare our quadratic fit approach, along with a reduced version, to both pattern-by-pattern using constant step-size and batch training with optimal step-size. The results indicate a higher consistency of both our approaches with batch gradient in finding a final mean square error smaller (of at least one order of magnitude) than the one obtained by using the common pattern-by-pattern when starting from the same random initial conditions. Moreover, the results of our approach are comparable and sometimes better than the ones obtained using batch training with optimal step-size. In terms of computational efficiency, the quadratic fit approach imposes a very small overhead with respect to pattern-by-pattern. Moreover, the higher convergence rate generally compensates for this overhead.

The introduction of the quadratic fit approach and the consequent revitalization of batch training stimulates the use of higher order methods in order to accelerate convergence. Therefore, we also test the use of several second order methods (or deflection and variable metric methods, since in precise terms the problem is not differentiable) in conjunction with the quadratic fit. Second order methods are used in order to generate an update direction, while the quadratic fit is used to determine the step-size along this direction. The application of second order methods in conjunction with the quadratic fit is then tested on some examples and its final errors and convergence characteristics are compared to the use of a pattern-by-pattern approach. The increased speed and lower final errors are significant. Indeed, lower errors (from 1 to as much as 7 orders of magnitude) are achieved consistently.

Some questions are still open regarding the common use of some pre-conditioning of the gradient. Namely, the step-sizes along the consequent parameters are generally set at higher values than those for the antecedent parameters. A connection of this common and successful use to the structure of the problem is a very interesting avenue for further research. This is connected to the scaling of the problem, and can be seen in the eigenspace of the Hessian of the problem (i.e., where it exists). Moreover, a thorough second order analysis could lead to improved characteristics of the model on which the quadratic fit is based. Indeed, Hessian information could be used to refine the type of model used in the quadratic fit, and eventually also to redefine a basis for the variables used in the quadratic fit. This could lead to further improved efficiency and convergence of the quadratic fit.

6.2 Summary of Contributions

In summary, the contributions of this dissertation are:

- Providing an extensive literature review and: 1) synthesizing this information into a list of issues with the state of the art; 2) identifying different types of fuzzy systems[1] (e.g., IMF-FLS) and providing with a common model for their analysis and design.
- Showing the problems presented by non-differentiability in some concrete cases, illustrating the possibility of both divergence and slowed convergence of training. These ideas have never been discussed and presented in the literature.
- Proposing a novel problem formulation characterized by enhanced properties (i.e., polynomial or factorable), and showing how it can lead to global design optimization of FLS.
- Proposing and testing with both gradient and second order methods a novel step-size selection approach (limited memory quadratic fit) that offers more global convergence characteristics (consistently lower errors), enhanced convergence speed, and slightly larger cost per iteration. This approach is new and should be useful to practitioners in using batch training more often and in improving training performance.

[1] Please note that the discussion on IMF-FLS is “almost” novel. It was conceived at the beginning of 2000 and at that time it was novel (to the author). At the same time Shi and Mizumoto [74] independently published a paper presenting similar considerations.

References

[1] P. Arabshahi, R.J. Marks, S. Oh, T.P. Caudell, J.J. Choi, and B.G. Song, “Pointer Adptation and Pruning of Min-Max Fuzzy Inference and Estimation,” *IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing*, 44(**9**), 696-709, September 1997

[2] M. S. Bazaraa and, H. D. Sherali, "On the choice of step size in subgradient optimization", *European Journal of Operations Research*, 7, 380-388, 1981

[3] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty, *Nonlinear Programming*, John Wiley & Sons, 2nd edition, 1993

[4] H. Bersini, and V. Gorrini, “An Empirical Analysis of one type of Direct Adaptive Fuzzy Control,” *Fuzzy Logic and Intelligent Systems*, H. Li, and M. Gupta, Editors, Chapter 11, 289-309, Kluwer

[5] H. Bersini, and G. Bontempi, “Now Comes the Time to Defuzzify Neuro-Fuzzy Models,” *Fuzzy Sets and Systems*, 90, 161-169, 1997

[6] J.C. Bezdek, “Editorial: Fuzzy Models – What Are They, and Why?,” *IEEE Transactions on Fuzzy Systems*, 1(**1**), February 1993.

[7] M. Black, “Vagueness: An Exercise in Logical Analysis,” *Philosophy of Science*, 4, 427-455, 1937

[8] G.E.P. Box, and K.B. Wilson, “On the experimental attainment of optimum conditions,” *Journal of the Royal Statistics Society*, B13, 1-38, 1951

[9] B.C. Cetin, J. Barhen, and J.W. Burdick, “Terminal Repeller Unconstrained Subenergy Tunneling (TRUST) for Fast Global Optimization,” *Journal of Optimization Theory and Applications*, 77(**1**), April 1993

[10] P. Dadone, H.F. VanLandingham, and B. Maione, “Modeling and Control of Discrete-Event Dynamic Systems: a Simulator-Based Reinforcement-Learning Paradigm,” *International Journal of Intelligent Control and Systems*, 2(**4**), 609-631, 1998

[11] P. Dadone, and H.F. VanLandingham, “Non-differentiable Optimization of Fuzzy Logic Systems,” *Proceedings of ANNIE 2000: Smart Engineering System Design*, St. Louis, MO, November 5-8, 2000

[12] P. Dadone, and H.F. VanLandingham, “On the non-Differentiability of Fuzzy Logic Systems,” *Proceedings of IEEE Conference on Systems, Man and Cybernetics 2000*, Nashville, TN, October 8-11, 2000

[13] P. Eklund, and J. Zhou, “Comparison of Learning Strategies for Adaptation of Fuzzy Controller Parameters,” *Fuzzy Sets and Systems*, 106, 321-333, 1999

[14] P.Y. Glorennec, “Learning Algorithms for Neuro-Fuzzy Networks,” *Fuzzy Control Systems*, A. Kandel, and G. Langholz, Editors, pp. 4-18, CRC Press, Boca Raton, FL, 1994

[15] D. Gorse, A.J. Shepherd, and J.G. Taylor, “The new ERA in Supervised Learning,” *Neural Networks*, 10(**2**), 343-352, 1997

[16] F. Guely, and P. Siarry, “Gradient Descent Method for Optimizing Various Fuzzy Rule Bases,” *Proceedings of the second IEEE Conference on Fuzzy Systems*, 1241-1246, 1993

[17] F. Guely, R. La, and P. Siarry, “Fuzzy Rule Base Learning Through Simulated Annealing,” *Fuzzy Sets and Systems*, 105, 353-363, 1999

[18] H.B. Gurocak, “A Genetic-Algorithm-Based Method for Tuning Fuzzy Logic Controllers,” *Fuzzy Sets and Systems*, 108, 39-47, 1999

[19] I. Hayashi, H. Nomura, and N. Wakami, “Acquisition of Inference Rules by Neural Network Driven Fuzzy Reasoning,” *Japanese Journal of Fuzzy Theory and Systems*, 2(**4**), 453-469, 1990

[20] S. Haykin, *Neural Networks*, Macmillan/IEEE Press, 1994

[21] M. Held, P. Wolfe, and, H. P. Crowder, “Validation of Subgradient Optimization”, *Mathematical Programming*, 6, 62-88, 1974

[22] A. Hofbauer, and M. Heiss, “Divergence Effects For Online Adaptation of Membership Functions," *Intelligent Automation and Soft Computing*, 4(**1**), 39-52, 1998

[23] A. Homaifar, and E. McCormick, “Simultaneous Design of Membership Functions and Rule Sets for Fuzzy Controllers Using Genetic Algorithms,” *IEEE Transactions on Fuzzy Systems*, 3(**2**), May 1995

[24] S. Horikawa, T. Furuhashi, and Y. Uchikawa, “Composition Methods and Learning Algorithms of Fuzzy Neural Networks,” *Japanese Journal of Fuzzy Theory and Systems*, 4(**5**), 529-556, 1992

[25] S.H. Jacobson, and L.W. Schruben, “Techniques for Simulation Response Optimization,” *Operations Research Letters*, 8, 1-9, February 1989

[26] J.S.R. Jang, “ANFIS: Adaptive-Network-Based Fuzzy Inference Systems,” *IEEE Transactions on Systems, Man, and Cybernetics*, 23(**3**), 665-685, May-June 1993

[27] J.S.R. Jang, and C.T. Sun, “Neuro-Fuzzy Modeling and Control,” *Proceedings of the IEEE*, 83(**3**), 378-406, March 1995

[28] J.S.R. Jang, C.T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing*, Prentice Hall, Upper Saddle River, NJ, 1997

[29] S.S. Joshi, H.D. Sherali, and J.D. Tew, "An Enhanced Response Surface Methodology (RSM) Algorithm Using Gradient-Deflection and Second-Order Search Strategies", *Computers and Operations Research*, 25(7/8), 531-541, 1998

[30] C.C. Jou, "Supervised Learning in Fuzzy Systems: Algorithms and Computational Capabilities," *Proceedings of the second IEEE Conference on Fuzzy Systems*, 1-6, 1993

[31] R. Katayama, Y. Kajitani, and Y. Nishida, "A Self Generating and Tuning Method for Fuzzy Modeling Using Interior Penalty Method and Its Application to Knowledge Acquisition of Fuzzy Controller," *Fuzzy Control Systems*, A. Kandel, and G. Langholz, Editors, 198-224, CRC Press, Boca Raton, FL, 1994

[32] B. Kosko, "Fuzzy Systems as Universal Approximators," *Proceedings of the first IEEE Conference on Fuzzy Systems*, 1153-1162, 1992

[33] B. Kosko, and J.A. Dickerson, "Function Approximation with Additive Fuzzy Systems," *Theoretical Aspects of Fuzzy Control*, H.T. Nguyen, M. Sugeno, R. Tong, and R. Yager, Editors, Chapter 12, 313-347, John Wiley and Sons, New York, NY, 1994

[34] P.M. Larsen, "Industrial Applications of Fuzzy Logic Control," *International Journal of Man, machine Studies*, 12(**1**), 3-10, 1980.

[35] C.C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller – Part I and II," *IEEE Transactions on Systems, Man, and Cybernetics*, 20(**2**), 404-435, March-April 1990

[36] A.V. Levy, and A. Montalvo, "The Tunneling Algorithm for the Global Minimization of Functions," *SIAM Journal of Scientific and Statistical Computing,* 6(**1**), January 1985

[37] C.T. Lin, and C.S.G. Lee, "Neural-Network-Based Fuzzy Logic Control and Decision System," *IEEE Transactions on Computers*, 40(**12**), 1320-1336, December 1991

[38] D.G. Luenberger, *Introduction to Linear and Nonlinear Programming*, 2nd edition, Addison-Wesley, Reading, MA, 1984

[39] J. Lukasiewicz, Selected Works, L. Borkowski Ed., North Holland, London, 1970

[40] E.H. Mamdani, "Applications of fuzzy algorithms for simple dynamic plant," *Proceedings of IEE*, 121, 1585-1588, 1974.

[41] E.H. Mamdani, and S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *International Journal of Man-Machine Studies*, 7, 1-13, 1975

[42] D. McNeill and P. Freiberger, Fuzzy Logic, Simon & Schuster, New York, NY, 1993

[43] J.M. Mendel, "Fuzzy Logic Systems for Engineering: A Tutorial," *Proceedings of IEEE*, 83(**3**), 1995

[44] B.B. Meunier, M. Dotoli, and B. Maione, "On the Choice of membership Functions in a Mamdani-Type Fuzzy Controller," *1st Online World Congress on Soft Computing*

[45] S. Mitaim, and B. Kosko, "What is the Best Shape for a Fuzzy Set in Function Approximation?," *Proceedings of the 5th IEEE Conference on Fuzzy Systems*, 1237-1243, 1996

[46] H. Miyata, M. Ohki, and M. Ohkita, "Self-Tuning of Fuzzy Reasoning by the Steepest Descent Method and Its Application to a Parallel Parking," *IEICE Transactions on Information and Systems*, E79-D(**5**), 561-569, May 1996

[47] K.S. Narendra, and K. Parthasarathy, "Identification and Control of a Dynamical System using Neural Networks," *IEEE Transactions on Neural Networks*, 1(**1**), 4-27, 1990

[48] R. Narducci, B. Grossman, M. Valorani, A. Dadone, and R.T. Haftka, "Optimization methods for non-smooth or noisy objective functions in fluid design problems," *Proceedings of the 12th AIAA Computational Fluid Dynamics Conference*, 1, 21-32, AIAA paper 95-1648-CP, San Diego, CA, June 19-22, 1995

[49] D. Nauck, and R. Kruse, "Neuro-Fuzzy Systems for Function Approximation," *Fuzzy Sets and Systems*, 101, 261-271, 1999

[50] J.A. Nelder, and R. Mead, "A Simplex method for Function Minimization," *Computer Journal*, 7, 308-313, 1964

[51] H. Nomura, I. Hayashi, and N. Wakami, "A Learning Method of Fuzzy Inference Rules by Descent Method," *Proceedings of the first IEEE Conference on Fuzzy Systems*, 203-210, 1992

[52] H. Nomura, I. Hayashi, and N. Wakami, "Self-Tuning Fuzzy Reasoning By Delta Rule and Its Application to Obstacle Avoidance," *Japanese Journal of Fuzzy Theory and Systems*, 4(**2**), 261-272, 1992

[53] H. Nomura, I. Hayashi, and N. Wakami, "A Self-Tuning Method of Fuzzy Reasoning by Genetic Algorithm," *Proceedings of the 1992 International Fuzzy Systems and Intelligent Control Conference*, 236-245, 1992

[54] H. Nomura, I. Hayashi, and N. Wakami, "A Self-Tuning Method of Fuzzy Reasoning by Genetic Algorithm," *Fuzzy Control Systems*, A. Kandel, and G. Langholz, Editors, 338-354, CRC Press, Boca Raton, FL, 1994

[55] K. Nozaki, H. Ishibuchi, and H. Tanaka, "A simple but Powerful Heuristic Method for Generating Fuzzy Rules from Numerical Data," *Fuzzy Sets and Systems*, 86, 251-270, 1997

[56] M. Nyberg, and Y.H. Pao, "Automatic Optimal Design of Fuzzy Systems based on Universal Approximation and Evolutionary Programming," *Fuzzy Logic and Intelligent Systems*, H. Li, and M. Gupta, Editors, Chapter 12, 311-363, Kluwer

[57] M. Ohki, H. Miyata, M. Tanaka, and M. Ohkita, “A Self-Tuning of Fuzzy Reasoning by Modifying Learning Coefficients,” *Nonlinear Analysis, Theory, Methods & Applications: Proceedings of the 2nd World Congress of Nonlinear Analysts*, 30(**8**), 5291-5301, 1997

[58] J.V. de Oliveira, “Towards Neuro-Linguistic Modeling: Constraints for Optimization of Membership Functions,” *Fuzzy Sets and Systems*, 106, 357-380, 1999

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

[60] W. Pedrycz, “Why Triangular membership Functions?,” *Fuzzy Sets and Systems*, 64, 21-30, 1994

[61] C.S. Peirce, Collected Papers of Charles Sanders Peirce, C. Hartshorne and P. Weiss Eds., 5-6, Harvard University Press, Cambridge, MA, 1934-1935

[62] B.T. Poljak, “A General Method for Solving Extremum Problems,” *Soviet Mathematics*, 8, 593-597, 1967

[63] B.T. Poljak, “Minimization of Unsmooth Functionals,” *U.S.S.R. Computational Mathematics and Mathematical Physics*, 9, 14-29, (English translation), 1969

[64] A. Putz, and R. Weber, “Automatic Adjustments of Weights in Fuzzy Rule Bases,” *Proceedings of EUFIT’96*, 933-938, September 2-5, 1996

[65] N.K.A. Rashid, and A.S. Heger, “Tuning of Fuzzy Logic Controllers by Parameter Estimation Method,” *Fuzzy Logic and Control Software and Hardware Applications*, M. Jamshidi, N. Vadiee, and T. Ross, Editors, Chapter 18, 374-392, PTR Prentice Hall, Englewood Cliffs, NJ, 1993

[66] H. Robbins, and S. Monro, “A Stochastic Approximation Method,” *Annals of Mathematical Statistics*, 3, 400-407, September 1951

[67] B. Russell, “Vagueness,” *Australian Journal of Psychology and Philosophy*, 1, 84-92, 1923

[68] H. D. Sherali and, O. Ulular, “A Primal-Dual Conjugate Subgradient Algorithm for Specially Structured Linear and Convex Programming Problems”, *Applied Mathematics and Optimization*, 20, 193-221, 1989

[69] H.D. Sherali, and C.H. Tuncbilek, “A Global Optimization Algorithm for Polynomial Programming Problems Using a Reformulation-Linearization Technique,” *Journal of Global Optimization*, 2, 101-112, 1992

[70] H.D. Sherali, and C.H. Tuncbilek, “A Reformulation-Convexification Approach for Solving Nonconvex Quadratic Programming Problems,” *Journal of Global Optimization,* 7, 1-31, 1995

[71] H.D. Sherali, and W.P. Adams, *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*, Kluwer Academic, 1999

[72] H.D. Sherali, G. Choi, and Z. Ansari, “Limited Memory Space Dilation and Reduction Algorithms,” *Computational Optimization and Applications*, 19, 55-77, 2001

[73] H.D. Sherali, and H. Wang, “Global Optimization of Nonconvex Factorable Programming Problems,” *Mathematical Programming,* 89(**3**), 459-478, 2001

[74] Y. Shi, and M. Mizumoto, “Some Considerations on Conventional Neuro-Fuzzy Learning Algorithms by Gradient Descent Method,” *Fuzzy Sets and Systems*, 112, 51-63, 2000

[75] Y. Shi, and M. Mizumoto, “A new Approach of Neuro-Fuzzy Learning Algorithm for Tuning Fuzzy Rules,” *Fuzzy Sets and Systems*, 112, 99-116, 2000

[76] B.G. Song, R.J. Marks, S. Oh, P. Arabshahi, T.P. Caudell, and J.J. Choi, “Adaptive Membership Function Fusion and Annihilation in Fuzzy If-Then Rules,” *Proceedings of the second IEEE Conference on Fuzzy Systems*, 961-967, 1993

[77] T. Takagi, and M. Sugeno, "Fuzzy identification of Systems and its Applications to Modeling and Control," *IEEE Transactions on Systems, Man, and Cybernetics*, 15(**1**), January-February 1985.

[78] D. Teodorovic, and K. Vukadinovic, *Traffic Control and Transport Planning*, Kluwer Academic, 1999

[79] L.N. Teow, and K.F. Loe, "Effective Learning in Recurrent Max-Min Neural Networks," *Neural Networks*, 11, 535-547, 1998

[80] D. Wang, T. Chai, and L. Xia, "Adaptive Fuzzy Neural Controller Design," *Proceedings of the American Control Conference*, 4258-4262, Seattle, WA, June 1995

[81] L.X. Wang, and J.M. Mendel, "Fuzzy Basis Functions, Universal Approximation, and Orthogonal Least-Squares Learning," *IEEE Transactions on Neural Networks*, 3(**5**), 807-813, September 1992

[82] L.X. Wang, "Fuzzy Systems as Nonlinear Dynamic System identifiers Part I: Design," *Proceedings of the 31st Conference on Decision and Control*, 897-902, Tucson, AZ, 1992

[83] L.X. Wang, and J.M. Mendel, "Back-Propagation Fuzzy System as Nonlinear Dynamic System Identifiers," *Proceedings of the first IEEE Conference on Fuzzy Systems*, 1409-1418, 1992

[84] L.X. Wang, "Fuzzy Systems are Universal Approximators," *Proceedings of the first IEEE Conference on Fuzzy Systems*, 1163-1170, 1992

[85] L.X. Wang, *Adaptive Fuzzy Systems and Control: Design and Stability Analysis*, PTR Prentice Hall Inc., Englewood Cliffs, NJ, 1994

[86] L.X. Wang, "Design and Analysis of Fuzzy Identifiers of Nonlinear Dynamic Systems," *IEEE Transactions on Automatic Control*, 40(**1**), 11-23, January 1995

[87] Y. Yao, “Dynamic Tunneling Algorithm for Global Optimization,” *IEEE Transactions on Systems, Man, and Cybernetics*, 19(**5**), 1222-1230, September-October 1989

[88] L.A. Zadeh, “Fuzzy Sets,” *Information and Control*, 8, 338-352, 1965

[89] L. A. Zadeh, “Outline of a New Approach to the Analysis of Complex Systems and Decision Processes,” *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(**1**), 28-44, January 1973.

[90] J. Zhang, and A. Knoll, “Designing Fuzzy Controllers by Rapid Learning,” *Fuzzy Sets and Systems*, 101, 287-301, 1999

[91] “The future is fuzzy,” *Newsweek*, May 1990

Vita

Paolo Dadone was born in Torino, Italy. He received the “laurea” degree (5 years degree) summa cum laude in Electronics Engineering in 1995 from Politecnico di Bari, Bari, Italy. In the Fall of 1995 he was a visiting scholar at Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, were he performed research on scheduling in flexible manufacturing systems, which led to his “laurea” thesis. In January 1996 he was accepted as a graduate student in the Electrical Engineering department at Virginia Polytechnic Institute and State University and received the M.S. degree in Electrical Engineering in 1997. He remained enrolled as a graduate student from January 1998 to May 2001, when he received the Ph.D. degree in Electrical Engineering from Virginia Polytechnic Institute and State University. In the summer of 1999 he was an intern in the Control Systems group at General Electric Corporate Research and Development, Schenectady, NY, USA. In June 2001 he will begin working as an Associate in the Systems and Strategies group of Constellation Power Source, Baltimore, MD, USA.

Dr. Dadone’s research interests are in mathematical and simulation modeling, development and implementation of control and optimization algorithms, soft computing, discrete-event dynamical systems, and scheduling of manufacturing systems. Dr. Dadone is a member of several honor societies as well as IEEE technical societies. He was the recipient of the 1996 IEEE VMS graduate student paper contest and of the 1996 Politecnico di Bari fellowship for studying abroad.

Publications

Work In Progress

P. Dadone, W. Lacarbonara, A.H. Nayfeh, and H.F. VanLandingham, "Payload Pendulation Reduction Using a VGT Architecture with LQR and Fuzzy Controls," submitted to *Journal of Vibration and Control*, (under final revision)

P. Dadone, and H.F. VanLandingham, "Multi-Objective Evolutionary Design for Fuzzy Logic Based Scheduling," submitted to *International Journal of Engineering Applications of Artificial Intelligence*, (under final revision)

P. Dadone, and H.F. VanLandingham, "On the non-Differentiability of Fuzzy Logic Systems," to be submitted to *Transaction on Fuzzy Systems*

P. Dadone, and H.F. VanLandingham, "Supervised Learning of Fuzzy Systems with Learning Rate Adaptation by Limited Memory Quadratic Fit," to be submitted to *Transactions on Systems, Man, and Cybernetics*

P. Dadone, and H.F. VanLandingham, "Supervised Learning of Fuzzy Systems with Limited Memory Quadratic Fit and Gradient Deflection," to be submitted to *Fuzzy Sets and Systems*

P. Dadone, and H.F. VanLandingham, "Current Approaches and Perspectives in the Supervised Learning of Fuzzy Logic Systems," to be submitted

Published

P. Dadone, H.F. VanLandingham, and B. Maione, "Modeling and Control of Discrete-Event Dynamic Systems: a Simulator-Based Reinforcement-Learning Paradigm," *International Journal of Intelligent Control and Systems*, 2(4), 609-631, 1998

P. Dadone, and H.F. VanLandingham, "Load Transfer Control for a Gantry Crane with Large Delay Constraints," *Nonlinear Vibration Conference*, Blacksburg, VA, July 23-27, 2000, and accepted *Journal of Vibration and Control*, 2001

P. Dadone, and H.F. VanLandingham, "Remote Control of Industrial Processes," *Proceedings of SMCia/01, 2001 IEEE Workshop on Soft Computing Methods in Industrial Applications*, Blacksburg, VA, USA, June 25–27, 2001

P. Dadone, and H.F. VanLandingham, "Non-differentiable Optimization of Fuzzy Logic Systems," in *Intelligent Engineering Systems through Artificial Neural Networks*, (C.H Dagli, A.L. Buczak, J. Ghosh, M.J. Embrechts, O. Ersoy, and S. Kercel editors), 10, 349-354, ASME Press, NY, 2000

F. Azam, P. Dadone, and H.F. VanLandingham, "Derivative Constrained Function

Approximation," in *Intelligent Engineering Systems through Artificial Neural Networks*, (C.H Dagli, A.L. Buczak, J. Ghosh, M.J. Embrechts, O. Ersoy, and S. Kercel editors), 10, 63-68, ASME Press, NY, 2000

P. Dadone, and H.F. VanLandingham, "On the non-Differentiability of Fuzzy Logic Systems," *Proceedings of IEEE Conference on Systems, Man and Cybernetics 2000*, 2703-2708, Nashville, TN, October 8-11, 2000

P. Dadone and H.F. VanLandingham, "The Use of Fuzzy Logic for Controlling Coulomb Friction in Crane Swing Alleviation," in *Intelligent Engineering Systems through Artificial Neural Networks*, (C.H Dagli, A.L. Buczak, J. Ghosh, M.J. Embrechts, and O. Ersoy editors), 9, 751-756, ASME Press, NY, 1999

P. Dadone, and H.F. VanLandingham, "Adaptive Online Parameter Tuning Using Genetic Algorithms," *4th Online World Conference on Soft Computing in Industrial Applications* (Invited paper), 21-30 September 1999

P. Dadone, and H.F. VanLandingham, "Genetic Algorithms for Tuning PLC Loops," *Proceedings of SMCia/99, 1999 IEEE Workshop on Soft Computing Methods in Industrial Applications*, Kuusamo, Finland, June 16–18, 1999

P. Dadone and H.F. VanLandingham, "Fuzzy Control of Flexible Manufacturing Systems," *Proceedings of ANNIE'98: Smart Engineering System Design*, (C.H Dagli, A.L. Buczak, J. Ghosh, M.J. Embrechts, and O. Ersoy editors), 8, St. Louis, MO, 1998

P. Dadone, and H.F. VanLandingham, "PLC Implementation of a Genetic Algorithm for Controller Optimization," *Proc, of the 3rd Int'l Conf. Computational Intelligence and Neuroscience*, Raleigh, NC, October 1998

P. Dadone, and H.F. VanLandingham, "Fuzzy-Neural Adaptation Through Evolutionary Exploration," *Proc. Int'l Conf. Systems, Signals, Control, Computers* (*SSCC'98*), Durban, South Africa, September 1998

P. Dadone, and H.F. VanLandingham, "Genetic-Based Fuzzy Adaptation," *Proceedings of the 1998 IEEE International Conference of Fuzzy Systems (WCCI'98)*, 1094-1099, Anchorage, AK, May 4-9, 1998

P. Dadone, and H.F. VanLandingham, "Fuzzy Adaptation through Genetic Exploration," *Proceedings of the European Symposium on Intelligent Techniques*, 126-130, Bari, Italy, March 20-21, 1997

P. Dadone, "Fuzzy Control of Flexible Manufacturing Systems," *Master's Thesis*, Virginia Polytechnic Institute and State University, Department of Electrical Engineering, Blacksburg, VA, 1997

P. Dadone, H.F. VanLandingham, and B. Maione, "Fuzzy Techniques for Controlling Flexible Manufacturing Systems," *Proceedings of IASTED Control'97*, 171-174, Cancun, Mexico, May 28-31, 1997

P. Dadone, and H.F. VanLandingham, "Fuzzj Toolbox," *Proceedings of the Seventh Semi-Annual Meeting MURI – Nonlinear Active Control of Dynamical Systems*, Virginia Tech, March 2000

P. Dadone, and H.F. VanLandingham, "Fuzzy Control Toolbox," *Proceedings of the Sixth Semi-Annual Meeting MURI – Nonlinear Active Control of Dynamical Systems*, Virginia Tech,

October, 1999

P. Dadone, and H.F. VanLandingham, "*Visual* Fuzzy Controller Design," *Proceedings of the Fifth Semi-Annual Meeting MURI – Nonlinear Active Control of Dynamical Systems*, Virginia Tech, March 1999

P. Dadone, and H.F. VanLandingham, "Intelligent Control Methods for Ship-Mounted Cranes: the Visual Crane Simulator," *Proceedings of the Fourth Semi-Annual Meeting MURI – Nonlinear Active Control of Dynamical Systems*, Virginia Tech, October 1998

P. Dadone, and H.F. VanLandingham, "Intelligent Control Methods for Ship-Mounted Cranes," *Proceedings of the Third Semi-Annual Meeting MURI – Nonlinear Active Control of Dynamical Systems*, Virginia Tech, April 1998

P. Dadone, and H.F. VanLandingham, "Crane Swing Control Using the Maryland Rigging," *Proceedings of the Second Semi-Annual Meeting MURI – Nonlinear Active Control of Dynamical Systems*, Virginia Tech, August 1997

H.F. VanLandingham, P. Dadone, and M.S. Moon, "Control of Crane Swing Using Fuzzy Control Methods," *Proceedings of the First Semi-Annual Meeting MURI – Nonlinear Active Control of Dynamical Systems*, Virginia Tech, February 1997