KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

Politechnika Wrocławska

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

ROZWÓJ POTENCJAŁU I OFERTY DYDAKTYCZNEJ POLITECHNIKI WROCŁAWSKIEJ

Wrocław University of Technology

Advanced Informatics and Control

A. J. Koshkouei, O. C. L. Haas

THEORY AND PRACTICE OF ARTIFICIAL INTELLIGENCE FOR CONTROL

Artificial Intelligence for Control

Wrocław 2011

Projekt współfinansowany ze środków Unii Europejskiej w ramach
Europejskiego Funduszu Społecznego

Wrocław University of Technology

Advanced Informatics and Control

A. J. Koshkouei, O. C. L. Haas

THEORY AND PRACTICE OF ARTIFICIAL INTELLIGENCE FOR CONTROL

Artificial Intelligence for Control

Wrocław 2011

Reviewer: K. Burnham

ISBN 978-83-62098-36-1

Published by PRINTPAP Łódź, www.printpap.pl

Preface

This book is one of a series of Masters level texts which has been produced for taught modules within a common course designed for Advanced Informatics and Control. The new common course development forms a collaboration between Coventry University, United Kingdom, and Wroclaw University of Technology, Poland. The new course recognises the complexity of new and emerging advanced technologies in informatics and control, and each text is matched to the topics covered in an individual taught module. The source of much of the material contained in each text is derived from lecture notes, which have evolved over the years, combined with illustrative examples, which may well have been used by many other authors of similar texts. Whilst the sources of the material may be many, any errors that may be found are the sole responsibility of the authors.

Intelligent techniques are playing an increasingly important role in engineering and science having evolved from a specialised research subject to mainstream applied research and commercial products. This book focuses on the topics of fuzzy logic, artificial neural networks (ANN) and genetic algorithms (GAs). This book presents a brief theoretical introduction on each subject before describing it using examples dealing with modelling and control. Most of the work is illustrated using the computer software package MATLAB®.

Artificial Neural Networks (ANNs) or Neural Networks (NN) are similar to the biological neural networks in the sense of information processing paradigm as their functions and structures are similar to the biological neural networks consisting of a set of parallel units which operates collectively. Humans normally learn from the past events and gain experience. Such experience and learning is subsequently used to find solutions for future problems of similar complexity. Similar processes are applied to create an ANN. In this regard, ANNs are configured via a learning process by using available data obtained from past experiences and/or facts to form an algorithm to solve a complex problem. The ability of problem solving of ANNs, has been recognised as one of the popular and effective methods which are used in various disciplines including economics, biomedical systems, pattern recognition, data classification, engineering and other subject areas. Advanced computational technologies and software have enhanced the usability of ANNs regardless of the size of data and their complexity. The intention of this book is not to provide various algorithms with mathematical proofs and computational intelligence paradigms for all techniques. However, the most popular techniques and algorithms are presented such that the readers understand them readily and are able to apply these methods for solving relatively complex problems in their subjects. Many applications in informatics and control with associated MATLAB® m-files are provided.

Fuzzy logic theory is based on fuzzy set theory introduced by Zadeh in 1965, and yields an approximate solution for a problem by generating laws from assumptions, facts, logic explanations and other available information. Fuzzy logic has a wide range of applications in many areas including multi-criteria optimisation, medical diagnoses and sciences, transportation, predictions of price markets and engineering. It should be emphasised that a method may benefit from both fuzzy logic and ANN techniques which are able to solve a complex problem.

The applications of genetic algorithms have been implemented using the Genetic Algorithm toolbox developed in Sheffield by Chipperfield, Fleming, Polheim and Fonseca because it is freely available to download and is appropriate for the purpose.

The brief outline of this book is as follows:

Chapter 1 starts with an introduction of the basic concepts of fuzzy logic with application to control engineering prior to moving on to fuzzy operations and the methods of defuzzification. The Last part of the Chapter focuses on fuzzy control design methods including Mamdani and Takagi-Sugeno controllers.

Chapter 2 presents neural networks including some theoretical background supported by exercises and MATLAB® examples. A number of common neural network architectures are described including single and multilayer perceptrons (MLP), Radial basis function as well as Gaussian radial basis functions and generalised regression neural network. Training algorithms based on backpropagation and k-means algorithms are discussed and demonstrated using MATLAB® examples. Finally a worked example is used to highlight the importance f network pruning and regularisation.

Chapter 3 presents genetic algorithm as a method of optimisation and demonstrate its use through MATLAB® examples applied to proportional + integral + derivative controller tuning and system identification.

This book has been written such that concepts are introduced through examples. A bibliography is provided for reader wishing to gain more detailed information on particular aspects of the material presented in this book.

Table of Contents

1 Fuzzy Logic and applications in control

1.1 Brief history

The concept of fuzzy set theory was introduced in 1965 by Lotfi Zadeh (Zadeh, 1965. However, at the beginning, the development of this idea was very slow. In 1972, the first working group on fuzzy systems was established in Japan by Toshiro Terano(Sugeno and, 2005). Zadeh published a paper about fuzzy algorithms in 1973 (Zadeh, 1973). After this date, interest in fuzzy theory started to grow leading to the development of many algorithms (Verbuggem, 1999) together with their application to a wide range of application domain. Early applications included control of a steam engine system(Mamdani, 1974), an expert system for loan applicant evaluation (Hans Zimmermann, 1977), control of a cement kiln system (Smidth et al, 1980), water treatment control system to control chemical injection, controlling of the subway Sendai transportation system (Jantzen, 2007), chess and a backgammon program (Hans Berliner, 1999). Since this date, a number of programs and algorithms capable of beating world-class human players have been developed for both chess and backgammon.

The first fuzzy logic chip was developed by Masaki Togai and Hiroyuke Watanabe at AT & T Bell Laboratories (USA) in 1985 (Hans Zimmermann, 1993). By 1987 many fuzzy logic applications including container crank control, tunnel excavation, soldering robot and automated aircraft vehicle landing, had been introduced.

Togai InfraLogic Inc. was the first fuzzy company established in Irvine (USA), 1987. In 1989 the Laboratory for International Fuzzy Engineering Research (LIFE) in Japan was established. In 1990 the Fuzzy Logic Systems Institute (FLSI), led by Professor Takeshi Yamakawa, developed the fuzzy neuron chip in BiCMOS (bipolar complementary metal oxide semiconductor) technology which facilitates hand-written character recognition within one microsecond using a single fuzzy neuron chip and the chaos chip in CMOS technology in (1991) and (1992), respectively. In 1991, the Intelligent Systems Control Laboratory was created in Siemens (Germany) and as well as the Fuzzy Artificial Intelligent Promotion Centre in Japan. Since 1992 many events, inventions and applications concerning fuzzy logic and its applications have appeared.

Having presented a brief introduction on the origin and some developments in fuzzy logic, the remainder of the chapter is composed as follows. Section 1.2 introduces the concept of fuzzy set illustrated with examples detailing how a fuzzy set can be constructed before describing common fuzzy operations such as ‘min’, ‘max’ and relation are. The primary concepts such as membership function and fuzzy logic inferences and implications are described in Section 1.3. The concept of defuzzification and the methods of defuzzification are presented in Section 1.4. Finally fuzzy controllers are introduced and the differences between the Mamdani and Takagi and Sugeno controllers are illustrated. Throughout Section 1 examples are used to illustrate the principles and laws described.

1.2 Fuzzy Sets and primary concepts

In this section the concept of the fuzzy sets and their application are introduced by using simple examples. Then membership functions and related concepts such as crossover, support and core are defined.

'Clearly, the "class of all real numbers which are much greater than 1,"or "the class of beautiful women," or "the class of tall men," do not constitute classes or sets in the usual mathematical sense of these terms. (Zadeh, 1965])'

Zadeh's Complaint

A crisp set is a well-defined class of objects or elements such that a definitive list may be given. The elements may be defined based on rules. For example the names of some of the days of the week can be defined as a set A,

A= {Sunday, Monday, Tuesday, Friday, Saturday}

where A is arbitrarily chosen to contain five elements. The order of the elements is not important, therefore the set A may be written as

A= {Tuesday, Monday, Sunday, Friday, Saturday}

All the possible days of a week constitutes a set which is termed the universe of discourse, or simply universe or universal set. The universal set defining the days of the weeks is therefore:

U= {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}

All sets are a subset of the universal set. The universal set is the largest set that the associated elements belong to. The universal set depends on the nature of the elements and it includes all elements or objects that are considered.

For example, assuming that B is the set of natural number less than 10:

B= {1, 2, 3, 4, 5, 6, 7, 8, 9}

Then the universal set is the set of all natural numbers.

Assume that C is a set and define a function μ_C on C such that:

$\mu_C(x) = 1$ if x is an element of *C* and

$\mu_C(x) = 0$ if x is not a member of *C*

$$\mu_C(x) = \begin{cases} 1 & \text{if } x \in C \\ 0 & \text{if } x \notin C \end{cases}$$

It follows that μ_C is a function from the universal set U into $\{0,1\}$

$$\mu_C : U \to \{0,1\}$$

and the set C is defined as:

$$C = \{x \in U : \mu_C(x) = 1 \ \}$$

in which μ_C is called the membership function. In this case every element of the universal set is either in C or not in C.
For example, for the set of days of the week, A, $\mu_A(\text{Monday}) = 1$ as it is present in the set A however $\mu_A(\text{Wednesday}) = 0$ because it is not present.

Given the set B, $\mu_B(5) = 1$ and $\mu_B(15) = 0$, the range of the membership function can then be defined as the set $\{0,1\}$.

Now consider the concepts such as 'nearly weekend' or 'number near to 6'. In this case, the membership of the set is increased and the range of the set is not $\{0,1\}$. In fact, the range of the set includes values in the interval [0, 1],

$$\mu_A : U \rightarrow [0,1]$$

where the membership function is within the interval $0 \leq \mu_A \leq 1$, with the values in the interval [0, 1] depending on the definition of the membership function.

Example 1.2.1 (Days of a week): The following are questions about the days of a week and membership values g they could be associated with:

- Is Tuesday a weekend day? The value is 0 because it is a false statement.
- Is Friday a weekend day? It is also a not correct, however, given that it is near the weekend, i.e. the answer is mostly yes. So the value should not be 0, for example it is considered as 0.8.
- Is Saturday a weekend day? It is a true statement, because it is the weekend. So the membership value can be 1.
- Is Sunday a weekend day? It can be considered as a true statement, however, it is also close to Monday which is a working day. By contrast to Saturday, it is not followed by a week end day so and the membership value could be less than one, say 0.95.
- Is Monday, Tuesday or Wednesday a weekend day? They are not and even if Monday is close to the week-end it marks the beginning of the working week, so we are v very far from the next week end. The membership value for each of these elements could be set to 0.
- Is Thursday a weekend day? It is not, however, it is nearing the week end. Given that it is not as close to the weekend as Friday, a membership value of 0.2 could be selected.

Example 1.2.2 (The seasons):

There are four seasons and the weather of each season does normally change from the middle of a season. Therefore, the most typical weather that describes a season is around the middle of the season. The weather then gradually changes from one recognised weather pattern to the next one. The natural definitions of the membership function of the seasons are shown in Figure 1.2.1. Each

season can therefore be represented by a sine wave which peaks at the middle of each season with one the two neighbouring sinewave indicating when the adjacent season peaks.

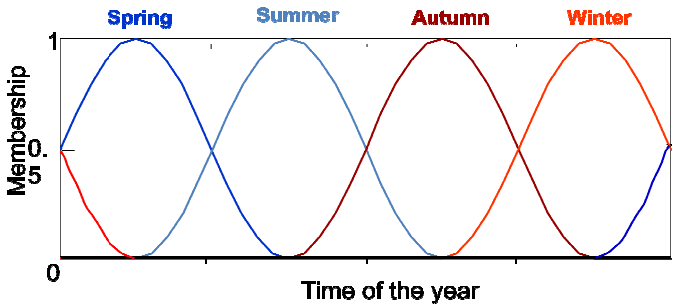


Figure 1.2.1 The memberships of the seasons

Example 1.2.3 (The age: old and young definitions):

It is standard to classify the population according to its age. Let's assume that a population could be separated into the following groups: very young, young, middle age, not old, old or very old. It is assumed that the maximum life of a particular population is 100 years. Based on the definition of each of the aforementioned group a membership function may be defined. It is assumed that a 50 year old person is middle age, a new born baby is very young and a 100 year old person is very old. Based on these assumptions, the membership functions for the very young, young, not too old, not too young, middle age, old and very old variables are defined, see Figure 1.2.1.The shape of the membership functions were selected to characterise how the age of a person in a population is perceived. For example, 'middle age' would have a fairly wide base whereas very young or very old would be marked by sharper thresholds.

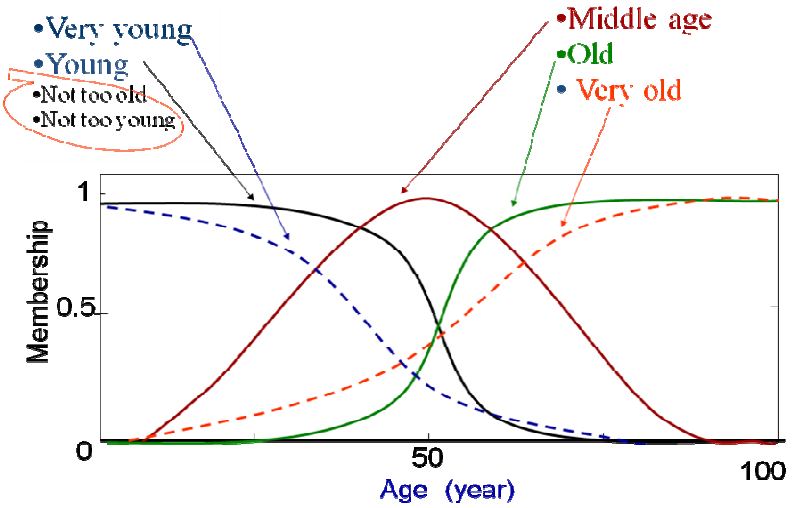


Figure 1.2.2: The membership functions of age.

Example 1.2.4 (Temperature or pressure): The temperature or pressure of a control system is a continuous function of time. The maximum and minimum temperature or pressure depends on the system specifications. The pressure levels can be considered to be low, normal or high. The pressure level thresholds may change with the system considered. However, from a fuzzy logic perspective the same generic approach can be adopted. In this example, the shape of the membership function was selected to be triangular to reflect the linear changes between pressure levels see Figure 1.2.3.

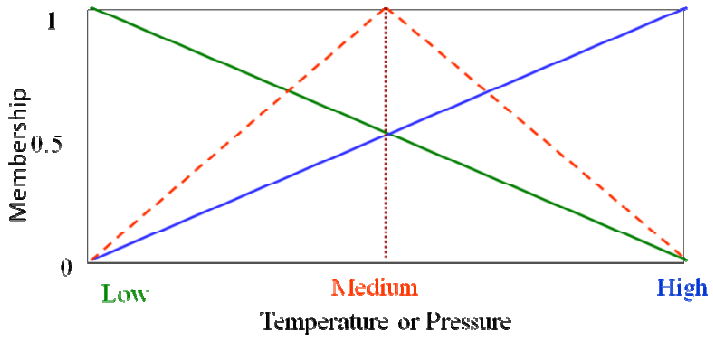


Figure 1.2.3: The membership functions of temperature or pressure of a control system.

1.3 Representation of fuzzy sets

Let U be a universal set and $A \subseteq U$. A fuzzy set denoted $\tilde{A}$ on the universal set U is a set of ordered pairs defines as: $\tilde{A} = \{(x, \mu_A(x): \ x \in U\}$

in which for all $x \in U - A$, $\mu_A(x) = 0$. The domain and range of the membership function are given by U and $[0, 1]$, respectively. It follows that: $\mu_A : U \to [0,1]$ where $0 \le \mu_A \le 1$. For each $x \in U$, $\mu_A(x)$ is called membership degree of x.

1.3.1 Discrete time fuzzy set examples

Example 1.3.1:

Let $U = \{0, 1, 2, 3, 4, 5, 6\}$ be the universal set and $A = U$. Define a fuzzy set on $A = U$ as follows:

$$\tilde{A} = \{(0,0.1),(1,0.3),(2,0.4),(3,1),(4,0.7),(5,0.5),(6,0)\}$$

in which $\mu_A(0) = 0.1$, $\mu_A(1) = 0.3$, $\mu_A(2) = 0.4$, $\mu_A(3) = 1$, $\mu_A(4) = 0.7$, $\mu_A(5) = 0.5$ and $\mu_A(6) = 0.$

The membership function can be written as

x	0	1	2	3	4	5	6
$\mu_{\tilde{A}}(x)$	0.1	0.3	0.4	1	0.7	0.5	0

Zadeh used the following notation to represent the above membership function:

$$\tilde{A} = \left\{ \frac{0.1}{0} + \frac{0.3}{1} + \frac{0.4}{2} + \frac{1}{3} + \frac{0.7}{4} + \frac{0.5}{5} + \frac{0}{6} \right\}$$

Facts: Let ϕ and $\overline{A}$ be the null set and the complement of the set A, respectively, then

- For all $x \in U, \quad \mu_{\phi}(x) = 0$
- For all $x \in U, \quad \mu_{U}(x) = 1$
- If $X \subseteq Y$ then $\mu_X(x) \leq \mu_Y(x)$
- For all $x \in U, \quad \mu_{\overline{A}}(x) = 1 - \mu_A(x)$

Example 2.3.2:

Let $U = \{-2, -1, 0, 1, 2, 3, 4\}$ be the universal set, $X = \{-1, 0, 1\}$ and $Y = \{-1, 0, 2\}$.

Define the two following sets corresponding to the crisp sets X and Y as

$$\tilde{X} = \left\{ \frac{0}{-2} + \frac{0.3}{-1} + \frac{0.5}{0} + \frac{0.7}{1} + \frac{0}{2} + \frac{0}{3} + \frac{0}{4} \right\}$$

$$\tilde{Y} = \left\{ \frac{0}{-2} + \frac{0.1}{-1} + \frac{0.6}{0} + \frac{0}{1} + \frac{1}{2} + \frac{0}{3} + \frac{0}{4} \right\}$$

These two fuzzy sets may be written as

$$\tilde{X} = \left\{ \frac{0.3}{-1} + \frac{0.5}{0} + \frac{0.7}{1} \right\}$$

$$\tilde{Y} = \left\{ \frac{0.1}{-1} + \frac{0.6}{0} + \frac{1}{2} \right\}$$

Note that the elements with 0 membership degree may not be included. In fact all elements of the universal sets which do not appear in the representation have 0 membership degree.

1.3.2 Operations $\cup$ and $\cap$

There are two common definitions for $\cap$ and $\cup$. Let A and B be two crisp sets, and, $\mu_A(x)$ and $\mu_B(x)$ are their membership degree, respectively. Then the membership degree on the set $A \cap B$ is defined as

$$\mu_{A\cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}$$

$$\mu_{A\cap B}(x) = \mu_A(x)\mu_B(x)$$

and the membership degree on the set $A \cup B$ is defined as

$$\mu_{A\cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}$$

$$\mu_{A\cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)$$

In fuzzy logic, the definitions for $\cap$ and $\cup$ using the min and max operations are normally used and are therefore adopted for this book.

There are two common definitions for $\cap$ and $\cup$. Let A and B be two crisp sets, and, $\mu_A(x)$ and $\mu_B(x)$ are their membership degree, respectively. Then

$$\tilde{A} = \left\{\frac{0}{-2} + \frac{0.3}{-1} + \frac{0.5}{0} + \frac{0.7}{1} + \frac{0}{2} + \frac{0}{3} + \frac{0}{4}\right\}$$
$$\tilde{B} = \left\{\frac{0}{-2} + \frac{0.2}{-1} + \frac{0.6}{0} + \frac{0}{1} + \frac{0}{2} + \frac{0.9}{3} + \frac{1}{4}\right\}$$

Example 2.4.1:

Consider the universal set $U = \{-2,\ -1,\ 0,\ 1,\ 2,\ 3,\ 4\}$ and the two subsets A = {-1, 0, 1} and B = $\{-1,\ 0,\ 3,\ 4\}$. Define the fuzzy sets on A and B as follows:

$$\widetilde{A} = \left\{\frac{0}{-2} + \frac{0.3}{-1} + \frac{0.5}{0} + \frac{0.7}{1} + \frac{0}{2} + \frac{0}{3} + \frac{0}{4}\right\}$$
$$\widetilde{B} = \left\{\frac{0}{-2} + \frac{0.2}{-1} + \frac{0.6}{0} + \frac{0}{1} + \frac{0}{2} + \frac{0.9}{3} + \frac{1}{4}\right\}$$

The membership degrees of $\tilde{A} \cup \tilde{B}$ ($\tilde{A} \cap \tilde{B}$) is given by the maximum (minimum) between the membership degree $\tilde{A}$ and $\tilde{B}$.

$$\tilde{A} \cup \tilde{B} = \left\{\frac{0.3}{-1} + \frac{0.6}{0} + \frac{0.7}{1} + \frac{0.9}{3} + \frac{1}{4}\right\},\ \tilde{A} \cap \tilde{B} = \left\{\frac{0.2}{-1} + \frac{0.5}{0}\right\}$$

$\tilde{A} \cup \tilde{B}$ is obtained by taking the maximum degree of membership between $\tilde{A}$ and $\tilde{B}$ for each elements of the universal set. The elements of the universal set are represented by the denominators of $\tilde{A} \cup \tilde{B}$ whereas the degree of membership is in the numerator. The membership degree of all the elements that do not belong to $A \cup B$ is zero. For example -2 and 2 do not belong to either A or B so their membership is zero. Note that when the membership degree is zero, then it is

not necessary to include it in the set $\tilde{A} \cup \tilde{B}$. This is equivalent to saying that the membership degree of elements not present in the set is zero.

Similarly $\tilde{A} \cap \tilde{B}$ is obtained by taking the minimum between the sets $\tilde{A}$ and $\tilde{B}$. Most elements are zero and hence do not need to be included in the resulting set.

Example 2.4: 2:

Let $U = \{-2,\ -1,\ 0,\ 1,\ 2,\ 3,\ 4\}$ be the universal set and

$$\tilde{A} = \left\{\frac{0}{-2} + \frac{0.3}{-1} + \frac{0.5}{0} + \frac{0.7}{1} + \frac{0}{2} + \frac{0}{3} + \frac{0}{4}\right\},$$
$$\tilde{B} = \left\{\frac{0}{-2} + \frac{0.2}{-1} + \frac{0.6}{0} + \frac{0}{1} + \frac{0}{2} + \frac{0.9}{3} + \frac{1}{4}\right\},$$
$$\tilde{C} = \left\{\frac{0.4}{-2} + \frac{0}{-1} + \frac{0.8}{0} + \frac{0}{1} + \frac{1}{2} + \frac{0}{3} + \frac{0}{4}\right\}$$

be the fuzzy sets on the crisp sets

$$\text{A} = \{\text{-1, 0, 1}\},\ \ \text{B} = \{-1,\ 0,\ 3,\ 4\},\ C = \{-2,\ 0,\ 2\},$$

respectively. Then $\tilde{A} \cup (\tilde{B} \cap \tilde{C})$ is given by the minimum between $\tilde{B}$ and $\tilde{\text{C}}$ followed by the maximum between the result and $\tilde{A}$. Similarly $(\tilde{A} \cup \tilde{B}) \cap \tilde{C}$ is given by the maximum between $\tilde{A}$ and $\tilde{B}$ followed by the minimum between the result and $\tilde{\text{C}}$

$$\tilde{A} \cup (\tilde{B} \cap \tilde{C}) = \left\{\frac{0.3}{-1} + \frac{0.6}{0} + \frac{0.7}{1}\right\}$$
$$(\tilde{A} \cup \tilde{B}) \cap \tilde{C} = \left\{\frac{0.6}{0}\right\}$$

Note that operations $\cup$ and $\cap$ are binary. In addition, the operation laws for crisp sets may not be valid for fuzzy operations including $\cup$ and $\cap$ operations.

Now the distribution and associative laws are examined for the above fuzzy sets. For the above example, the complement of A is defined as $\overline{A} = U - A = \{-2,2,3,4\}$ and since the fuzzy membership degree is defined as $\mu_{\overline{A}}(x) = 1 - \mu_A(x)$, therefore,

$$\tilde{\overline{A}} = \left\{\frac{1}{-2} + \frac{0.7}{-1} + \frac{0.5}{0} + \frac{0.3}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4}\right\}$$

On the other hand, according to the definition of operations $\cup$ and $\cap$??????????????????, the following is obtained

$$\tilde{\bar{A}} \cup \tilde{A} = \left\{ \frac{1}{-2} + \frac{0.7}{-1} + \frac{0.5}{0} + \frac{0.7}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \right\}$$

$$\tilde{\bar{A}} \cap \tilde{A} = \left\{ \frac{0}{-2} + \frac{0.3}{-1} + \frac{0.5}{0} + \frac{0.3}{1} + \frac{0}{2} + \frac{0}{3} + \frac{0}{4} \right\}$$

Note that according to standard set theory the following result would be obtained
$A \cap \bar{A} = \phi, \quad A \cup \bar{A} = U$

This means that the standard rules of set theory are not valid for fuzzy sets.

1.3.3 Fuzzy sets and operations for continuous case

Let $X \subseteq U$ where the universal set U is a subset of the real numbers with uncountable elements. For example the universal set should be the union of intervals as a subset of real numbers. Then the fuzzy set on X is defined as

$$\tilde{X} = \left\{ \left(x,\ \mu_{\tilde{X}}(x) \right) \ :\ x \in X \right\} = \left\{ \int \frac{\mu_{\tilde{X}}(x)}{x} \ :\ x \in X \right\}$$

Example 2.5.1:

Assume that $X = \{ x :\ \ 0 \le x \le 12 \}$ and a Gaussian membership function illustrated in Figure 1.3.1 is adopted then the fuzzy set on X is given by:

$$\tilde{X} = \left\{ \left(x,\ \mu_{\tilde{x}}(x) \right) :\ \mathrm{x} \in X \ \text{ and } \mu_{\tilde{x}}(x) = e^{-\left(\frac{x-6}{2}\right)^2} \right\}$$

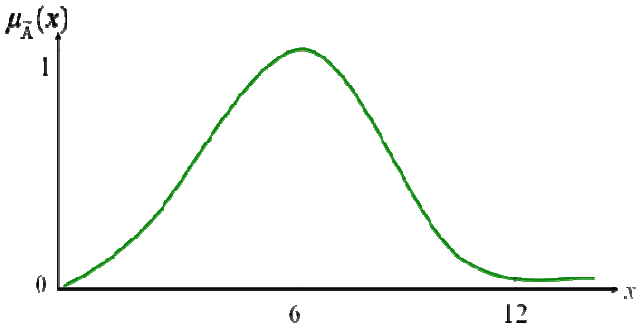


Figure 1.3.1: Gaussian membership function.

1.3.4 Operations ∪ and ∩ for the continuous case

Example 2.5.2:

Let $A=\{x:\ \ 0\le x\le 12\}$ and $B=\{y:\ \ 0\le y\le 6\}$.

Define the fuzzy sets arbitrarily selected as Gaussian membership functions:

$$\tilde{A}=\left\{\ \left(x,\ \mu_{\tilde{A}}(x)\right)\ :\ \mathrm{x}\in A\ \ \text{and}\ \mu_{\tilde{A}}(x)=e^{-\left(\frac{x-6}{2}\right)^2}\right\}$$

$$\tilde{B}=\left\{\ \left(y,\ \mu_{\tilde{B}}(y)\right):\ \mathrm{y}\in B\ \ \text{and}\ \mu_{\tilde{B}}(x)=e^{-\left(\frac{y}{2}\right)^2}\right\}$$

Using the fuzzy sets $\tilde{A}$ and $\tilde{B}$, the membership degrees of the new fuzzy sets $\tilde{C}$ and $\tilde{D}$ are defined as

$$\mu_{\tilde{C}}(x)=\mu_{A\cup B}(x)=\max\left\{\mu_{\tilde{A}}(x),\mu_{\tilde{B}}(x)\right\}$$

$$\mu_{\tilde{D}}(x)=\mu_{A\cap B}(x)=\min\left\{\mu_{\tilde{A}}(x),\mu_{\tilde{B}}(x)\right\}$$

respectively, and their traces are shown in Figure 1.3.2.

The results of the above rules (or operations), i.e. $\mu_{\tilde{C}}(x)$ and ($\mu_{\tilde{D}}(x)$, are also called the firing strengths of the rules.

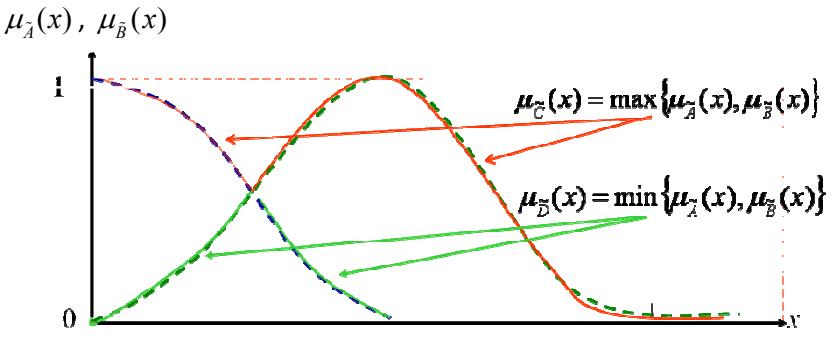


Figure 1.3.2: The fuzzy sets $\tilde{A}$ and $\tilde{B}$ and using max and min operations.

1.3.5 Classical and fuzzy relations

Let X and Y be two sets. Then the classical relation from X into Y is defined as follows:

$$X \times Y = \{ (x, y) : x \in X,\ y \in Y \}$$

Example 2.6.1:

Assume that $X = \{ 1,\ 2,\ 3,\ 4 \}$ and $Y = \{ 1,\ 2,\ 3 \}$. Then

$$X \times Y = \{(1,1),\ (1,2),\ (1,\ 3),\ (2,1),\ (2,2),\ (2,\ 3),\ (3,1),\ (3,2),\ (3,3),\ (4,1),\ (4,2),\ (4,3)\}$$

Now define a fuzzy membership degree on the relation $X \times Y$ using the following law

$$\mu_R(x,y) = \begin{cases} 1 & \text{if } |\text{x-y}| = 0 \\ 0.7 & \text{if } |\text{x-y}| = 1 \\ 0.2 & \text{if } |\text{x-y}| = 2 \\ 0 & \text{otherwise} \end{cases}$$

Where $|\text{x-y}|$ denotes the absolute value and represents the distance between x and y. For example taking x =4 within X and y=1 within Y, then $|\text{x-y}| = 3$. Based on the membership function definition $\mu_R(x,y)$ given that $|\text{x-y}|$ is not equal to 0, 1 or 2 then $\mu_R(4,1) = 0$, denoted in the ser R as $\frac{0}{(4,1)}$.

Therefore, the fuzzy relation constructed from the crisp relation $X \times Y$ is

$$R = \left\{ \frac{1}{(1,1)} + \frac{1}{(2,2)} + \frac{1}{(3,3)} + \frac{0.7}{(1,2)} + \frac{0.7}{(2,1)} + \frac{0.7}{(3,2)} + \frac{0.7}{(2,3)} + \frac{0.2}{(1,3)} + \frac{0.2}{(3,1)} + \frac{0.2}{(4,2)} + \frac{0}{(4,1)} + \frac{0.7}{(4,3)} \right\}$$

This fuzzy relation can be shown in the form of an array

$$R =: X \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} Y \\ \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0.7 & 0.2 \\ 0.7 & 1 & 0.7 \\ 0.2 & 0.7 & 1 \\ 0 & 0.2 & 0.7 \end{bmatrix} \end{array}$$

1.3.6 Operations on fuzzy relations

Operations $\cup$ and $\cap$ can also be defined for fuzzy relations. Using the operations $\cup$ or $\cap$, two new relations are obtained by considering the maximum or the minimum of the corresponding elements of the relations R_1 and R_2. Let R_1 and R_2 be two fuzzy relations then the elements of the relation

$R_1 \cap R_2$ are obtained from the minimum of the two entries which are on the same columns and rows. Similarly the elements of $R_1 \cup R_2$ are calculated by taking the maximum of two corresponding entries d. For example, if a_{ij} and b_{ij} are the elements of R_1 and R_2 on the *i*th-row and *j*th-column, respectively then the *i*th-row and *j*th-column of the fuzzy relations $R_1 \cap R_2$ and $R_1 \cup R_2$ are $\min\{a_{ij}, b_{ij}\}$ and $\max\{a_{ij}, b_{ij}\}$, respectively.

Example 2.6.1:

Consider the following fuzzy relations

$$R_1 =: \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{bmatrix} 0.3 & 0.7 & 0.25 \\ 0.7 & 0.45 & 0.4 \\ 0.2 & 0.6 & 1 \\ 1 & 0.2 & 0.9 \end{bmatrix} \end{array}, \qquad R_2 =: \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{bmatrix} 0.55 & 0.4 & 0.75 \\ 0.27 & 0.65 & 1 \\ 0.66 & 0.26 & 0.3 \\ 0.3 & 0.6 & 0.7 \end{bmatrix} \end{array}$$

Then the fuzzy sets obtained using the relations $\cap$ and $\cup$ are

$$R_1 \cap R_2 =: \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{bmatrix} 0.3 & 0.4 & 0.25 \\ 0.27 & 0.45 & 0.4 \\ 0.2 & 0.26 & 0.3 \\ 0.3 & 0.2 & 0.7 \end{bmatrix} \end{array}, \quad R_1 \cup R_2 =: \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{bmatrix} 0.55 & 0.7 & 0.75 \\ 0.7 & 0.65 & 1 \\ 0.66 & 0.6 & 1 \\ 1 & 0.6 & 0.9 \end{bmatrix} \end{array}$$

1.4 The common types of membership functions:

There are many common types of functions which are used as membership functions. This book will describe triangular, trapezoidal and Gaussian membership functions.

Triangular type of membership function:

This type of membership function is generally as shown in Figure 1.4.1. The mathematical equation to describe the membership function as shown in Figure 1.4.1 is:

$$\mu_A(x) = \begin{cases} 0 & x < a_1 \\ \dfrac{x - a_1}{a_2 - a_1} & a_1 \le x < a_2 \\ \dfrac{a_3 - x}{a_3 - a_2} & a_2 \le x < a_3 \\ 0 & x \ge a_3 \end{cases}$$

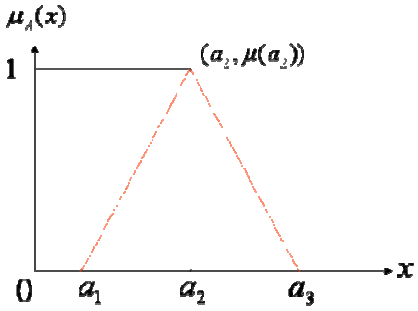


Figure 1.4.1: Triangular fuzzy membership function.

Trapezoidal type of membership function

Błąd! Nie można odnaleźć źródła odwołania. Depicts a trapezoidal membership function and its mathematical expression can be described as follows

$$\mu_A(x) = \begin{cases} 0 & x < a_1 \\ \dfrac{x - a_1}{a_2 - a_1} & a_1 \le x < a_2 \\ 1 & a_2 \le x < a_3 \\ \dfrac{a_4 - x}{a_4 - a_3} & a_3 \le x < a_4 \\ 0 & x \ge a_4 \end{cases}$$

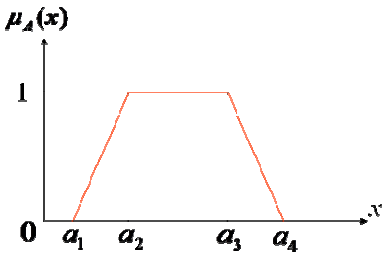


Figure 1.4.2: Trapezoidal fuzzy membership function.

Gaussian shape of membership function

The Gaussian membership function is shown in Figure 1.4.3 and the mathematical relation

$$\tilde{A} = \left\{ \left(x,\ \mu_{\tilde{A}}(x)\right) :\ \mathrm{x} \in X \ \text{ and } \mu_{\tilde{A}}(x) = e^{-\left(\frac{x-\mu}{\sigma}\right)^2} \right\}$$

represents this type of function.

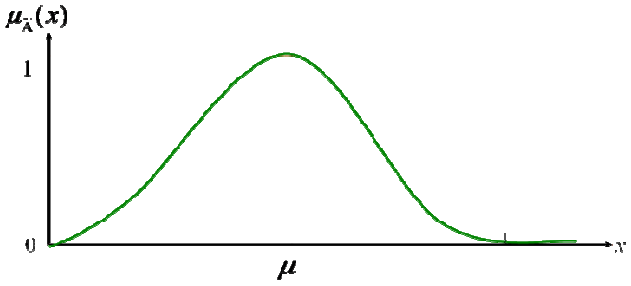


Figure 1.4.3: Gaussian fuzzy membership function.

Triangular membership functions are the simplest and most widely used in practice. Gaussian membership functions are typically used to model large amount of data with typically Gaussian distributions. When there are upper or lower threshold limits that make a variable constant then a trapezoidal membership function is suitable.

1.4.1 Support and boundaries of membership

A normal fuzzy set is one whose membership function has at least one element in the universe whose membership value is unity. The support of a membership function is defined as all values where the membership values are not zero. The core of a membership function is defined as the set of all values which correspond to their membership degree of 1 while the membership degrees of the remainder of the support points are between 0 and 1 as shown in Figure 1.4.4. The crossover points of a membership function are defined as those elements in the universe for which a particular fuzzy set A has membership values equal to 0.5. See Figure 1.4.5.

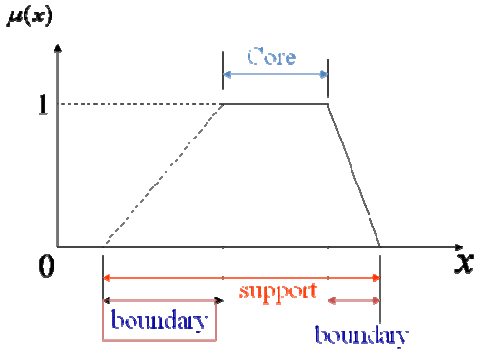


Figure 1.4.4: The boundary, support and core of a membership function.

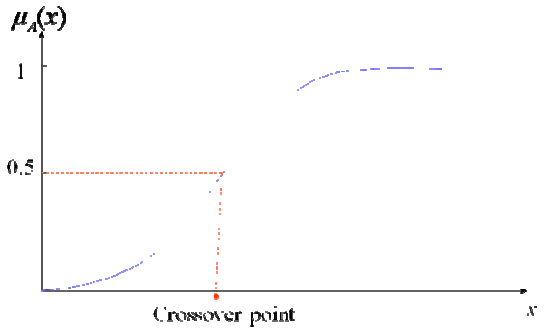


Figure 1.4.5: The crossover point of a membership function.

1.4.2 Convex fuzzy set

The fuzzy set $\widetilde{A}$ is convex if for any element x, y, and z in a set A, the relation $x < y < z$ implies that $\mu_{\tilde{A}}(y) \geq \min\left\{\mu_{\tilde{A}}(x), \mu_{\tilde{A}}(z)\right\}$. Then $\widetilde{A}$ is a convex fuzzy set (Zadeh, 1965). Select $y = \lambda x + (1-\lambda)z$ where $0 \leq \lambda \leq 1$. Then this definition is equivalent to

$$\mu_{\tilde{A}}(\lambda x + (1-\lambda)z) \geq \min\left\{\mu_{\tilde{A}}(x), \mu_{\tilde{A}}(z)\right\}$$

1.4.3 The height of a fuzzy set

The height of a fuzzy set $\tilde{A}$ is the maximum value of the membership degree, i.e. if the height of a fuzzy set is less than unity, the fuzzy set is said to be subnormal. Therefore, if the height of a fuzzy set unity, the fuzzy set is normal. If $\tilde{A}$ is a convex single-point normal fuzzy set (i.e. there is one point whose membership is 1) defined on the real axis, then $\tilde{A}$ is often termed a fuzzy number. Note that a fuzzy number does not refer to one single value. A fuzzy singleton is a fuzzy set, whose support is a single point in the universal set with a membership function of one,

$$\mu_{\tilde{A}}(x) = \begin{cases} 1 & \text{if } x = x_0 \in X \\ 0 & \text{if } x \neq x_0 \in X \end{cases}$$

1.4.4 Projections of a relation

Consider the fuzzy relation R from X to Y. Define the projection of the fuzzy relation R on X

$$\text{proj}(R \text{ on } X) = \sum_x \max_y \frac{\mu_R(x,y)}{x}$$

which is a fuzzy set on X. Similarly, the projection of the fuzzy relation R on Y is defined as

$$\text{proj}(R \text{ on } Y) = \sum_y \max_x \frac{\mu_R(x,y)}{y}$$

which is a fuzzy set on Y.

The projection of R on X is a fuzzy set on X whose membership values are the maxima of the row entries that the corresponding point belongs to. Similarly the projection of the fuzzy relation R on Y is a fuzzy set on Y whose membership function are the minima values of the column entries of that the corresponding point in Y.

Example 2.10.1:

Consider the following fuzzy relations R_1 and R_2 from X to Y

$$R_1 =: \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{bmatrix} 0.3 & 0.7 & 0.25 \\ 0.7 & 0.45 & 0.4 \\ 0.2 & 0.6 & 1 \\ 1 & 0.2 & 0.9 \end{bmatrix} \end{array}, \qquad R_2 =: \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{bmatrix} 0.55 & 0.4 & 0.75 \\ 0.27 & 0.65 & 1 \\ 0.66 & 0.26 & 0.3 \\ 0.3 & 0.6 & 0.7 \end{bmatrix} \end{array}$$

Therefore, the projection of R_1 on X and projection of the fuzzy relation R_1 on Y are the following fuzzy sets:

$$\text{proj}(R_1 \text{ on } X) = \left\{ \frac{0.7}{x_1} + \frac{0.7}{x_2} + \frac{1}{x_3} + \frac{1}{x_4} \right\}$$

This set is obtained by taking the maximum of the first row (0.7) of the relation R_1 and placing it over the x_1 element inside the domain X such that0.7 is the membership degree of x_1 denoted $\frac{0.7}{x_1}$ within the set $\text{proj}(R_1 \text{ on } X)$.

, $$\text{proj}(R_1 \text{ on } Y) = \left\{ \frac{1}{y_1} + \frac{0.7}{y_2} + \frac{1}{y_3} \right\}$$

Exercise 1.4.1:

Find $\text{proj}(R_2 \text{ on } X) = \left\{ \frac{}{x_1} + \frac{}{x_2} + \frac{}{x_3} + \frac{}{x_4} \right\}$ and $\text{proj}(R_2 \text{ on } Y) = \left\{ \frac{}{y_1} + \frac{}{y_2} + \frac{}{y_3} \right\}$.

1.4.5 Cylindrical extension of fuzzy sets

Assume that $\tilde{A}$ is defined on *Y*. The cylindrical extension of $\tilde{A}$ ı *X*×*Y* where *X* is a second universe of discourse, is the fuzzy set of all pairs $(x, y) \in X \times Y$ with membership degree, i.e.

$$\text{ce}\ (\tilde{A}) = \sum_{X \times Y} \frac{\mu_{\tilde{A}}(x)}{(x, y)}$$

where ce denotes the cylindrical operation.

Example1.4.1:

Let $\tilde{A}=\left\{\frac{0.3}{y_1}+\frac{0.5}{y_2}+\frac{0.7}{y_3}\right\}$ and $\tilde{B}=\left\{\frac{0.2}{x_1}+\frac{0.6}{x_2}+\frac{0.9}{x_3}+\frac{1}{x_4}\right\}$ be two fuzzy set on Y and X. Then

$$\mathrm{ce}(\tilde{A})=:\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\begin{array}{c} \begin{array}{ccc} y_1 & \mathrm{y}_2 & \mathrm{y}_3 \end{array} \\ \begin{bmatrix} 0.3 & 0.5 & 0.7 \\ 0.3 & 0.5 & 0.7 \\ 0.3 & 0.5 & 0.7 \\ 0.3 & 0.5 & 0.7 \end{bmatrix} \end{array}, \qquad \mathrm{ce}(\tilde{B})=:\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\begin{array}{c} \begin{array}{ccc} y_1 & \mathrm{y}_2 & \mathrm{y}_3 \end{array} \\ \begin{bmatrix} 0.2 & 0.2 & 0.2 \\ 0.6 & 0.6 & 0.6 \\ 0.9 & 0.9 & 0.9 \\ 1 & 1 & 1 \end{bmatrix} \end{array}$$

1.4.5.1 $\tilde{A}$ Composition of a fuzzy set with a relation

Let $\tilde{A}$ be a fuzzy set on X and R a fuzzy relation defined on $X\times Y$. The composition of $\tilde{A}$ with R is given by

$$\tilde{B}=\tilde{A}\circ R=\mathrm{proj}\left(\mathrm{ce}(\tilde{A})\cap R\right) \text{ on } Y$$

which produces a fuzzy set $\tilde{B}$ on Y with membership degree

$$\mu_{\tilde{B}}(y)=\max_{x}\left\{\min\left(\mu_{\tilde{A}}(x),\mu_R(x,y)\right)\right\}$$

Example1.4.2:

Let $\tilde{A}=\left\{\frac{0.3}{x_1}+\frac{0.5}{x_2}+\frac{0.7}{x_3}+\frac{1}{x_4}\right\}$ be fuzzy set on X and R be a relation defined on $X\times Y$

$$R=:\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\begin{array}{c} \begin{array}{ccc} y_1 & \mathrm{y}_2 & \mathrm{y}_3 \end{array} \\ \begin{bmatrix} 0.3 & 0.7 & 0.25 \\ 0.7 & 0.45 & 0.4 \\ 0.2 & 0.6 & 1 \\ 1 & 0.2 & 0.9 \end{bmatrix} \end{array}$$

Then

$$\mathrm{ce}(\tilde{A})=:\begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\begin{array}{c} \begin{array}{ccc} y_1 & \mathrm{y}_2 & \mathrm{y}_3 \end{array} \\ \begin{bmatrix} 0.3 & 0.3 & 0.3 \\ 0.5 & 0.5 & 0.5 \\ 0.7 & 0.7 & 0.7 \\ 1 & 1 & 1 \end{bmatrix} \end{array}$$

Since $\tilde{B}=\tilde{A}\circ R=\mathrm{proj}\left(\mathrm{ce}(\tilde{A})\cap R\right)$ on Y then $\tilde{B}=\tilde{A}\circ R=\left\{\frac{1}{y_1}+\frac{0.6}{y_2}+\frac{0.9}{y_3}\right\}$.

The composition of the relation R_2 with $\tilde{B}$ is $\tilde{C} = \tilde{B} \circ R_2 = \text{proj}\left(\text{ce}(\tilde{B}) \cap R\right)$ on X with membership degree $\mu_{\tilde{C}}(x) = \max_y \left\{ \min\left(\mu_{\tilde{B}}(y), \mu_{R_2}(x,y)\right)\right\}$ where

$$R_2 =: \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{bmatrix} 0.55 & 0.4 & 0.75 \\ 0.27 & 0.65 & 1 \\ 0.66 & 0.26 & 0.3 \\ 0.3 & 0.6 & 0.7 \end{bmatrix} \end{array}$$

The first task is therefore to calculate the cylinder of B such that:

$$\text{ce}(\tilde{B}) =: \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & \text{y}_2 & \text{y}_3 \end{array} \\ \begin{bmatrix} 1 & 0.6 & 0.9 \\ 1 & 0.6 & 0.9 \\ 1 & 0.6 & 0.9 \\ 1 & 0.6 & 0.9 \end{bmatrix} \end{array}$$

It is then possible to calculate the composition between $\tilde{B}$ and R_2.given by:

$$\begin{aligned} \tilde{C} = \tilde{B} \circ R_2 &= \text{proj}\left(\text{ce}(\tilde{B}) \cap R_2\right) \text{ on X} \\ &= \left\{ \frac{0.75}{x_1} + \frac{0.9}{x_2} + \frac{0.66}{x_3} + \frac{0.7}{x_4} \right\} \end{aligned}$$

1.4.6 Composition of two relations

Assume that R_1 and R_2 are two relations on $X \times Y$ and $Y \times Z$, respectively such that:

$$R_1 =: \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & \text{y}_2 & \text{y}_3 \end{array} \\ \begin{bmatrix} 0.15 & 0.3 & 0.65 \\ 0.2 & 1 & 0.81 \\ 0.76 & 0.3 & 0.4 \\ 0.3 & 0.6 & 0.2 \end{bmatrix} \end{array}, \quad \text{R}_2 =: \begin{array}{c} \\ y_1 \\ y_1 \\ y_1 \end{array} \begin{array}{c} \begin{array}{ccccc} z_1 & z_2 & z_3 & z_4 & z_5 \end{array} \\ \begin{bmatrix} 0.4 & 0.3 & 1 & 0.7 & 1 \\ 0.8 & 1 & 0.3 & 0.1 & 0.4 \\ 0.6 & 0.9 & 0.1 & 1 & 0.6 \end{bmatrix} \end{array}$$

The compositions of these relations is given by $R = R_1 \circ R_2$ with the membership degree

Therefore,

$$\mu_{X \times Z}(x,z) = \max_{y \in Y} \left\{ \min\left(\mu_{X \times Y}(x,y), \mu_{Y \times Z}(y,z)\right)\right\}$$

$$R = R_1 \circ R_2 =: \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccccc} z_1 & z_2 & z_3 & z_4 & z_5 \end{array} \\ \begin{bmatrix} 0.6 & 0.65 & 0.3 & 0.65 & 0.6 \\ 0.8 & 1 & 0.3 & 0.81 & 0.6 \\ 0.4 & 0.4 & 0.76 & 0.7 & 0.76 \\ 0.6 & 0.6 & 0.3 & 0.3 & 0.4 \end{bmatrix} \end{array}$$

The composition of a relation with itself can be obtained in the same manner. For example, $R_1^2 = R_1 \circ R_1$ with membership degree is:

$$\mu_{X\times X}(x_1, x_3) = \max_{x_2 \in X} \{ \min(\mu_{X\times X}(x_1, x_2), \mu_{X\times X}(x_2, x_3)) \}$$

which gives:

$$R_1^2 = R_1 \circ R_1 = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{bmatrix} 1 & 0.3 & 0.65 & 0.3 \\ 0.3 & 1 & 0.3 & 0.6 \\ 0.65 & 0.3 & 1 & 0.3 \\ 0.3 & 0.6 & 0.3 & 1 \end{bmatrix} \end{array}$$

Therefore, for any integer n, the relation $R_1^{n+1} = R_1 \circ R_1^n$ is defined. For example, $R_1^3 = R_1 \circ R_1^2$.

1.5 Approximate reasoning and defuzzification methods

The fuzzy concepts, fuzzy sets and relations have been presented in the previous Section. This section introduces appropriate fuzzy rules illustrated with a set of examples. The rules are normally addressed in linguistic terms that relate to human expressions and understanding. The linguistic variables, hedges and the methods of defuzzification are also described.

1.5.1 Linguistic variable and hedges

Fuzzy logic operates with words and terms of natural language while fuzzy set theory allows for modelling in terms of natural language using linguistic variables. The common linguistic variables are primary terms such as, 'cold', 'warm', 'hot' , 'low', 'medium', 'high'. The connectives include 'AND', 'OR' and 'NOT'. Hedges are qualitative description such as 'very ', 'more or less', slightly, rather and markers such as parenthesis.

For example, consider the temperature of a room or closed area, which is normally, described using the term cold, medium and hot. Assuming that cold is 0°C, medium is 16°C and hot is 36°C, then the membership functions of the three variables (cold, medium and hot) is given in Figure 1.5.1. The terms, very, more or less and slightly may also be used.

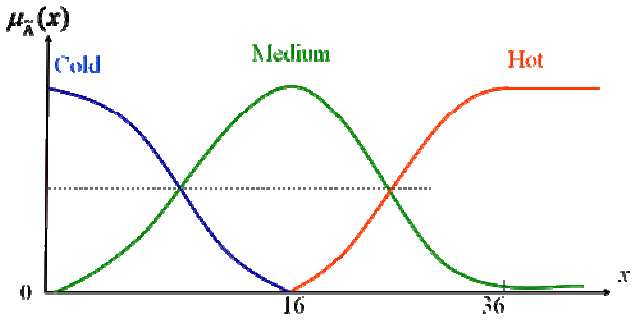


Figure 1.5.1: Membership functions of the temperature of a room.

A linguistic variable is defined as a quadruple $L(X) = (X, T(X), U, M_X)$ in which

- U is the universe (of discourse).
- X is the label or name of the linguistic variable, e.g. blood pressure, demand, number of patients, etc.
- $T(X)$ is the set of the linguistic values of the linguistic variables and usually comprises of primary values: e.g. 'low', 'moderate', 'medium', 'high'.
- $T(X)$ may also include qualifiers such as 'very very low', 'very low', …, 'high', 'very high', or negatives such as 'not hot', 'not cold', 'not very hot'.
- M_X is a function which produces a fuzzy set from the linguistic variables.

  For example, $U = \{1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7\}$ denotes the days of the week starting from Monday (corresponding to day number 1). Given $T(X)$ ={'mid-week', 'early in the week', 'end of the week'} then $M_X(\text{'early in the week'}) = \left\{\frac{1}{1} + \frac{0.75}{2} + \frac{0.2}{3}\right\}$ where $\frac{1}{1}$ represents Monday which memebership is 1 as it is definitely early in the week. $\frac{0.75}{2}$ represents Tuesday (day 2) with a membership of 0.75 (still early in the week but not as early as Monday). $\frac{0.2}{3}$ represents Wednesday (day 3) with a membership of 0.2 (it is not early in the week anymore) $M_X(\text{'mid}-\text{week'}) = \left\{\frac{0.5}{3} + \frac{1}{4} + \frac{0.3}{5}\right\}$ in this case the middle of the week is Thursday (equally spaced between Monday and Sunday), hence its membership is 1 for day 4

1.5.2 Linguistic hedges

Linguistic hedges can be considered as modifiers of primary values in the set $T(X)$ of a linguistic variable X . In the example presented in Section 1.5.1 , some types of modifiers for a primary value

defined as 'week' were 'mid' and 'early'. Other modifiers including-'end', 'very early: 'early in the week', 'end of the week' and 'very early in the week' could also be defined.

Assume that $\tilde{A} = \left\{ \frac{\mu_A(x)}{x} : \ x \in U \ \right\}$ is a fuzzy set. Using linguistic hedges, the following operations on fuzzy membership functions can be defined:

(a) Concentration (very $\tilde{A}$):

$$\mu_{CON(\tilde{A})} = (\mu_{\tilde{A}}(x))^2$$

(b) Dilatation (more or less $\tilde{A}$):

$$\mu_{DILL(\tilde{A})} = (\mu_{\tilde{A}}(x))^{1/2}$$

(c) Plus:

$$\mu_{PLUS(\tilde{A})} = (\mu_{\tilde{A}}(x))^{5/4}$$

(d) Contrast intensification (INT) :

$$\mu_{INT(\tilde{A})} = \begin{cases} 2(\mu_{\tilde{A}}(x))^2 & \text{if } \ \mu_{\tilde{A}}(x) \geq 0.5 \\ 1 - 2(1 - \mu_{\tilde{A}}(x))^2 & \text{if } \ \mu_{\tilde{A}}(x) < 0.5 \end{cases}$$

(e) Slightly $\tilde{A}$ =INT(NORM(plus $\tilde{A}$ and not (very $\tilde{A}$))), where NORM(plus $\tilde{A}$ and not (very $\tilde{A}$)) denotes the fuzzy set whose membership degrees are obtained from division all the membership degrees of the fuzzy set 'plus $\tilde{A}$ and not (very $\tilde{A}$)' by the largest membership value.

Examples 1.5.1:

(a) Operator 'very': $\text{very}(\text{very}(\tilde{A})) = (\text{very}(\tilde{A}))^2 = \left((\mu_{\tilde{A}}(x))^2\right)^2 = (\mu_{\tilde{A}}(x))^4$.

(b) Assume that P is the linguistic variable for blood pressure, U=[600 mmHg, 1500mm Hg] is the universal set and $\tilde{H}$ is a fuzzy set of high blood pressure. Using the linguistic connectives (**or**, **and**), the following statements can be implied

'blood pressure is high' **and** 'blood pressure is medium.'

'blood pressure is high' **or** 'blood pressure is medium.'

Note that only the second statement may be true.

(c) Let $\tilde{H} = \left\{ \frac{0.1}{700} + \frac{0.8}{800} + \frac{0.95}{900} + \frac{1}{1000} \right\}$ and $\mu_{VH} = (\mu_{\tilde{H}}(x))^2$ Be the membership function. then $VH = \left\{ \frac{0.01}{700} + \frac{0.64}{800} + \frac{0.9}{900} + \frac{1}{1000} \right\}$ .Assume that $\neg\mu_{VH}(x)$ denotes for linguistic 'not blood pressure is very high'. Since $\neg\mu_{VH}(x) = 1 - \mu_{VH}(x)$ it follows:

$$\neg VH = \left\{ \frac{1}{600} + \frac{0.99}{700} + \frac{0.36}{800} + \frac{0.1}{900} + \frac{0}{1000} + \frac{1}{1100} + \frac{1}{1200} + \frac{1}{1300} + \frac{1}{1400} + \frac{1}{1500} \right\}$$

Therefore,

$$\neg VH \cap \tilde{H} = \tilde{C} = \left\{ \frac{0.1}{700} + \frac{0.36}{800} + \frac{0.1}{900} \right\}$$

This fuzzy set is not normal as the value of membership degree for all elements are less than 1. As stated before, to normalise this fuzzy set, all membership degrees are divided by the largest value, i.e. $\frac{\mu_{\tilde{C}}(x)}{0.36}$. Therefore,

$$\text{NORM}(\tilde{C}) = \left\{ \frac{0.27}{700} + \frac{1}{800} + \frac{0.27}{900} \right\}$$

1.5.3 Modus ponens inference scheme

Sometimes it is required to obtain a true statement using mathematical logic theory. Here an important mathematical logic rule, the so-called modus ponens, is presented. Assume that *p* and *q* are two statements, then 'if *p* then *q*' is termed a conditional statement and is represented by '$p \rightarrow q$'. Here *'p', 'q'* and '$p \rightarrow q$' are called proposition (antecedent), inference (consequent) and implication, respectively. Assume that *p* and $p \rightarrow q$ are true statements then *q* is also true. In fact, $\left(p \wedge (p \rightarrow \mathrm{q})\right) \rightarrow q$ where $\wedge$ indicates 'AND' connective. The modus ponens inference scheme can be generalised. Assume that $p \rightarrow q$ is an implication and $\hat{p}$ is the proposition which is an extension of the proposition *p.* Then the same quality applies to the inference $\hat{q}$, which can be concluded as an extension of *q*, exactly in same level as $\hat{p}$ is the extension of *p.*

For example, assuming that the following rules exist: if the weather is hot then the humidity is high, then it can be derived that: if the weather is very hot then the humidity is very high.

Generalised modus ponens inference can be summarised as follows:

(a) Assume that *A* and *B* are the linguistic variables.

(b) $\tilde{A}$ and $\tilde{B}$ are the values of the linguistic variables *A* and *B, i.e.* low, medium and high

(c) *'A* is $\tilde{A}_1$'

(d) 'If A is $\tilde{A}_1$ then B is $\tilde{B}$ '

(e) 'B is $\tilde{B}_1$

Therefore,

Rule: If A is $\tilde{A}_1$ then B is $\tilde{B}$ '.

Fact: A is $\tilde{A}_1$.

Conclusion: B is $\tilde{B}$

Example 1.5.2:

Suppose that *p* and *q* refer to the statements 'the salt intake is low' and 'blood pressure will be low', respectively. Now consider the following conditional statement:

$p \rightarrow q$: 'If salt intakes is low, then blood pressure will be low.'

Assume that the conditional statement is true and a normal person use a low amount of salt. Then the blood pressure of this person should be low.

Suppose that a normal person uses very low salt, i.e. $\hat{q}$ stands for 'The salt intake is very low'. Also assume that

$p \rightarrow q$: ' If salt intakes is low, then blood pressure will be low'.

is an implication. Then $\hat{q}$ which indicates that ' blood pressure will be very low' is implied. Therefore, $\left(\hat{p} \wedge (p \rightarrow \mathrm{q})\right) \rightarrow \hat{q}$ ' If salt intakes is VERY low, then blood pressure will be VERY low'.

1.6 The Mamdani implication

This section describes the widely used Mamdani implication. The Mamdani implication is equivalent to the 'AND' operation which means that a conditional statement, 'if p then q, is true' if and only if the proposition (p) and the inference (q) are true. Note that in the classical mathematical logic, the conditional statement 'if p then q' is false, if and only if the statement p is true and the statement q is false. For the rest of cases the conditional statement is true.

Assume that that *p* and *q* are two statements, the Mamdani implication is summarised in Table 1.6.1 in which 'M' refers to the implication in the Mamdani sense.

p	*q*	$P \xrightarrow{M} q \leftrightarrow p \wedge q$

T	T	T
T	F	F
F	T	F
F	F	F

Table 1.6.1: The Mamdani implication.

Since the Mamdani implication is equivalent to 'AND', the operation which describes the minimum. The following example shows the Mamdani implication.

Example 1.6.1: Let $\tilde{A}=\left\{\frac{0.20}{3}+\frac{0.80}{4}+\frac{1}{5}+\frac{1}{6}\right\}$ and $\tilde{B}=\left\{\frac{0.40}{3}+\frac{0.80}{4}+\frac{1}{5}\right\}$ be two fuzzy sets. Then the denoted R_M relation obtained from the min operation (Mamdani implication) is given by

$$R_M = X: \quad \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} \multicolumn{6}{c}{Y} \\ 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.2 & 0.2 & 0.2 \\ 0 & 0 & 0 & 0.4 & 0.8 & 0.8 \\ 0 & 0 & 0 & 0.4 & 0.8 & 1 \\ 0 & 0 & 0 & 0.4 & 0.8 & 1 \end{array}\right] \end{array}$$

Note that the pairs with zero membership degrees do not need to be included in the relation. .

If $\tilde{A}'=\left\{\frac{0}{1}+\frac{0}{2}+\frac{1}{3}+\frac{0}{4}+\frac{0}{5}+\frac{0}{6}\right\}=\left\{\frac{1}{3}\right\}$ is a given fuzzy set then a new fuzzy set is obtained as follows:

$$\begin{aligned}\tilde{B}' = \tilde{A}' \circ R_M &= \text{proj}(\text{ce}(\tilde{A}') \cap R_M) \\ &= \left\{\frac{0}{1}+\frac{0}{2}+\frac{0.2}{3}+\frac{0.2}{4}+\frac{0.2}{5}\right\}\end{aligned}$$

1.7 Defuzzification Methods

In this section the results of inference or reasoning processes are interpreted so that they become useable. The final stage in using a fuzzy decision aid is the determination of a crisp decision which might be a number which represents how many items to order, how many staff needs to be employed, or by how many percent a radiator valve should be opened to keep a room at a constant temperature. The process of extracting a crisp value from some techniques or contributing sets is called defuzzification. There are a number of methods for performing defuzzification. The common techniques are presented in this section.

Assume that $U = \{u_1, u_2, ..., u_N\}$ is the universe of discourse and $\mu_{\tilde{Q}_j}(u_i),\ j = 1, ..., m$ are the membership functions. The clipped fuzzy sets, illustrated in Figure 1.7.1, can then be obtained by considering α_j as the membership degrees of all elements whose membership degree is larger than α_j, that is

$$\mu_{C(\tilde{Q}_j)} = \begin{cases} \mu_{\tilde{Q}_i}(u_i) & \text{if } \mu_{\tilde{Q}_j}(u_i) \le \alpha_j \\ \alpha_j & \text{if } \mu_{\tilde{Q}j}(u_i) > \alpha_j \end{cases}, \quad j = 1, ..., m$$

where $\mu_{C(\tilde{Q}_j)}$ is the membership function of $C(\tilde{Q}_j)$. , Assume that $C(\tilde{Q}_1), C(\tilde{Q}_2), ..., C(\tilde{Q}_N)$ are α_j -clipped fuzzy. .

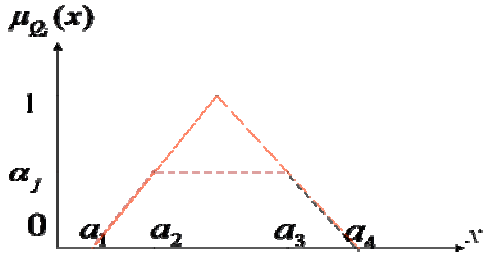


Figure 1.7.1: Clipped fuzzy sets (α_j -cut)

The process of defuzzification is summarised as follows:

(a) Assume that the number of clipped sets is N .

(b) A single fuzzy set is constructed using the $\cup$ operation. The resultant fuzzy set is then determined as follows

$$\tilde{Q} = \cup \{ c(\tilde{Q}_1) : \alpha_1, c(\tilde{Q}_2) : \alpha_2, ..., c(\tilde{Q}_N) : \alpha_N \}$$

(c) Sample points on the x axis representing the independent variable should be taken sufficiently close together to increase the accuracy of the solution.

(d) Calculate the fuzzy output using the membership degrees of the selected samples.

Note that a set of samples together with their membership degrees forms a fuzzy set.

1.7.1 The centre of area method (centre of gravity)

1.7.1.1 Discrete case

For completion of the defuzzification process, crisp numbers are required

Assume that $U = \{ u_1, u_2, ..., u_N \}$ in the universal set and $n \le N$. The defuzzified 'crisp' value denoted q^* is given by

$$q^* = \frac{\sum_{i=1}^{n} \mu_{\tilde{Q}}(u_i) u_i}{\sum_{i=1}^{n} \mu_{\tilde{Q}}(u_i)} = \frac{\sum_{i=1}^{n} \max_j \mu_{C(\tilde{Q}_j)}(u_i) u_i}{\sum_{i=1}^{n} \max_j \mu_{C(\tilde{Q}_j)}(u_i)}$$

See Figure 1.7.2

.

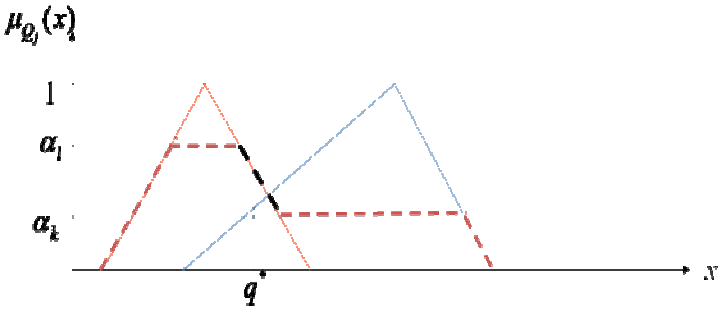


Figure 1.7.2: Defuzzification using the centre of area.

Example 1.7.1:

Let $U = \{0, 1, ..., 10\}$, $\tilde{Q}_1 = \left\{ \frac{0.5}{2} + \frac{0.75}{3} + \frac{0.5}{4} \right\}$, $\tilde{Q}_2 = \left\{ \frac{0.25}{4} + \frac{0.25}{5} + \frac{0.25}{6} \right\}$ and the clipping level for $\tilde{Q}_1$ and $\tilde{Q}_2$ be 0.75 and 0.25, respectively. Then

$$\tilde{Q} = \tilde{Q}_1 \cup \tilde{Q}_2 = \left\{ \frac{0.5}{2} + \frac{0.75}{3} + \frac{0.5}{4} + \frac{0.25}{5} + \frac{0.25}{6} \right\}$$

Using the centre of area method yields

$$q^* = \frac{\sum_{i=1}^{n} \mu_{\tilde{Q}}(u_i) u_i}{\sum_{i=1}^{n} \mu_{\tilde{Q}}(u_i)} = \frac{\frac{1}{2}(2) + \frac{3}{4}(3) + \frac{1}{2}(4) + \frac{1}{4}(5) + \frac{1}{4}(6)}{\frac{1}{2} + \frac{3}{4} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4}} = \frac{32}{9} = 3.56$$

1.7.1.2 Continuous case

Assume that $X \subseteq U$ where the universal set is U. The defuzzified 'crisp' value is denoted by

$$q^* = \frac{\int_X \mu(x)x dx}{\int_X \mu(x)dx}$$

Example 1.7.2: Assume that

$$\mu_1(x) = \frac{1}{2}(x-1), \quad 1 \le x \le 3$$
$$\mu_2(x) = \frac{1}{2}(5-x), \quad 3 \le x \le 5$$
$$\mu_3(x) = \frac{1}{2}(x-3), \quad 3 \le x \le 5$$
$$\mu_4(x) = \frac{1}{2}(7-x), \quad 5 \le x \le 7$$

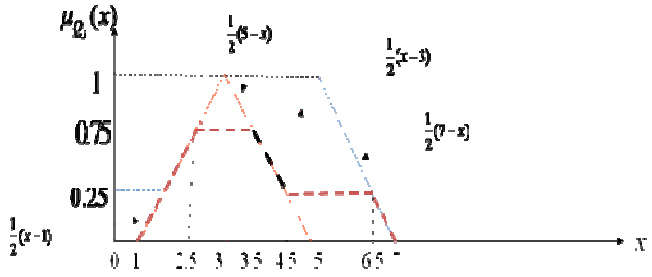


Figure 1.7.3: The centre of area method for Figure 1.7.3.

The clipped fuzzy sets at level ¾ (3/4-cut) are $\frac{3}{4} = \mu_1(x) = \frac{1}{2}(x-1)$ and $\frac{3}{4} = \mu_2(x) = \frac{1}{2}(5-x)$. Therefore, solving for x $\frac{3}{4} = \frac{1}{2}(x-1)$ gives $x = 5/2$ and $\frac{3}{4} = \frac{1}{2}(5-x)$ gives $x = 7/2$. The clipped fuzzy sets at level 1/4 (1/4-cut) are obtained from $\frac{1}{4} = \mu_3(x) = \frac{1}{2}(x-3)$ and $\frac{1}{4} = \mu_4(x) = \frac{1}{2}(7-x)$. Thus $x = 7/2$ and $x = 13/2$. The overall membership function shown in Figure 1.7.3 is given by:

$$\mu(x) = \begin{cases} 0, & 0 \le x \le 1 \\ \frac{1}{2}(x-1), & 1 \le x \le 2.5 \\ 0.75, & 2.5 \le x \le 3.5 \\ \frac{1}{2}(5-x), & 3.5 \le x \le 4.5 \\ 0.25, & 4.5 \le x \le 6.5 \\ \frac{1}{2}(7-x), & 6.5 \le x \le 7 \\ 0, & 7 \le x \le 10. \end{cases}$$

The centre of gravity $q^* = \dfrac{\int_X \mu(x)xdx}{\int_X \mu(x)dx}$ is obtained. For obtaining the numerator and the denominator the following integrals must be calculated:

$$\int_1^{2.5} \frac{1}{2}(x-1)xdx + \int_{2.5}^{3.5} \frac{3}{4}xdx + \int_{3.5}^{4.5} \frac{1}{2}(5-x)xdx + \int_{4.5}^{6.5} \frac{1}{4}xdx + \int_{6.5}^{7} \frac{1}{2}(7-x)xdx$$

$$\int_1^{2.5} \frac{1}{2}(x-1)dx + \int_{2.5}^{3.5} \frac{3}{4}dx + \int_{3.5}^{4.5} \frac{1}{2}(5-x)dx + \int_{4.5}^{6.5} \frac{1}{4}dx + \int_{6.5}^{7} \frac{1}{2}(7-x)dx$$

$$\int_1^{2.5} \frac{1}{2}(x-1)xdx = 1.125,\ \int_{2.5}^{3.5} \frac{3}{4}xdx = 2.25,\ \int_{3.5}^{4.5} \frac{1}{2}(5-x)xdx = 1.9583, \int_{4.5}^{6.5} \frac{1}{4}xdx = 2.75,\ \int_{6.5}^{7} \frac{1}{2}(7-x)xdx = 0.4167$$

$$\int_1^{2.5} \frac{1}{2}(x-1)dx = 0.5625,\ \int_{2.5}^{3.5} \frac{3}{4}dx = 0.75,\ \int_{3.5}^{4.5} \frac{1}{2}(5-x)dx = 0.55,\ \int_{4.5}^{6.5} \frac{1}{4}dx = 0.5,\ \int_{6.5}^{7} \frac{1}{2}(7-x)dx = 0.0625$$

Therefore**,**

$$q^* = \frac{1.125 + 2.25 + 1.9583 + 2.75 + 0.41667}{0.5625 + 0.75 + 0.5 + 0.5 + 0.0625} = \frac{8.5}{2.375} = 3.58$$

1.7.2 The centre of sums method

1.7.2.1 Discrete case

Assume that $U = \{ u_1, u_2, ..., u_N \}$ in the universal set and, $n \le N$ then the defuzzified 'crisp' value using the centre of sums method for the discrete case is denoted:

$$q^* = \frac{\sum_{i=1}^{n} u_i \sum_{j=1}^{m} \mu_{C(\tilde{Q}_j)}(u_i)}{\sum_{i=1}^{n} \sum_{j=1}^{m} \mu_{C(\tilde{Q}_j)}(u_i)}$$

Example 1.7.3:

Consider **Example 1.7.1**. Now the centre of sums method is used to find the crisp value.

$$q^* = \frac{\sum_{i=1}^{11} u_i \sum_{j=1}^{2} \mu_{C(\tilde{Q}_j)}(u_i)}{\sum_{i=1}^{11} \sum_{j=1}^{2} \mu_{C(\tilde{Q}_j)}(u_i)} = \frac{\sum_{i=1}^{11} u_i [\mu_{C(\tilde{Q}_1)}(u_i) + \mu_{C(\tilde{Q}_2)}(u_i)]}{\sum_{i=1}^{11} [\mu_{C(\tilde{Q}_1)}(u_i) + \mu_{C(\tilde{Q}_2)}(u_i)]}$$

$$= \frac{2\left[\frac{1}{2}+0\right] + 3\left[\frac{3}{4}+0\right] + 4\left[\frac{1}{2}+\frac{1}{4}\right] + 5\left[0+\frac{1}{4}\right] + 6\left[0+\frac{1}{4}\right]}{\left[\frac{1}{2}+0\right] + \left[\frac{3}{4}+0\right] + \left[\frac{1}{2}+\frac{1}{4}\right] + \left[0+\frac{1}{4}\right] + \left[0+\frac{1}{4}\right]}$$

$$= \frac{9}{2.5} = 3.6$$

1.7.2.2 Continuous case

Assume that $X \subseteq U$ where the universal set is U. The defuzzified 'crisp' value is denoted by

$$q^* = \frac{\int_X x \sum_{i=1}^{n} \mu_{C\tilde{Q}_i}(x)dx}{\int_X \sum_{i=1}^{n} \mu_{C\tilde{Q}_i}(x)dx}$$

Example 1.7.4: Consider **Example 1.7.1**. Now the centre of sums method is used to find the crisp value. Therefore the following integrals must be calculated:

$$I_1 = \int_1^{2.5} x\frac{(x-1)}{2}dx = 1.125, \quad I_2 = \int_{2.5}^{3} 0.75xdx = 1.0313, I_3 = \int_3^{3.5} x\left(0.75 + \frac{(x-3)}{2}\right)dx = 1.4271, \quad I_4 = \int_{3.5}^{5} x\left(0.25 + \frac{(5-x)}{2}\right)dx = 3.8438,$$

$$I_5 = \int_5^{6.5} 0.25xdx = 2.1563, \qquad I_6 = \int_{6.5}^{7} x\frac{(7-x)}{2}dx = 0.41667$$

$$\hat{I}_1 = \int_1^{2.5} \frac{x-1}{2}dx = 0.5625, \quad \hat{I}_2 = \int_{2.5}^{3} 0.75dx = 0.375, \hat{I}_3 = \int_3^{3.5}\left(0.75 + \frac{(x-3)}{2}\right)dx = 0.4375, \quad \hat{I}_4 = \int_{3.5}^{5}\left(0.25 + \frac{5-x}{2}\right)dx = 0.9375,$$

$$\hat{I}_5 = \int_5^{6.5} 0.25dx = 0.375, \quad \hat{I}_6 = \int_{6.5}^{7} \frac{7-x}{2}dx = 0.0625$$

The numerator is

$$I = \sum_{i=1}^{6} I_i = 1.1250 + 1.0313 + 1.4271 + 3.8438 + 2.1563 + 0.4167 = 10$$

The denominator is

$$\hat{I} = \sum_{i=1}^{6} \hat{I}_i = 0.5625 + 0.375 + 0.4375 + 0.9375 + 0.375 + 0.0625 = 2.75$$

Therefore,

$$q^* = \frac{10}{2.75} = 3.67$$

1.7.3 The centre of the height method

Let the locations of elements with maximum degrees in the (unclipped) fuzzy set $\tilde{Q}_i, i = 1, \ldots, n,$ be $u_i^m, i = 1, \ldots, n,$ with the membership degree $\overline{\mu}_i$. Then the defuzzified value using the height method is

$$q^* = \frac{\sum_{i=1}^{n} u_i^m \overline{\mu}_i}{\sum_{i=1}^{n} \overline{\mu}_i}$$

Example 1.7.5:

Consider **Example 1.7.1**. The maximum membership degrees of two fuzzy sets are $\overline{\mu}_1 = \frac{3}{4}$ and $\overline{\mu}_2 = \frac{1}{4}$ at the locations $u_1^m = 3$ and $u_2^m = 5$. The defuzzified value is obtained with the weighted average method such that:

$$q^* = \frac{\left(\frac{3}{4} \times 3\right) + \left(\frac{1}{4} \times 5\right)}{\frac{3}{4} + \frac{1}{4}} = \frac{14}{4} = 3.5$$

Note that this method is also applicable to the continuous case.

1.8 Defuzzification methods and decision making process

Using a defuzzification method a crisp value is obtained and a user has to interpret this value and make a decision based on this interpretation. In this section the differences between the three defuzzification methods presented are highlighted and the decision making process explained. **Example 1.8.1:** Assume that an organisation needs to order supplies. The hedge variables 'Large', 'Above normal'', 'Normal', 'Below Normal' and 'Small' indicate the order sizes. Let $U = \{0,\ 1,\ 2,\ 3,\ 4\}$ be the universe of discourse, and the fuzzy sets

$$Q_1 = \left\{\frac{0}{0} + \frac{0}{1} + \frac{0}{2} + \frac{0}{3} + \frac{1}{4}\right\},\ Q_2 = \left\{\frac{0}{0} + \frac{0}{1} + \frac{0}{2} + \frac{1}{3} + \frac{0.7}{4}\right\},\ Q_3 = \left\{\frac{0}{0} + \frac{0}{1} + \frac{1}{2} + \frac{0}{3} + \frac{0}{4}\right\},$$
$$Q_4 = \left\{\frac{0.8}{0} + \frac{1}{1} + \frac{0}{2} + \frac{0}{3} + \frac{0}{4}\right\},\ Q_5 = \left\{\frac{1}{0} + \frac{0}{1} + \frac{0}{2} + \frac{0}{3} + \frac{0}{4}\right\}$$

represent the order sizes: 'Large', 'Above normal'', 'Normal', 'Below Normal' and 'Small', respectively.

The defuzzification process includes the following steps:

i) Calculate the clipped fuzzy sets

ii) Use the maximum operation to find the firing strength of each element

iii) Use centre of gravity or the centre of sums or the maximum height to find a single crisp value that represents the fuzzy output.

The six clipping levels of the fuzzy sets Q_1, to Q_5 were arbitrarily selected and are given in Table 1.1.

Above Normal	0.29
Normal	0.29 0.51 0.005
Below Normal	0.55 0.05

Table 1.1: The nonzero firing strengths.

The six clipped fuzzy sets are

$\tilde{Q}_1$, the' Clipped fuzzy set 'Below Normal' at level point is: $0.55' = \left\{\frac{0.55}{0} + \frac{0.55}{1}\right\}$

$\tilde{Q}_2$, the 'Clipped fuzzy set 'Normal' at level point is: $0.51' = \left\{\frac{0.51}{2}\right\}$

$\tilde{Q}_3$, the 'Clipped fuzzy set 'Above Normal' at level point is: $0.29' = \left\{\frac{0.29}{3} + \frac{0.29}{4}\right\}$

$\tilde{Q}_4$ 'Clipped fuzzy set 'Normal' at level point $0.29' = \left\{\frac{0.29}{2}\right\}$

$\tilde{Q}_5$ 'Clipped fuzzy set 'Normal' at level point $0.005' = \left\{\frac{0.005}{2}\right\}$

$\tilde{Q}_6$ 'Clipped fuzzy set 'Below Normal' at level point $0.05' = \left\{\frac{0.05}{0} + \frac{0.05}{1}\right\}$

The second step is to apply the maximum operation on the clipped fuzzy sets $\tilde{Q}_1$ to $\tilde{Q}_6$.

$$Q_c = \tilde{Q}_1 \cup \tilde{Q}_2 \cup \tilde{Q}_3 = \left\{\frac{0.55}{0} + \frac{0.55}{1} + \frac{0.51}{2} + \frac{0.29}{3} + \frac{0.29}{4}\right\}$$

Note that $Q_c = \tilde{Q}_1 \cup \tilde{Q}_2 \cup \tilde{Q}_3 = \tilde{Q}_1 \cup \tilde{Q}_2 \cup \tilde{Q}_3 \cup \tilde{Q}_4 \cup \tilde{Q}_5 \cup \tilde{Q}_6$ because the clipped sets $\tilde{Q}_1$ to $\tilde{Q}_3$ were calculated to give the maximum membership degrees.

The third step is to calculate the fuzzy output using three defuzzification methods.

The defuzzified value using the centre of area is given by:

$$q^* = \frac{0.55\times 0 + 0.55\times 1 + 0.51\times 2 + 0.29\times 3 + 0.29\times 4}{0.55 + 0.55 + 0.51 + 0.29 + 0.29} = \frac{3.6}{2.19} = 1.644$$

The centre of sums method is calculated as follows:

The numerator is 4.24 and calculated using the summation of the following terms

$$(0.55+0.05)\times 0=0, \qquad (0.55+0.05)\times 1=0.6, \qquad (0.29+0.51+0.005)\times 2=0.805\times 2=1.61$$
$$(0.29)\times 3=0.87, \qquad (0.29)\times 4=1.16$$

The denominator is 2.585 calculated as the following sum:

$$0.05+0.05+0.55+0.55+0.29+0.51+0.005+0.29+0.29=2.585$$

Therefore, the defuzzified value using the centre of sums is

$$q^{*}=\frac{4.24}{2.585}=1.64$$

Finally, applying the height method to calculate the fuzzy output is given by the maximum membership degrees at the following locations:

$$u^{\mathrm{m}}_{Below\ Normal}=1$$
$$u^{m}_{Normal}=2$$
$$u^{\mathrm{m}}_{Above\ \mathrm{Normal}}=3$$

The defuzzified value is

$$q^{*}=\frac{(0.55+0.05)\times 1+(0.29+0.51+0.005)\times 2+(0.29)\times 3}{0.55+0.05+0.29+0.51+0.005+0.29}=\frac{3.08}{1.695}$$
$$=1.82$$

To compare three defuzzification methods it is assumed that the universe of discourse $U=\{0,1,2,3,4\}$ corresponds to the order size between 50% and 150% of the normal size. Therefore the following information is obtained:

(a) 0 corresponds to 50% of the normal order size.

(b) 1 corresponds to 75% of the normal order size.

(c) 2 corresponds to 100% of the normal order size.

(d) 3 corresponds to 125% of the normal order size.

(e) 4 corresponds to 150% of the normal order size.

Note that the values between 0 and 1 correspond to 75%-50% = 25% of the normal order size.

The centre of areas defuzzified values 1.64 corresponds to order size

$$75\% + 0.644 \times 25\% = (75 + 16.1)\% = 91.1\%$$

The defuzzified values 1.65 corresponds to order size

$$75\% + 0.64 \times 25\% = (75 + 16)\% = 91\%$$

The height defuzzified values 1.82 corresponds to order size.

$$75\% + 0.82 \times 25\% = 95.5\%$$

Therefore, the order sizes obtained from the centre of areas and the centre of sums are nearly the same while order size resulted from the height method is slightly larger.

The above example shows that the centre of gravity (area) is more appropriate as all data are used with maximum membership degree of each element in the calculations and the final answer does not suffer from extreme data (very low or very high).

The centre of sums use all the membership degrees of the elements (not only the maximum value).

The height method is the simplest method; however it only considers the elements with the maximum memberships. Therefore, it is not a suitable method for sensitive problems.

1.9 Fuzzy Controllers

> "Fuzzy logic control (FLC) may be viewed as a branch of intelligent control which serves as an emulator of human decision-making behaviour that is approximate rather than exact."
>
> (C.C.Lee in Singh: Systems and Control Encyclopaedia, 1992)

Fuzzy control design methods are now popular methods for the control of dynamical systems, particularly when it may be difficult to express the system behaviour mathematically, but it may be possible to describe the behaviour based on human thinking, heuristic judgement. In this section, the procedures underpinning the design of fuzzy logic controllers (FLC) is introduced. These controllers are based on Mamdani inference, Takagi-Sugeno and PID approaches. A fuzzy control law may be described by a knowledge-based system consisting of **IF...THEN** rules with vague predicates and a fuzzy logic inference mechanism. The fuzzy system is formed by a family of logical rules that describe the relationship between the input and the output of a system or controller. In general, the procedure to design a FLC system is as follows:

(a) **Fuzzification.** The universal sets and fuzzy sets are defined. Then the membership functions for all the fuzzy sets which describe a problem are specified.

(b) **Rule evaluation**. Appropriate fuzzy rules are first defined. The fuzzy rules are in the form of a series of '**if … then …**' statements. There are mainly two fuzzy sets of rules comprising of plant rules and control rules. If there is an observer, then a third set of observer rules is also required.

(c) **Defuzzification.** The crisp results which are termed the final outputs are obtained. This output is normally is found using the centre of gravity (or centre of area) method.

These three processes are illustrated in Figure 1.9.1.

1.9.1 Rule format, implication and inference

To create a fuzzy algorithm, a set of fuzzy rules must be defined. It is normally required to define n rules to describe the entire set of scenarios. When there are two variables, the rules are defined in the following form:

For every $1 \leq i \leq n$, R_i: **If** x is A_i and y is B_i **then** z is C_i.

where ' x is A_i and y is B_i ' describe the propositions and 'z is C_i' describe the inferences.

There are three processes employed to create a set of rules for the fuzzy controller, these are rule format, inference and implication. During the rule process, a set of rules are introduced, for example, If temperature is x° C then the cooling system is adjusted to y watts). Inference is a statement that can be concluded from a proposition, for example assumptions on the adjustment of the cooling system, e.g. the cooling system is adjusted to y watts. The implication is a conditional statement in the form of If-Then. Note that an assumption on temperature, say temperature is x° C, is a proposition. For example, if it is required to control the temperature of a room, a rule may be defined as follows:

'**IF** room is warm **THEN** set cooling at 450 watts.'

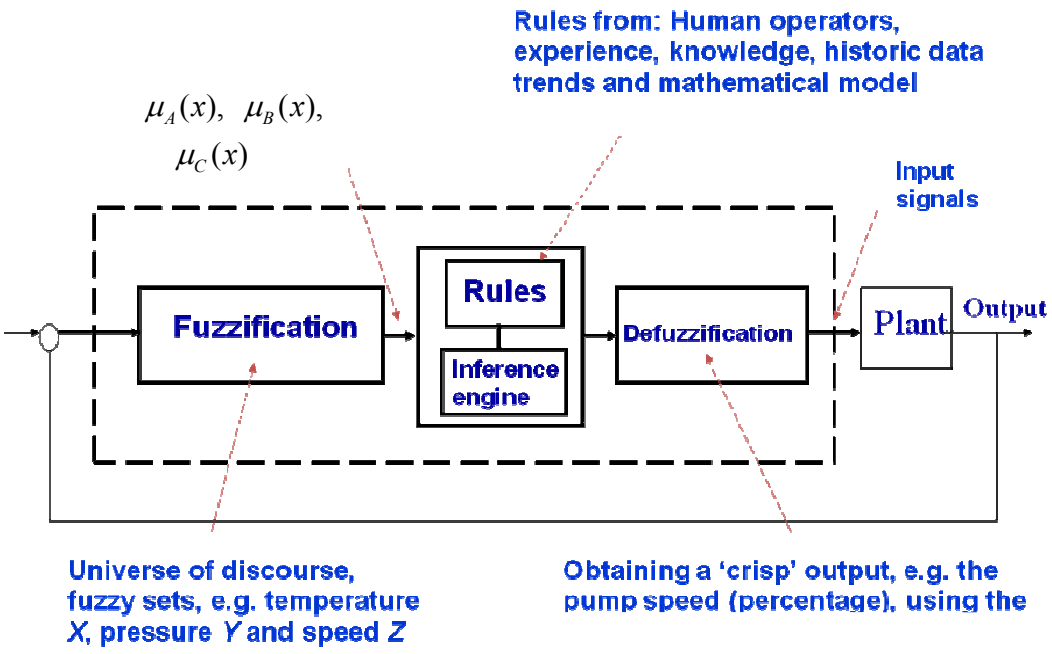


Figure 1.9.1 Fuzzy Control Design procedure.

The following examples illustrate the creation of rules.

Example 1.9.1: illustrates the creation of rules for two different scenarios likely to occur

Scenario 1: Assume that the current room temperature is 19° C and the cooling system has been adjusted to 450 watts. Suppose that a lower temperature is required. Therefore, the cooling system should be regulated to a value less than 450 watts. Based on available previous data, or user knowledge and experience, the cooling system should be appropriately tuned, for example the cooling system should be adjusted at 230 watts.

The rule could then be written as:'**IF** room is 19 degree **AND** new target temperature is low **AND** current cooling is at 450 watt **THEN** set cooling at 230 watt.'

Scenario 2: Aassume that the room temperature is 8° C and a higher temperature, say 16 ° C, is required. Similarly, based on available previous data, or user knowledge and experience, the cooling system should be regulated to 380 watts. The rule could then be written as:

'**IF** room is 8 degree **AND** new target temperature is 16 **THEN** set cooling at 380 watt.'

Example 1.9.2: It is assumed that a vehicle is approaching a train track and the road will close to enable the train to cross the road. As the train approaches the barriers located on either side of the track are lowered forcing the traffic to stop. The driver of a vehicle which is approaching a train track has to use the brake depending on its distance from the barrier. The usage of the brake depends on the vehicle speed and the distance from the vehicle to the barrier.

Assuming that the barrier is closed, the vehicle must stop. Based on the vehicle speed and its distance the following rules may be derived**:**

Rule 1: If distance is long and the vehicle speed is high, then the pressure on the brake pedal is medium.

Rule 2: If distance is long and the vehicle speed is low, then the pressure on the brake pedal is zero.

Rule 3: If distance is short and the speed is high then the pressure on the brake pedal is hard.

Rule 4: If distance is short and vehicle speed is low, then pressure on the brake pedal is medium.

The above information is summarised in Table 1.2. If L, S and L_s and H are variables that stand for the distance, speed, low and high, respectively. The matrix in summarises all information in Table 1.3.

Distance	Speed	Pressure on the break
Long	High	Medium
Long	Low	Zero
Short	High	Hard
Short	Low	Medium

Table 1.2: The tabular rule format for a vehicle approaching a train track when the stop bar is in horizontal position

		Speed	
		L	H
Distance	S	Medium	Hard
	L	Zero	Medium

Figure 1.9.2: The associated matrix for a vehicle approaching a level crossing.

The following example illustrates the overall process starting from rule definition and moving on to defuzzification to calculate the fuzzy output.

Example 1.9.3:

Assume that the valve of a pipe must be controlled to obtain a specific temperature. The pressure of the fluid is an influencing quantity as a high pressure input flow of hot fluid will increase the output temperature. The quantities pressure and temperature are described as low, medium and high while the valve position (controller) is given in percent. Therefore the control variable is the valve position. From a fuzzy control design view point, there are two inputs and one output. A diagram matrix representation of a fuzzy control system is shown in Figure 1.9.3.

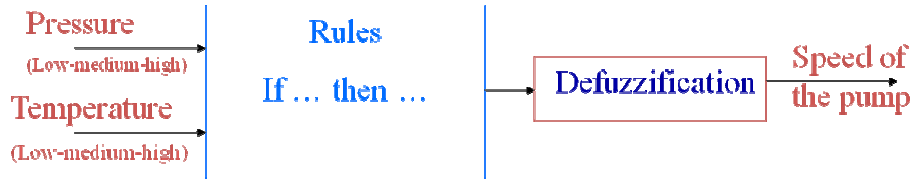


Figure 1.9.3: Fuzzy control design for controlling the speed of a pump.

Fuzzy rules are given by

(1) If ‘P is low’ and ‘T is low’ then ‘valve position is high (90%)’.

(2) If ‘P is medium’ and ‘T is low’ then ‘valve position is high (90%)’.

(3) If ‘P is high’ and ‘T is low’ then ‘valve position is medium (50%)’.

(4) If ‘P is low’ and ‘T is medium’ then ‘valve position is high (90%)’.

(5) If ‘P is medium’ and ‘T is medium’ then ‘valve position is medium (50%)’.

(6) If ‘P is high’ and ‘T is medium’ then ‘valve position is low (10%)’.

(7) If ‘P is low’ and ‘T is high’ then ‘valve position is high (50%)’.

(8) If ‘P is medium’ and ‘T is high’ then ‘valve position is medium (50%)’.

(9) If ‘P is high’ and ‘T is high’ then ‘valve position is low (10%)’.

Let P_l, P_m, P_h denote the low, medium and high pressure and T_l, T_m, T_h denote low, medium and high temperature, respectively. The fuzzy membership functions are arbitrarily selected to be triangular see Figure 1.9.4.

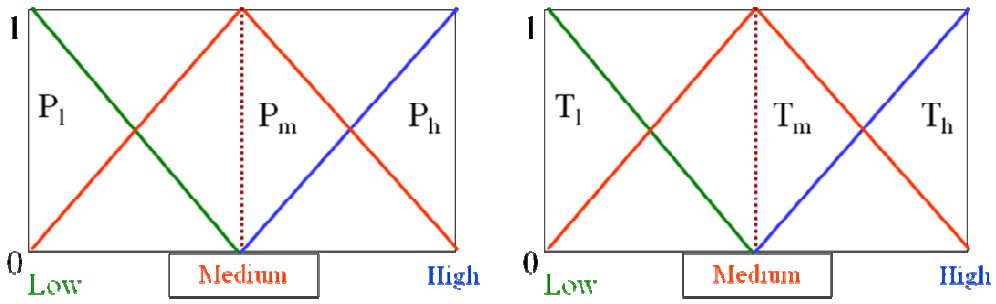


Figure 1.9.4: The membership functions of pressure and temperature.

The relation between the pressure and the temperature, and the valve position has been determined using nine rules, which can be represented as a matrix, the so-called fuzzy association matrix (FAM)

$$\text{FAM} =: \begin{array}{c} \\ P_l \\ P_m \\ P_h \end{array} \begin{array}{c} \begin{array}{ccc} T_l & T_m & T_h \end{array} \\ \begin{bmatrix} 90 & 90 & 50 \\ 90 & 50 & 50 \\ 50 & 10 & 10 \end{bmatrix} \end{array}$$

For example, suppose that $P^T = [0.8 \quad 0.2 \quad 0]$ and $T^T = [0.3 \quad 0.7 \quad 0]$. Then

$$\text{FAM} =: \begin{array}{c} \\ 0.8 \\ 0.2 \\ 0 \end{array} \begin{array}{c} \begin{array}{ccc} 0.3 & 0.7 & 0 \end{array} \\ \begin{bmatrix} 90 & 90 & 50 \\ 90 & 50 & 50 \\ 50 & 10 & 10 \end{bmatrix} \end{array}$$

Now using the min operation, the matrix of firing strengths (FSM) is defined

$$R_{P\cap T}=\begin{array}{c} \\ 0.8\\ 0.2\\ 0\end{array}\begin{array}{c}\begin{array}{ccc}0.3 & 0.7 & 0\end{array}\\ \begin{bmatrix}\min(0.8,0.3) & \min(0.8,0.7) & \min(0.8,0)\\ \min(0.2,0.3) & \min(0.2,0.7) & \min(0.2,0)\\ \min(0,0.3) & \min(0,0.7) & \min(0,0)\end{bmatrix}\end{array}$$

$$=\begin{array}{c} \\ 0.8\\ 0.2\\ 0\end{array}\begin{array}{c}\begin{array}{ccc}0.3 & 0.7 & 0\end{array}\\ \begin{bmatrix}0.3 & 0.7 & 0\\ 0.2 & 0.2 & 0\\ 0 & 0 & 0\end{bmatrix}\end{array}$$

The Mamdani centre of gravity defuzzification is obtained by using the FAM and FSM. This is achieved via the weighted sum of element by element multiplication divided by the sum of the firing strengths,

$$\frac{0.3\times 90+0.7\times 90+0.2\times 90+0.2\times 50}{1.4}=84$$

this means that the valve position should be 84% of its maximum. Zadeh calculated the fuzzy max defuzzification operation using the FAM and FSM. In this case, the highest output value is 0.7×90=63% which lowers than that obtained using the Mamdani approach. Note that all data and information is used in the Mamdani approach whilst only the maximum values are considered in Zadeh approach.

Example 4.1.1.9.4: In a dynamical system the following rules is described the relationship between the states (fuzzy inputs) and the controller values in which the error and change in error denote the first difference and second difference between the state variable and reference signal. Note that, in general, 'change in error' or second difference is derived using an approximation to the derivative of the error. Neg, NM, PM, Zero, and PB denote Negative, Negative Medium, Positive Medium and Positive Big, respectively. The fuzzy input-output rules are:

If *error* is Neg and *change in error* is Neg then control is NB.

If *error* is Neg and *change in error* is Zero then control is NM

If *error* is Neg and *change in error* is Pos then control is Zero

If *error* is Zero and *change in error* is Neg then control is NM

If *error* is Zero and *change in error* is Zero then control is Zero

If *error* is Zero and *change in error* is Pos then control is PM

If *error* is Pos and *change in error* is Neg then control is Zero

If *error* is Pos and *change in error* is Zero then control is PM

If *error* is Pos and *change in error* is Pos then control is PB

Table 1.3 summarises these relational input-output rules.

Error	Change in error	Control
Pos	Pos	PB
Pos	Zero	PM
Pos	Neg	Zero
Zero	Pos	PM
Zero	Zero	Zero
Zero	Neg	NM
Neg	Pos	Zero
Neg	Zero	NM
Neg	Neg	NB

Table 1.3: The rational rule to describe the fuzzy input-output relations.

The FAM for this example is

	Neg	Zero	Pos
Neg	NB	NM	Zero
Zero	NM	Zero	PM
Pos	Zero	PM	PB

Table 1.4: The tabular rule format for the FAM.

1.10 Takagi-Sugeno (T-S) fuzzy controllers

1.10.1 Open-loop systems

A nonlinear system may be described by a class of linear systems based on some operating conditions. In this case, a linear system shows the dynamical system behaviour when specific conditions are satisfied. The number of the linear systems depends on the structure, complexity of the nonlinear system. Suppose there are r linear systems corresponding to r different operating conditions. Assume that the ith plant rule is given by

If $x_1(t)$ is M_{i1} and … $x_n(t)$ is M_{in} then $\dot{x} = A_i x$

where $x \in R^n$ is the state, $A_i \in R^{n\times n}$, i indicates the number of rules, $1 \leq i \leq$r, and M_{ij} is input fuzzy sets.

Let μ_{ij} denote the membership function of the jth set in the ith rule. The firing strength (weight) of ith rule, w_i is first defined as

$$w_i(x) = \prod_{j=1}^{n} \mu_{ij}(x_j) \tag{1.10.1}$$

Then corresponding to each weight, a rate is defined

$$\alpha_i(x) = \frac{w_i(x)}{\sum_{i=1}^{r} w_i(x)} \tag{1.10.2}$$

Therefore,

$$\sum_{i=1}^{r} \alpha_i(x) = 1 \tag{1.10.3}$$

The Takagi-Sugeno fuzzy system is defined as

$$\dot{x} = \sum_{i=1}^{r} \alpha_i(x) A_i x \tag{1.10.4}$$

Note that the system (1.10.4) is nonlinear because $\alpha_i(x)$ are nonlinear terms. However, system (1.10.4) is stable if all the linear systems $\dot{x} = A_i x$ are stable. This fact can be stated as the following theorem which has been first presented by Tanaka and Sugeno (1992):

Theorem 3.2.1: The continuous-time Takagi-Sugeno (T-S) system) (1.10.4) is globally asymptotically stable if there exists a positive-definite matrix P such that

$$A_i^T P + P A_i < 0\,, \quad \forall i = 1, \cdots, r \tag{1.10.5}$$

where r is the number of the T-S rules.

1.10.2 Takagi-Sugeno fuzzy control: Closed-loop systems

In this case the rule plants are defined as

If $x_1(t)$ is M_{i1} and … $x_n(t)$ is M_{in} then $\dot{x} = A_i x + B_i u$

where $x \in R^n$ is the state, $A_i \in R^{n\times n}$, i indicates the number of rule, $1 \le i \le$r, and $u \in R^m$ is the control input and M_{ij} is input fuzzy sets. Define the weights and rates as in (1.10.1) and (1.10.2), respectively. Then the overall T-S fuzzy system is

$$\dot{x} = \sum_{i=1}^{r} \alpha_i(x)\left(A_i x + B_i u\right) \tag{1.10.6}$$

Now suppose that the control rules are given by

If $x_1(t)$ is M_{i1} and … $x_n(t)$ is M_{in} then $u = -K_i x$

Then the overall T-S fuzzy controller is

$$u = \sum_{j=1}^{r} \alpha_j(x) K_j x \tag{1.10.7}$$

Substituting (1.10.7) into (1.10.6) yields

$$\dot{x} = \sum_{i=1}^{r}\sum_{j=1}^{r} \alpha_i(x)\alpha_j(x)\left(A_i x - B_i K_j\right)x \tag{1.10.8}$$

which is the overall (T-S) fuzzy system.

The following theorem represents sufficient conditions for stability of the T-S system (1.10.8). Its proof is based on the Lyapunov directed method by employing an appropriate Lyapunov function (Wang *et al.*, 1996). Note that for all $i = 1, \ldots, r$ and $= 1, \ldots, r$, $\alpha_i(x)$ and $\alpha_j(x)$ are nonlinear functions of the sate while and in the system (1.10.4), terms $\alpha_i(x)$ $(i = 1, \ldots, r)$ are nonlinear. Therefore, in comparison with the system (1.10.4), further conditions are required to guarantee the stability of the T-S system (1.10.8).

Theorem 3.2.2: The continuous-time Takagi-Sugeno system is globally asymptotically stable if there exists a positive-definite matrix *P* such that

$$\begin{aligned} &(A_i - B_i K_i)^T P + P\left(A_i - B_i K_i\right) < 0\ , \quad \forall \mathrm{i} = 1, \cdots, \mathrm{r} \\ &G_{ij}^T P + P G_{ij} < 0\ , \qquad j < \mathrm{i} \le r \end{aligned} \tag{1.10.9}$$

where r is the number of the rules and $\text{for } j < \mathrm{i} \le r,$

$$G_{ij} = A_i - B_i K_j + A_j - B_j K_i$$

1.10.3 Discrete-time Takagi-Sugeno fuzzy system

Similar to time-continuous case, a set of appropriate fuzzy rules are first introduced and then based on these rules and the firing strengths (weights) of rule, the T-S fuzzy system is constructed.

Assume that the *ith plant rule* is given by

$$\text{If } x_1(t) \text{ is } M_{i1} \text{ and } \ldots x_n(t) \text{ is } M_{in} \text{ then } x(k+1) = A_i x(k) + B_i u(k)$$

where $x \in R^n$ is the state, $A_i \in R^{n\times n}$, i indicates the number of rule, $1 \leq i \leq r$, and M_{ij} denotes the input fuzzy sets. The weights and coefficient rates are defined as in (1.10.1) and (1.10.2) , then the aggregated T-S fuzzy system is

$$x(k+1) = \sum_{i=1}^{r} \alpha_i(k)\left(A_i x(k) + B_i u(k)\right) \tag{1.10.10}$$

Suppose that the control rules are given by

$$\text{If } x_1(t) \text{ is } M_{i1} \text{ and } \ldots x_n(t) \text{ is } M_{in} \text{ then } u(k) = -K_i x(k)$$

Then the overall T-S fuzzy controller is

$$u = -\sum_{i=1}^{r} \alpha_i(x) K_i x \tag{1.10.11}$$

Then the T-S fuzzy closed-loop system is

$$x(k+1) = \sum_{i=1}^{r}\sum_{j=1}^{r} \alpha_i(k)\alpha_j(x)\left(A_i x - B_i K_j\right) x(k) \tag{1.10.12}$$

Note that if the system (1.10.10) is an unforced system, i.e. $u = 0$, in this case, the system is $x(k+1) = \sum_{i=1}^{r} \alpha_i(k) A_i x(k)$ and it is globally asymptotically stable if there exists a positive definite solution P such that for all $i = 1, \cdots, r$, $A_i^T P A_i - P < 0$. The following theorem yields sufficient conditions for global asymptotic stability of the system (Wang *et al*., 1996).

Theorem 3.2.3: The discrete-time Takagi-Sugeno system (1.10.12) is globally asymptotically stable if there exists a positive-definite matrix P such that

$$\begin{aligned} (A_i - B_i K_i)^T P\left(A_i - B_i K_i\right) - P < 0 \ , \quad & \forall i = 1, \cdots, r \\ G_{ij}^T P G_{ij} - P < 0 \ , \quad & j < i \leq r \end{aligned}$$

where r is the number of the rules and

$$G_{ij} = A_i - B_i K_j + A_j - B_j K_i$$

1.10.4 Numerical example (an inverted pendulum)

Consider the inverted pendulum system

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{g\sin(x_1) - \frac{1}{2}amlx_2^2\sin(2x_1) - a\cos(x_1)u}{\frac{4}{3}l - aml\cos^2(x_1)}$$

where

x_1 : angle (in radian) of the pendulum from the vertical position

x_2 : angular velocity

g: gravitational acceleration constant, 9.81 m/s^2

m : mass of the pendulum (2Kg)

M : mass of the cart

l : length of the pendulum (0.8 m)

u : control input, which is the force (in Newton) applied to the cart

and $a = \frac{1}{m+M}$

Assume that two linear systems describe the dynamics of the inverted pendulum. Define the plant rules as follows:

- If y is around *0* then $\dot{x} = A_1x + B_1u$
- If y is around $\pm\pi/2$ then $\dot{x} = A_2x + B_2u$

where

$$A_1 = \begin{bmatrix} 0 & 1 \\ \frac{3g}{4l-3aml} & 0 \end{bmatrix},\ \mathrm{B}_1 = \begin{bmatrix} 0 \\ \frac{3a}{4l-3aml} \end{bmatrix},\ A_2 = \begin{bmatrix} 0 & 1 \\ \frac{6g}{\pi\left(4l-3amlb^2\right)} & 0 \end{bmatrix},\ \mathrm{B}_2 = \begin{bmatrix} 0 \\ \frac{3ab}{4l-3amlb^2} \end{bmatrix}$$

Suppose that the controllers $u = -K_1x$ and $u = -K_2x$ stabilise $\dot{x} = A_1x + B_1u$ and $\dot{x} = A_2x + B_2u$, respectively. These controllers can be designed using the traditional methods such as the pole placement and optimal control methods.

Then the controller rules are:

- If y is around *0* then $u = -K_1x$
- If y is around $\pm\pi/2$ then $u = -K_2x$

Let $K_1 = [\alpha \quad \beta]$, $K_2 = [\gamma \quad \eta]$. Define the membership functions as shown in Figure 1.10.1.

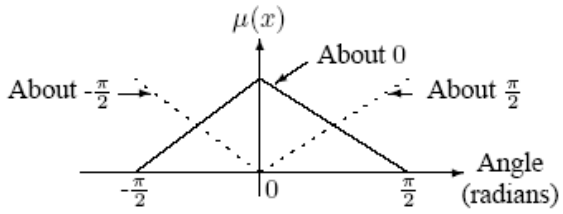


Figure 1.10.1: The membership functions for the inverted pendulum.

Therefore the membership functions are

The membership degrees around 0 and close to the points $\pm\pi/2$ is given by

$$\mu_1(x_1)=\begin{cases}\dfrac{2}{\pi}x_1+1 & \text{if } -\dfrac{\pi}{2}\le x_1<0\\ -\dfrac{2}{\pi}x_1+1 & \text{if } 0\le x_1<\dfrac{\pi}{2}\\ 0 & \text{otherwise}\end{cases}$$

and

$$\mu_2(x_1)=\begin{cases}-\dfrac{2}{\pi}x_1 & \text{if } -\dfrac{\pi}{2}\le x_1<0\\ \dfrac{2}{\pi}x_1 & \text{if } 0\le x_1<\dfrac{\pi}{2}\\ 1 & \text{otherwise}\end{cases}$$

respectively. Note that $\mu_1(x_1)+\mu_2(x_1)=1$. Therefore the definition of one membership functions is sufficient.

The T-S system is

$$\dot{x}=\left(\mu_1(x_1)A_1+\mu_2(x_1)A_2\right)x+\left(\mu_1(x_1)B_1+\mu_2(x_1)B_2\right)u$$

and the Takagi-Sugeno (T-S) fuzzy controller is

$$u=-\left(\mu_1(x_1)K_1+\mu_2(x_1)K_2\right)x$$

Substituting the T-S controller into the T-S system yields

$$\dot{x}=\left(\mu_1(x_1)A_1+\mu_2(x_1)A_2\right.$$
$$\left.-\left(\mu_1^2(x_1)B_1K_1+\mu_1(x_1)\mu_2(x_1)\left(B_1K_2+B_2K_1\right)+\mu_2^2(x_1)B_2K_2\right)\right)x$$

Let $K_1=\begin{bmatrix}\alpha & \beta\end{bmatrix}$, $K_2=\begin{bmatrix}\gamma & \eta\end{bmatrix}$. Now the T-S controller is

$$u = -\left(\mu_1(x_1)K_1 + \mu_2(x_1)K_2\right)x$$

$$= -\alpha x_1 - \beta x_2 + \begin{cases} \dfrac{-2}{\pi}(\alpha-\gamma)x_1^2 - \dfrac{2}{\pi}(\beta-\eta)x_1x_2 & \text{if } -\dfrac{\pi}{2} \le x_1 < 0 \\ \dfrac{2}{\pi}(\alpha-\gamma)x_1^2 + \dfrac{2}{\pi}(\beta-\eta)x_1x_2 & \text{if } 0 \le x_1 < \dfrac{\pi}{2} \\ 0 & \text{otherwise} \end{cases}$$

Note that $K_1 = \begin{bmatrix}\alpha & \beta\end{bmatrix}$, $K_2 = \begin{bmatrix}\gamma & \eta\end{bmatrix}$.

Exercise 1.10.1: Consider the inverted pendulum described in above **.**

(a) Using similar procedure as used for control law, simplify the T-S system.

(b) Simulate the responses of the system using MATLAB®.

(c) Check the asymptotic stability of the T-S system.

1.11 Fuzzy and PID controllers configurations

Many traditional controllers such PID, LQR, H_2/ H_∞ and sliding mode control may be designed based on the fuzzy logic rules. In this section, some configurations of fuzzy PID controllers are presented. Obviously, selection of each scenario depends on the plant structure and complexity and the desired performances. Some scenarios are shown in Figure 1.11.1.

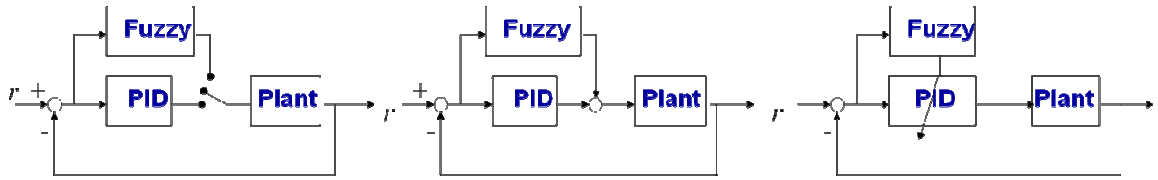


Figure 1.11.1: Fuzzy and PID controllers configurations

PI , PD and PID fuzzy control system

Figure 1.11.2-Figure 1.11.4 depicts the schematics PI, PD and PID fuzzy control system, respectively. In which $y(k)$ or $y(t)$ is the output, $r(k)$ or $r(t)$ is the reference signal and

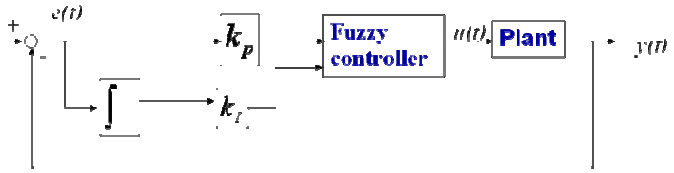


Figure 1.11.2: A PI fuzzy control system block diagram in which $u(t) = K_P e(t) + K_I \int e(t)dt$ **.**

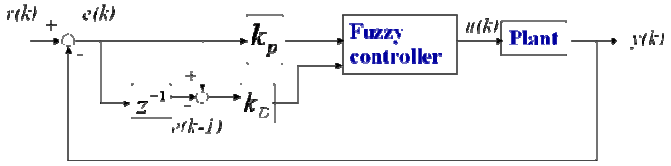


Figure 1.11.3: A PD fuzzy control system block diagram in which $e(k) = r(k) - y(k), \Delta e(k) = e(k) - e(k-1)$, $\Delta u(k) = u(k) - u(k-1)$ **and** $u(k) = K_P e(k) + K_D \Delta e(k)$.

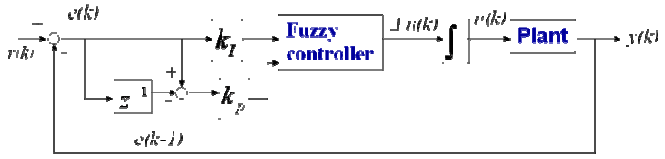


Figure 1.11.4: A PI fuzzy control system block diagram in which $e(k) = r(k) - y(k)$, $\Delta u(k) = u(k) - u(k-1) = K_p \Delta e(k) + K_I e(k)$ $u(k) = K_P e(k) + K_D \Delta e(k)$ **and** $\Delta e(k) = e(k) - e(k-1)$.

Table 1.5 a summary of the Mamdani and Takagi-Sugeno fuzzy approaches are given.

Mamdani-type fuzzy processing	Sugeno-type fuzzy processing
Consequent parts are fuzzy.	Consequent parts are singletons or mathematical functions of them.
Easily understandable by a human expert Rules simpler to formulate Proposed earlier and commonly used	More effective computationally More convenient in mathematical analysis and in systems analysis Guarantees the continuity of the output surface

Table 1.5: Comparison between the Mamdani and Takagi-Sugeno fuzzy processing at a glance.

1.12 Conclusions on fuzzy controllers

1.12.1 Takagi-Sugeno Fuzzy control method

(a) Design full-state feedback controllers for each linear model (using crisp control methods, e.g. LQR, H_2/ H_∞) using membership functions.

(b) It is possible to use quadratic Lyapunov functions to obtain sufficient conditions for nominal stability.

(c) Current methodology does not address stability robustness and performance robustness.

(d) Current methodology does not address output feedback requiring dynamic compensator designs**.**

1.12.2 Mamdani method

(a) Fuzzy feedback control methods (Mamdani) are suitable for trivial control problems requiring low accuracy (minimal performance).

(b) Easy to use.

(c) No models and specifications are properly given.

(d) Impossible to guarantee stability empirically.

(e) An ad-hoc approach to design leads to low-performance (low-gain, low-bandwidth) designs **.**

References

[1] S. Shao, Fuzzy self-organizing controller and its application for dynamic processes, *Fuzzy Sets and Systems* 26, 151-164, 1988.

[2] T.J. Procyk and E.H. Mandani, A linguistic self-organizing process controller. *Automatica,* 15, 15–30, 1979.

[3] M. Sugeno, An introductory survey of fuzzy control. *Inform. Sci*. 36, 59–83, 1985.

[4] E.H. Mamdani and A. Assilan, A case study on the application of fuzzy set theory to automatic control. In: *Proceedings of the IFAC Stochastic Control Symposium,* 1974.

[5] L.M. Sztandera, Experience augmented linguistic model for real industrial system. *Adv. Modelling Simulation,* 19(3), pp. 55–63, 1990.

[6] W. Pedrycz*, Fuzzy Control and Fuzzy Systems.* (2nd ed.),, Research Studies Press, John Wiley, Taunten, 1992.

[7] L. P. Holmblad and J. J. Ostergaard, Control of a cement kiln by fuzzy logic. In: fuzzy information and Decision Processes, M. M. Gupta and E. Sanchez, Eds. North-Holland, Amsterdam 1982, pp. 389-399.

[8] I.G. Umbers and P.J. King, An Analysis of Human Decision-Making in Cement Kiln Control and the Implications for Automation. *International Journal Man Machines Studies*, 12(1), 11-23, 1980.

[9] O. I. Franksen, Group representation of finite polyvalent logic. In A. Niemi, editor, *Proceedings 7th Triennial World Congress, Helsinki*, International Federation of Automatic Control (IFAC), Pergamon, 1978.

[10] S. Fukami, M. Mizumoto, and K. Tanaka. Some considerations of fuzzy conditional inference, *Fuzzy Sets and Systems*, 4:243–273, 1980.

[11] Mai Gerhke, Carol L. Walker, and Elbert A. Walker. Normal forms and truth tables for fuzzy logics, *Fuzzy Sets and Systems*, 138, 25–51, 2003.

[12] J.-S. R. Jang, C.-T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing*, New Jersy: Prentice Hall, Upper Saddle River, NJ, USA, 1997.

[13] J. Jantzen. Array approach to fuzzy logic.Fuzzy Sets and Systems, 70:359–370, 1995.

[14] J. B. Kiszka, M. E. Kochanska, and D. S. Sliwinska, The influence of some fuzzy implication operators on the accuracy of a fuzzy model, *Fuzzy Sets and Systems*, 15:(Part1), 111–128; (Part 2), 223 – 240, 1985.

[15] C. C. Lee. Fuzzy logic in control systems: Fuzzy logic controller. IEEE Trans. Systems, *Man & Cybernetics,* 20(2), 404–435, 1990.

[16] E. H. Mamdani. Application of fuzzy logic to approximate reasoning using linguistic synthesis, IEEE Transactions on Computers, C-26(12), 1182–1191, 1977.

[17] M. Mizumoto, S. Fukami, and K. Tanaka, Some methods of fuzzy reasoning. In Gupta, Ragade, and Yager, editors, *Advances in Fuzzy Set Theory Applications*. New York: North-Holland, 1979.

[18] R. R. Stoll, Set Theory and Logic. Dover Publications, New York: Dover Edition, 1979 (org 1963).

[19] F. Wenstop, Quantitative analysis with linguistic values, *Fuzzy Sets and Systems,* 4(2), 99–115, 1980.

[20] L. A. Zadeh, Fuzzy sets, *Inf. and control,* 8, 338–353, 1965.

[21] L. A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes. IEEE Trans. Systems, *Man & Cybernetics,* 1, 28–44, 1973.

[22] L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning. Information Sciences, 8:43–80, 1975.

[23] L. A. Zadeh, Fuzzy logic, *IEEE Computer*, 21(4), 83–93, 1988. [24] H.-J. Zimmermann, Fuzzy set theory and its application, Boston: Kluwer, Second edition, 1993.

[25] L.A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, IEEE Trans. on systems, Man and Cyber. SMC-3, 28-44,1973.

[26]H. B. Verbuggen, Henk B. Verbruggen (Editors), Fuzzy Algorithms For Control, Kluwer Academic Publishers , 1999

[27] E.H. Mamdani, Application of Fuzzy Algorithms for Control of Simple Dynamic Plant, Proceedings of IEEE, 121(12), 1974, 1585-1588.

2 Neural Networks

2.1 Background and initial innovations

The name neural network is derived from the human brain, which is extremely complex, nonlinear and parallel such that it passes and receives information at the same time. A neuron is responsible for transferring information including passing and receiving information from and to other brain cells. A brain cell (the so-called pyramidal cell) and three interconnected artificial neurons are depicted in Figure 2.1.1 (Haykin, 2009).

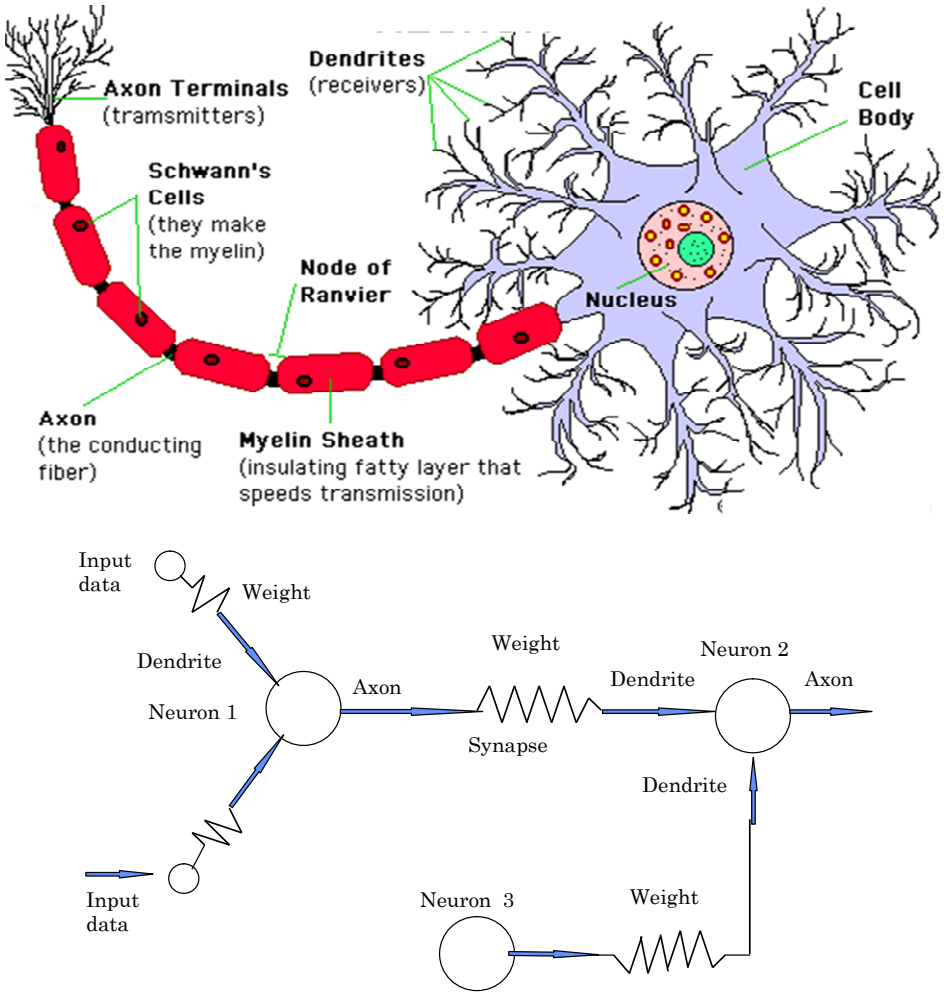


Figure 2.1.1: A brain cell (upper figure) and schematics of the interconnected artificial neurons with the input data and weights (lower figure).

The brain is continuously trained during an individual's life time. For example, a child learns how to speak or do a specific work by learning from the past and improve until the task can be done. The concept of learning and intelligence and how intelligence relates to learning may change with time and new technologies

2.2 History of neural networks

In 1943 McCulloch and Pitts presented a simple two–state binary threshold type of neuron. Their neural network was a simple network in which two cases were compared. In 1949 Hebb proposed the Hebbian learning model. A developed form of this type of learning model is still used. In 1956 Rochester and his colleagues reported the first computer simulation for Hebb's model, and this date can be considered as the official beginning of Computational Artificial Intelligence. During 1958 and 1961 Rosenblatt developed many networks that he called 'perceptrons'. In 1960 Widrow and Hoff developed a supervised learning approach using the least mean square method. In 1969: Minsky and Papert claimed that perceptrons were too limited in their computation and published a mathematical analysis book about the computational capabilities and limitations of perceptrons. The research on neural networks came to a halt because of diverse comments and there was a little effort on their development until 1972 when Kohonen and Anderson investigated an associative memory for a model in the form of a Hebbian. In 1974 Werbos discovered the backpropagation learning algorithm which was an important step in the development of neural networks. In 1985 Parker developed a second backpropagation learning algorithm. Since then many backpropagation implementations have been proposed, making it one of the most popular and powerful training method which gives a global solution. . In 1982 Hopfield proposed a new type of neural network which is called the Hopfield network. During the period 1969 until 1988, Fukushima and Myaki (1982) introduced a specific type of neural network which were termed the cognition and neocognitron networks. Many types of networks and methods have since then been proposed or developed during last two decades, see references at the end of this chapter.

2.3 A simple artificial neural network

Let $x_1, x_2 \cdots, x_n$ e the available inputs and $y_1^d, y_2^d, \cdots, y_m^d$ be the desired outputs of the network. A set of weights w_1, w_2, w_n can be selected to relate the inputs to the outputs . The linear combination of inputs and weights is called net, that is:

$$\text{net} = \sum_{i=1}^{n} w_i x_i$$

To set up a neural network a function, referred to as activation function, is used. This function is very important because the output of the network depends on its type.

. The network output is then defined as:

$$f(\text{net}) = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

The network must be tuned such that the error between the desired outputs and the network outputs are sufficiently small, e.g.

$$\|y - y_d\| = \left(\sum_{i=1}^{m} \left(y_i - y_i^d \right)^2 \right)^{\frac{1}{2}}$$

where $y_d = \left[y_1^d, y_2^d, \cdots, y_m^d \right]$ are the desired output which are also named as targets $t = \left[t_1, t_2, \cdots, t_m \right]$.

Figure 2.3.1 illustrates such a simple neural network with linear combination of inputs and weights transformed by the activation function in the network outputs $y_1, y_2, \ldots y_i, \ldots, y_m$.

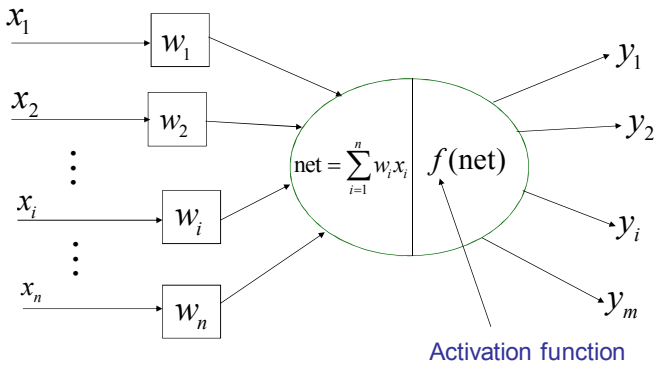


Figure 2.3.1: A simple neural network.

Example 2.3.1:
Assume that a network has three inputs $x_1 = 1,\ x_2 = 3,\ x_3 = 4$ and associated weights $w_1 = 0.3,\ w_2 = 0.5,\ w_3 = 0.1$. Find the network output for the following activation functions:

(a) $f(x) = x;$

(b) $f(x) = \dfrac{1}{1 + e^{-2x}}$

In this example

$$\begin{aligned} \text{net} &= x_1 w_1 + x_2 w_2 + x_3 w_3 \\ &= 0.3 \times 1 + 0.5 \times 3 + 0.1 \times 4 \\ &= 2.2 \end{aligned}$$

(a) Assume that $f(x) = x$, then $f(\text{net}) = f(2.2) = 2.2$.

(b) If $f(x) = \dfrac{1}{1 + e^{-2x}}$, then $f(\text{net}) = f(2.2) = \dfrac{1}{1 + e^{-2 \times 2.2}}$.

To improve the network quality to obtain the desired output, a constant (vector) is included to the net function, therefore the net function is defined as: $\text{net} = \sum_{i=1}^{n} w_i x_i + b$. This constant b is called, bias, see Figure 2.3.2.

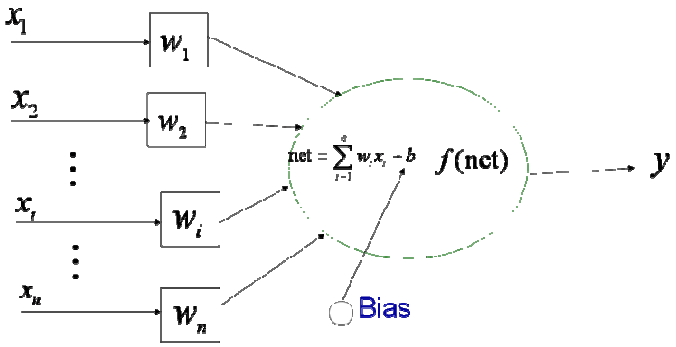


Figure 2.3.2: A simple neural network with bias.

The network output depends on the selection of the weights and the activation function. Typical activation functions are in the form of linear, step, sigmoid, symmetric sigmoid, hyperbolic tangent, Gaussian. The MATLAB® commands heaviside, hardlim, purelin, logsig, tansig and radbas are used for these activation functions, respectively. The functions sigmoid and symmetric sigmoid are known as logistic and symmetric logistic functions, respectively. Figure 2.3.3 shows the typical activation functions.

Example 5.2.2.3.2: Using MATLAB® obtain and sketch the graphs of the following functions:

- u=-5:0.1:5, plot(u,heaviside(u))
- u=-5:0.1:5, plot(u,hardlim(u))
- u=-5:0.1:5, plot(u,purelin(u))
- u=-5:0.1:5, plot(u,logsig(u))
- u=-5:0.1:5, plot(u, tansig(u))
- u=-5:0.1:5, plot(u,radbas(u))

The logistic function is widely employed as the activation function of hidden and output layers of neural networks, and in backpropagation algorithms. Its derivative is used to calculate the rates of change of the weights; the following exercise illustrates the important relationship between the logistic function and its derivative.

Excersie 2.3.14 **Let** $s(x) = \dfrac{1}{1+e^{-\alpha x}}$ **. Show that** $\dfrac{ds}{dx} = \alpha(1-s)s$ **.**

The typical activation functions are linear, step, logistic, hyperbolic tangent, Gaussian function. See Figure 2.3.3

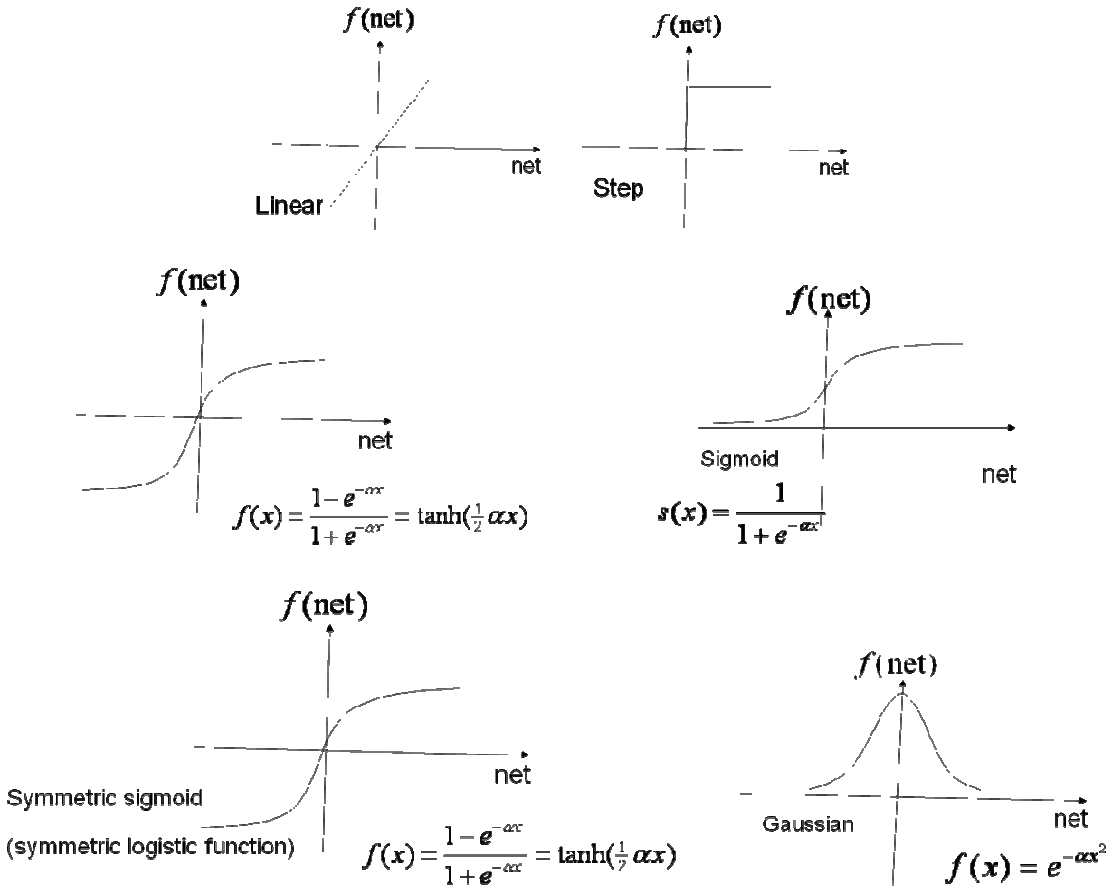


Figure 2.3.3: Typical activation functions.

A basic artificial neural network has only a single layer, comprising of the output layer is illustrated in Figure 2.3.4. The input layer is not included as a layer because this must always be available. In fact, the input layer provides the input data which are used to set up the neural network such that the output from the network can be tuned to match the desired output (targets).

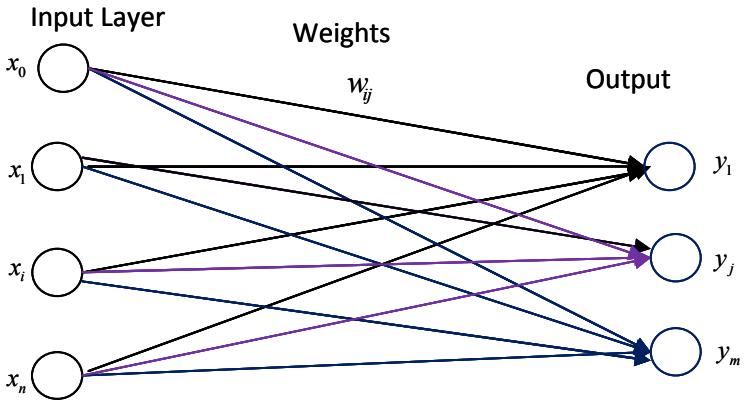


Figure 2.3.4: A simple neural network without activation function.

Example 2.2.3.3: Find the output of a neural network with input $x = \begin{bmatrix} 4 \\ 1 \\ 2 \end{bmatrix}$, weight matrix $w = \begin{bmatrix} 0.9 & -0.4 & 0.9 \\ 0.3 & -0.5 & -0.4 \end{bmatrix}$, bias $b = \begin{bmatrix} 0.9 \\ -1.5 \end{bmatrix}$ when the activation function is

- a linear function;
- logistic function (sigmoid);
- Hyperbolic tangent.
- The output of the neural network is

$$\begin{aligned} y &= f(wx + b) \\ &= f\left(\begin{bmatrix} 0.9 & -0.4 & 0.9 \\ 0.3 & -0.5 & -0.4 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.9 \\ -1.5 \end{bmatrix} \right) \end{aligned}$$

where f is the activation function. .

Solution:

Since

$$\begin{bmatrix} 0.9 & -0.4 & 0.9 \\ 0.3 & -0.5 & -0.4 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0.9 \\ -1.5 \end{bmatrix} = \begin{bmatrix} 5.9 \\ -1.6 \end{bmatrix}$$

Then the output is

- $\begin{bmatrix} 5.900 \\ -1.600 \end{bmatrix}$ if f is linear.
- $\begin{bmatrix} 0.9973 \\ 0.1680 \end{bmatrix}$ if f is the logistic function.
- $\begin{bmatrix} 0.9973 \\ 0.1680 \end{bmatrix}$ if f is the logistic function.

2.3.1 Perceptron

The net function $\sum_{i=1}^{n} w_i x_i$ is also known as the threshold function (Threshold logic unit). The threshold function output (Threshold logic unit output) is

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & \text{if } \sum_{i=1}^{n} w_i x < T \end{cases}$$

where the parameter T is a threshold. In the early literature, the threshold function output was known as the McCulloch-Pitts neuron. To make the system easier to analyse, 1 and -1 is used for the threshold function output (threshold logic unit output), i.e.

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq T \\ -1 & \text{if } \sum_{i=1}^{n} w_i x < T \end{cases} \qquad (2.3.1)$$

The threshold function output may be written as

$$y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i \geq 0 \\ -1 & \text{if } \sum_{i=0}^{n} w_i x < 0 \end{cases}$$

in which $w_0 = T$, $x_0 = -1$ and the target values are 1 and -1. If the target value and the actual output are the same the network it is said to work well, otherwise, the weights should be adjusted.

Assume that the target values is -1 $t = 1$ and the output is $y = -1$, i.e. $\sum_{i=0}^{n} w_i x < 0$. The weights should be updated such that $\sum_{i=0}^{n} w_i x \geq 0$. The following statements are considered to adjust the weights:

- If $x_i < 0$ it is required to make w_i smaller; if $x_i > 0$ it is required make w_i larger.

Note that the equation $\sum_{i=0}^{n} w_i x_i = 0$ divides the output in two different classes, class 1 and class -1 depending on whether the net function $\sum_{i=0}^{n} w_i x_i$ is nonnegative or negative. See Figure 2.3.5 in which w_0 denotes for the threshold *T*.

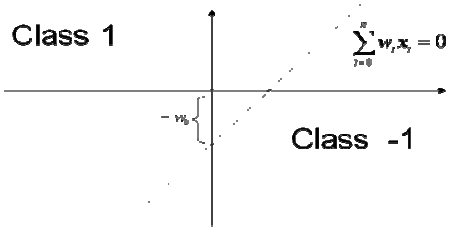


Figure 2.3.5: -1 and 1 classes.

2.3.2 Total output error and Delta Rule

The threshold function output is given by **Błąd! Nie można odnaleźć źródła odwołania.**), however if the target is 1 (or -1) , it is necessary to find appropriate weights such that the network output is 1 (or -1). If the target is different from the network output with a set of initial weights, the weights must be tuned. The Delta rule is a method to find a set of suitable weights.

Therefore, it is important to tune a network to obtain the target for available data. To ensure that the network has been tuned well, the difference between the network output and target must be reasonably small, ideally zero.

The target t and the network output y are (nearly) identical if the total output error is E has a minimum value, where

$$E = \|y - t\| = \frac{1}{2}\sum_{i=0}^{n}\left(t_i - y_i\right)^2 \qquad (2.3.2)$$

If the error is non zero, the weights must be updated. To find the rate of weight changes, the partial derivative E with respect to the weights is required such that:

$$\begin{aligned}\frac{\partial E}{\partial w_j} &= \frac{\partial}{\partial w_j}\left(\frac{1}{2}\sum_{i=0}^{n}\left(t_i - y_i\right)^2\right)\\ &= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial w_j}\\ &= -\left(t_j - y_j\right)x_j\end{aligned} \qquad (2.3.3)$$

The rate of change of E with respect w_j is therefore $\left(y_j - t_j\right)x_j$. To avoid premature convergence to an inappropriate set of weights it is necessary to reduce the rate of change of the error by weighting it to enforce a gradual change. The resulting rate of changes is defined as $\Delta w_j = \alpha\left(t_j - y_j\right)x_j$ where α is a scalar, termed learning rate, which is positive or negative depending on the values of y_j and t_j. The updated weights are $\tilde{w}_i = w_i + \Delta w_i$ where $\Delta w_j = \alpha\left(t_j - y_j\right)x_j$ and $\alpha$is normally selected to be a small value.

Example 2.3.4: Consider the inputs $x = \begin{bmatrix}-1 & -0.5 & 2 & 0.3\end{bmatrix}^T$, Select the weights $w = \begin{bmatrix}0.7 & 1 & -1 & 0.4\end{bmatrix}^T$. Then

$$\begin{aligned}\sum_{i=0}^{3} w_i x_i &= (-1)(0.7) + (-0.5)(1) + (2)(-1) + (0.3)(0.4)\\ &= \text{-}3.08\end{aligned}$$

Since $\sum_{i=0}^{3} w_i x_i < 0$, therefore the output is $y = -1$, see . If the target (the desired output) is assumed to be $t = -1$, no changes to the weights are required. If (2.3.1), by contrast, if the target is $t = 1$ the weights must be updated using the so called Delta Rule

$$\begin{aligned}\Delta w_i &= \alpha\left(t_i - y_i\right)x_i \\ &= \alpha\left(1-(-1)\right)x_i \\ &= 2\alpha x_i\end{aligned}$$

The updated weights for $\alpha = 0.3$:

$$\begin{aligned}\tilde{w} &= w + \Delta w = w + 2\alpha x \\ &= \begin{bmatrix}0.7 & 1 & -1 & 0.4\end{bmatrix}^T + \begin{bmatrix}-0.6 & -0.3 & 1.2 & 0.18\end{bmatrix}^T \\ &= \begin{bmatrix}0.1 & 0.7 & 0.2 & 0.58\end{bmatrix}^T\end{aligned}$$

Using the new set of weight the network output becomes $y = 1$ because

$$\begin{aligned}\sum_{i=0}^{3}\tilde{w}_i x_i &= (0.1)(-1) + (0.7)(-0.5) + (0.2)(2) + (0.58)(0.3) \\ &= 0.124\end{aligned}$$

For tuning the network, the following perceptron learning rules may be required:

- If the output is 1 and the target is 1 or if the output is -1 and the target -1, no change to weights are required.
- If the output is -1 and the target is 1, an increment on the weight values corresponding to all active inputs is required.
- If the output is 1 and the target -1, a decrement on the weight values corresponding to all active inputs is required.

Therefore, to update the weights, first the learning rate coefficient $\alpha > 0$ is selected then the rate of the weight changes are calculated as follows

$$\Delta w_i = \begin{cases}\alpha\left(t_j - y_j\right)x_i & \text{if the output is -1 and the target is 1,} \\ -\alpha\left(t_j - y_j\right)x_i & \text{if the output is 1 and the target is -1,} \\ 0 & \text{if the output and target are the same.}\end{cases}$$

The perceptron learning for the logical operation 'AND' is defined as

inputs		Targets
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.1. AND logic operation.

In the above example four epochs (cycles) are required to obtain the final weights. During each epoch (cycle), all inputs are used and a final set of weights is obtained. The process is completed when $t_j - y_j = 0$ for all $j = 1, 2, 3, 4,$. Note that the number of epochs depends on the selection of the initial weights and the learning rate.

Example 2.3.5: Consider the 'AND' problem. Select $\alpha = 0.1$ $T = 0.3$ $w_1 = 0.2$ and $w_2 = -0.3$. Then the final weights, $w_1 = 0.1$ and $w_2 = 0.1$ are obtained after five epochs. The full of calculations are summarised in
Table **2.2**.

To show that how the weights are calculated, the following steps are required to obtain the final weights after completion of the first epoch and the

Table 2.2 presents only the answers:

First step: The first inputs are initially considered, i.e. $x_1 = 0$ and $x_2 = 0$. Then the net is calculated as follows: net $= w_1x_1 + w_2x_2 - T = -0.3$. Since net<0 then y=0, and $e = t - y = 0 - 0$=0. Therefore, $\Delta w_1 = \alpha(t - y)x_1 = 0$ and $\Delta w_2 = \alpha(t - y)x_2 = 0$, i.e. no changes in weights are required.

Second step: The second inputs are now considered, i.e. $x_1 = 0$ and $x_2 = 1$. Then the net output is given by net$= w_1x_1 + w_2x_2 - T = 0.4 - 0.3 = 0.1$. Since net>0 then y=1, and $e = t - y = 0 - 1 = -1$. Therefore, $\Delta w_1 = \alpha(t - y)x_1 = 0$ and $\Delta w_2 = \alpha(t - y)x_2 = 0.2(-1)1 = -0.2$. Tthe new weights are therefore $w_1 + \Delta w_1 = w_1 + 0 = w_1$ and $w_2 + \Delta w_2 = w_2 + 0.2 = 0.2$.

Third step: The third inputs are considered, i.e. $x_1 = 1$ and $x_2 = 0$. Then net $= w_1x_1 + w_2x_2 - T = 0.1 - 0.3 = -0.2$. Since net<0 then y=0, and $e = t - y = 0 - 0 = 0$. Therefore, no changes in weights are required.

Fourth step: The fourth inputs are considered, i.e. $x_1 = 1$ and $x_2 = 1$. Then net $= w_1x_1 + w_2x_2 - T = 1 \times 0.1 + 1 \times 0.2 - 0.3 = -0.1$. Since net,0 then y=0, and $e = t - y = 0 - 0 = -1$. Hence, no change required in weights and the final weight at the completion of the first are $w_1 = 0.1$ and $w_2 = 0.2$.

The above procedure must be carried out to obtain zero error for all four inputs. At this stage, the final weights are found because when the error is zero for all inputs, changes in weights are zero and the network is tuned. Note that the final weights depends on the selections of the initial weights which means the network may be tuned using different weights. This implies that the tuned network may not be unique.

α=0.2 *T*=0.3	Inputs		Targets	Initial weights		net	Output	Error	The rate of updating		Final weights	
Epoch	x_1	x_2	t	w_1	w_2	$\sum_{i=0}^{2} w_i x_i$	y	t-y	$\alpha(t-y)x$; $\alpha\,(t-y)\times[x_1\ x_2]$		w_1	w_2
1	0	0	0	0.1	0.4	-0.3	0	0	0	0	0.1	0.4
	0	1	0	0.1	0.4	0.1	1	-1	0	-0.2	0.1	0.2
	1	0	0	0.1	0.2	-0.2	0	0	0	0	0.1	0.2
	1	1	1	0.1	0.2	0	1	0	0	0	0.1	0.2
2	0	0	0	0.1	0.2	-0.3	0	0	0	0	0.1	0.2
	0	1	0	0.1	0.2	-0.1	0	0	0	0	0.1	0.2
	1	0	0	0.2	0.2	0	1	-1	- 0.2	0	0	0.2
	1	1	1	0	0.2	-0.1	0	1	0	0	0.2	0.4
3	0	0	0	0.2	0.4	-0.3	0	0	0	0	0.2	0.4
	0	1	0	0.2	0.4	0.1	1	-1	0	-0.2	0.2	0.2
	1	0	0	0.2	0.2	-0.1	0	0	0	0	0.2	0.2
	1	1	1	0.2	0.2	0.1	1	0	0	0	0.2	0.2
4	0	0	0	0.2	0.2	0	0	0	0	0	0.2	0.2
	0	1	0	0.2	0.2	-0.1	0	0	0	0	0.2	0.2
	1	0	0	0.2	0.2	-0.1	0	0	0	0	0.2	0.2
	1	1	1	0.2	0.2	0.1	1	0	0	0	0.2	0.2

Table 2.2: Summary of the calculation for obtaining the final weights for the 'AND' problem.

Exercise 2.3.1: Let $\alpha = 0.1,\ \ T = 0.3$. Using the perceptron learning (2.3.4), find the final updated weights for AND and OR problems. Summarise the calculation using Table 2.3.

α=0.1 *T*=0.3	Inputs		Targets	Initial weights		net	Output	Erro r	The rate of updating		Final weights	
Epoch	x_1	x_2	t	w_1	w_2	$\sum_{i=0}^{2} w_i x_i$	y	t-y	$\alpha(t-y)x$; $\alpha\,(t-y)\times[x_1\ x_2]$		w_1	w_2
1	0	0	0	0.2	-0.3	-0.3	0	0	0	0	0.1	-0.3
	0	1	1	0.2	-0.3	0	1	1	0	0.1	0.1	-0.2
	1	0	1	0.1	-0.2	-0.2	0	1	0.1	0	0.2	-0.2
	1	1	1	0.2	-0.2	-0.3	0	1	0.1	0.1	0.2	-0.1
2	0	0	0	0.2	-0.1							
	0	1	0									
	1	0	0									

	1	1	1								
3	0 0 1 1	0 1 0 1	0 0 0 1								
4	0 0 1 1	0 1 0 1	0 0 0 1								
5	0 0 1 1	0 1 0 1	0 0 0 1								

Table 2.3: Summary of the calculation of (a) AND, b) OR.

2.3.3 Updating weights for a general case: The Delta Rule

Suppose that g is the activation function and the output is

$$y = g(v) = g(\sum_{i=1}^{n} w_i x_i)$$

where $v = \sum_{i=1}^{n} w_i x_i$.

To obtain the appropriate weights, it is required to minimise the sum of squared errors

$$S(w) = \frac{1}{2}\sum_{j}\left(t_j - y_j\right)^2$$

The derivative of $S(w)$ with respect to y_j;

$$\frac{\partial S}{\partial w_i} = -\sum_{j}\left(t_j - y_j\right)\frac{\partial y_j}{\partial w_i} \qquad 2.3.5$$

On the other hand,

$$\frac{\partial y_j}{\partial w_i} = \frac{dy_j}{dv}\frac{\partial v}{\partial w_i} = g'(v)x_i \qquad 2.3.6$$

Substituting 2.3.6) into 2.3.5) yields

$$\frac{\partial S}{\partial w_i} = -\sum_{j}\left(t_j - y_j\right)g'(v)x_i$$

The minimum value of $S(w)$ is obtained when, for all i, $\frac{\partial S}{\partial w_i} = 0$. Since this condition may not be exactly satisfied, to minimise the sum of squared errors the negative gradient direction must be considered, therefore the rate changes to the weights are

$$\Delta w_i = \beta \sum_j \left(t_j - y_j\right) g'(v) x_i$$

where β is the leaning rate update and should be appropriately selected. The update weights are $\tilde{w}_i = w_i + \Delta w_i$.

2.3.4 Summary

Consider a neural network model. For minimising the error function, the following steps are considered:

1. Select initial values (say, random) for the model parameters.
2. Calculate the gradient of the error function with respect to each model parameter.
3. Change the model parameters in the direction of the greatest rate of decrease of the error. Repeat steps 2 and 3 until the gradient of the error function becomes sufficiently close to zero. See Figure 2.3.6.

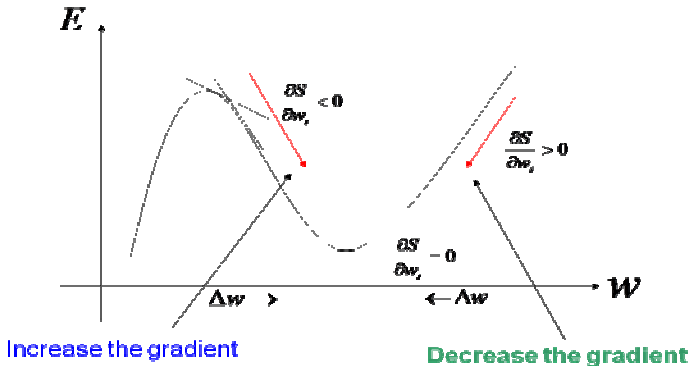


Figure 2.3.6: The gradient direction.

2.4 Neural network learning

Neural networks use the human-like technique of learning by example to resolve problems. Learning enables a neural network to change and configure its output behaviour based on changes in the inputs. Neural networks are configured for specific applications through a learning process called training.

2.4.1 Machine learning

Machine learning describes the development of algorithms and techniques that allow computers to learn. Machine learning uses one or more of the following methods:

(a) Computer programs

(b) Optimisation

(c) Classification: grouping patterns into classes

(d) Associative memory

(e) Regression

There are links between machine learning and statistics. During a learning process a statistical methods may be used to configure the network.

Two types of learning are: inductive and deductive. Inductive machine learning methods create an algorithm using massive data sets for obtaining rules and patterns. Inductive learning generalises the knowledge and facts by use of examples. Deductive learning extracts the new knowledge and facts by developing the existing knowledge and facts. When general facts and propositions are extracted from existing facts and rules, normally using a mathematical process, the learning process is deductive. By contrast, inductive learning derives facts from employing data and examples and they do not obtain using mathematical procedures. Therefore, inductive learning is used when a set of data is available and by using them a general case is proposed.

2.4.2 Learning strategies

Typical methods for basic learning strategies are Supervised, Reinforcement and Unsupervised. Supervised learning methods include the following methods: Delta rule, backpropagation, Hebbian, adaptive linear neural element, adaptive resonant theory, probabilistic and stochastic, radial basis function neural networks. Learning automata is reinforcement type of learning while competitive Hebbian, neocognitron, adaptive resonant theory can be classified as unsupervised learning. Some of types of network can be supervised or unsupervised based on their architectures. Figure 2.4.1 shows a typical supervised learning cycle.

There are a number of categories of neural network types including:

(a) Single layer feedforward. These types of neural networks include adaptive linear neural element, Hopfield, perceptron and associative memories.

(b) Multilayer feedforward. Multi-adaptive linear neural element, radial bias function (RBF) and neocognitron are multilayer feedforward neural networks.

(c) Recurrent. Adaptive resonant theory, Hopfield and probabilistic are recurrent neural networks. See the references, for example Haykin (2009) for details

2.4.3 Machine learning algorithms

A training set is formed by combining together all, or more appropriately, a subset of the known input and output patterns. In fact all data used as inputs and the outputs of a network are selected from the training set. In supervised learning methods, an algorithm generates a function which maps inputs to desired outputs. A subset of the training set which has not been used for network training may be used for testing the network, see Figure 2.4.1.

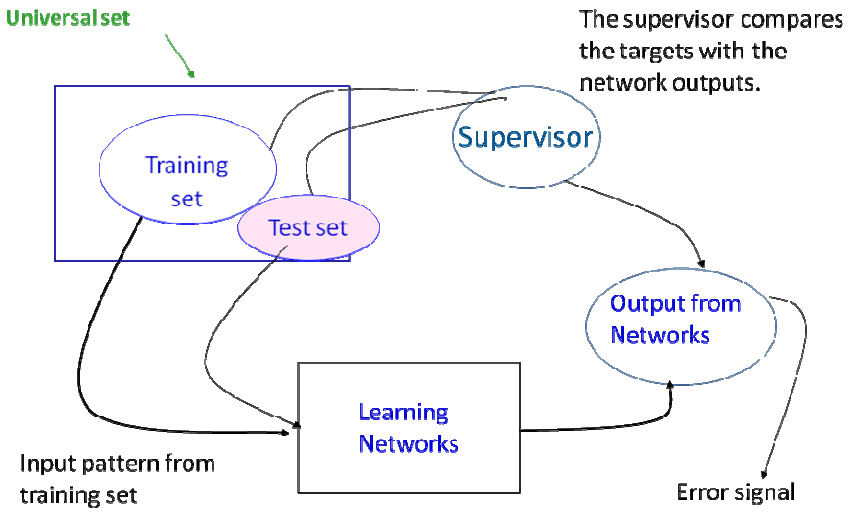


Figure 2.4.1: Supervised association learning cycle.

2.4.4 Testing a network

Testing a neural network is an important issue, because it ensures that the network works well for a wide range of data including unseen data within receptive field. In general, a network is tested using different data including:

- selected sets (as presented patterns to the networks during learning) from the training set to check if the network works well.
- unseen data (from outside of the training set other data has not been used during the training).
- a new pattern by using random noise (Random inputs), or a slightly different pattern. See Figure 2.4.1.

If all the pairs of input and output patterns have not been used, for example assume that there are 200 samples of input and associated output patterns and the network is trained with 100 samples, then the network test may be carried out with the remaining samples and specific patterns as unseen data.

2.4.5 Accuracy measurement of a network

In supervised learning the desired output pattern is known. To measure the accuracy of the network:

(a) Compute the network output pattern *y* and compare with the desired output y_d (target *t*).

(b) Define an error function and compute it: e.g.

$$\|y - y_d\| = \left(\sum_{i=1}^{m} \left(y(i) - y_d(i) \right)^2 \right)^{\frac{1}{2}}$$

There are many possible error functions. The network is said to work perfectly well if the total output error has a minimum value.

2.4.6 Limitations of single layer perceptrons

A simple perceptron, i.e. single perceptron, works well if the learning cases are linearly separable. This means, for example, that the inputs must lie above or the below a straight line $\sum_{i=0}^{n} w_i x_i = 0$ dividing two different possible outputs. The inputs are above of the line if $\sum_{i=0}^{n} w_i x_i > 0$ and under the line if $\sum_{i=0}^{n} w_i x_i < 0$ and these are termed class 1 and -1, respectively (see **Błąd! Nie można odnaleźć źródła odwołania.**).

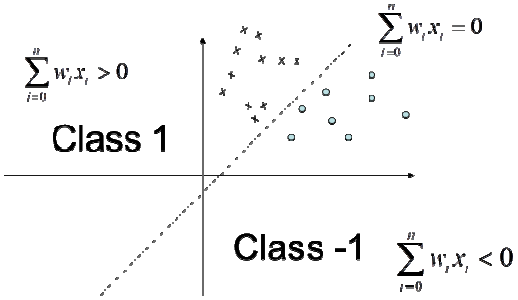


Figure 2.4.2: Classes 1 and -1.

Example 2.4.1:

Exclusive OR (XOR) is defined in mathematical logic as in a truth table. Exclusive OR operation for the statements p and q is denoted as p XOR q or p⊕q are presented in Table 2.4.

The equivalent of XOR operator is

$$\begin{aligned} p \oplus q &= (p \wedge \neg q) \vee (\neg p \wedge q) \\ &= (p \vee q) \wedge (\neg p \vee \neg q) \\ &= (p \vee q) \wedge \neg(p \wedge q) \end{aligned}$$

Table 2.4: The XOR truth table.

p	q	p XOR q
F	F	F
F	T	T
T	F	T
T	T	F

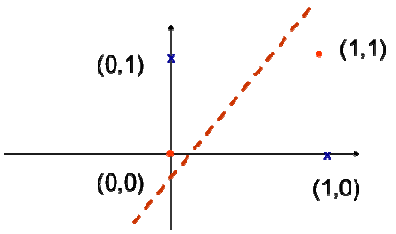


Figure 2.4.3: The XOR problem.

The exclusive OR (XOR) problem may also be written as in Table 2.5 shows that XOR inputs are not separable into two classes (i.e., the ON and OFF classes) by a single straight line. This means that it is impossible to solve the XOR problem using a single perceptron. However, it is possible to solve the problem using more than one line as illustrated in Figure 2.4.4

Table 2.5: The inputs and the outputs of XOR

Input		Output
0	0	0
0	1	1
1	0	1
1	1	0

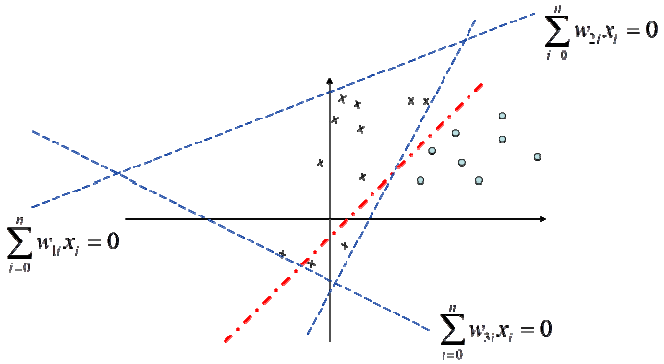


Figure 2.4.4: Separation of the inputs using three lines.

Exercise 2.4.1:
Let $\alpha = 0.1,\ T = 0.3$. Using the perceptron learning (2.4.1), find the final updated weights for XOR problem. Summarise the calculation using Table 2.3**.**

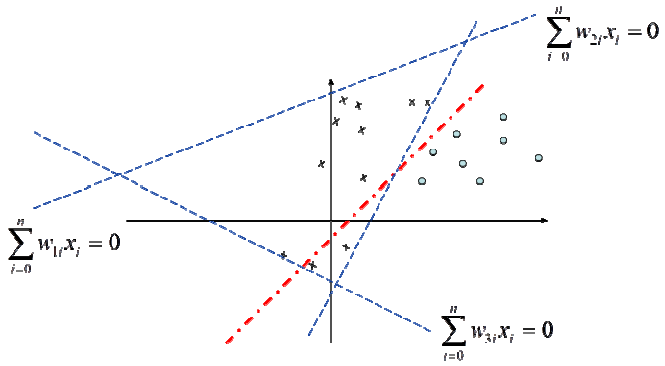


Figure 2.4.5: Separation of the inputs using three lines.

.

2.4.7 Solution region

To design a network a set of initial weights are selected. These weights must be updated using an algorithm to obtain the final weights which minimise the sum of the norm errors between the network output and targets. The final weights are within a region, the so-called the solution region. Therefore, the weights during updating have the following properties:

- Every set of weights corresponds to a net which separates two classes in the case of a single layer perceptron.
- As the perceptron learns, its weights move in weight space.
- The weights will eventually enter the solution region.

These properties are illustrated in Figure 2.4.6.

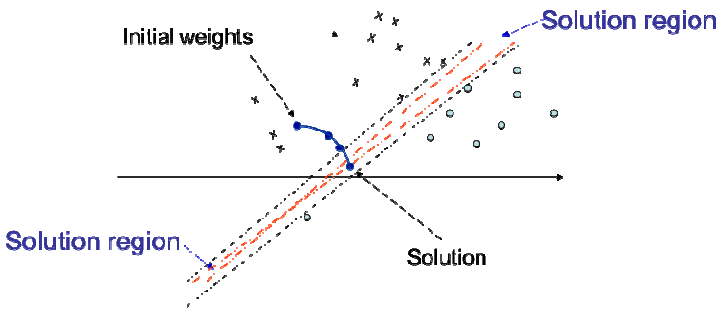


Figure 2.4.6: The solution region for the weights

2.5 Multilayer neural networks

Multilayer neural networks or the multilayer perceptron (MLP) comprises the input and output layers, as well as a number of so called hidden layers in between, see Figure 2.5.2. Note that each $h_i,\ i=1,\ldots,m,$ is termed a neuron or node and all h_i 's create the hidden layers. For a specific problem various networks with different numbers of hidden layers may be employed. Therefore, the values of the weights and also the number of hidden layers may differ. The designer should aim to design the simplest possible network to avoid modelling noise as opposed to the underlying 'true' relationship. Simple network may also lead to more predictable output and be more generalisable. The number of hidden layers will affect the number of cycles (epochs, iterations) that are required to obtain an acceptable solution. As seen in Figure 2.5.2, the k^{th} output layer and hidden layer j are given by

$$y_k = f\left(\sum_{j=0}^{m} v_{kj} h_j\right) \tag{2.5.1}$$

and

$$h_j = f\left(\sum_{i=0}^{n} w_{ji} x_i\right) \tag{2.5.2}$$

respectively. Substituting (2.5.2) in to (2.5.1) yields

$$\begin{aligned} y_k &= f\left(\sum_{j=0}^{m} v_{kj} h_j\right) \\ &= f\left(\sum_{j=0}^{m} v_{kj} f\left(\sum_{i=0}^{n} w_{ji} x_i\right)\right) \\ &= f\left(v_{00} + \sum_{j=1}^{m} v_{kj} f\left(w_{0j} + \sum_{i=1}^{n} w_{ij} x_i\right)\right) \end{aligned}$$

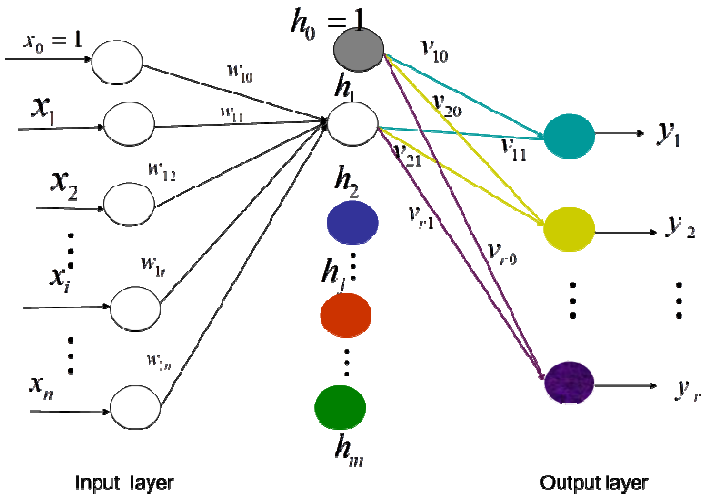


Figure 2.5.1: A MLP and connection of one node and bias.

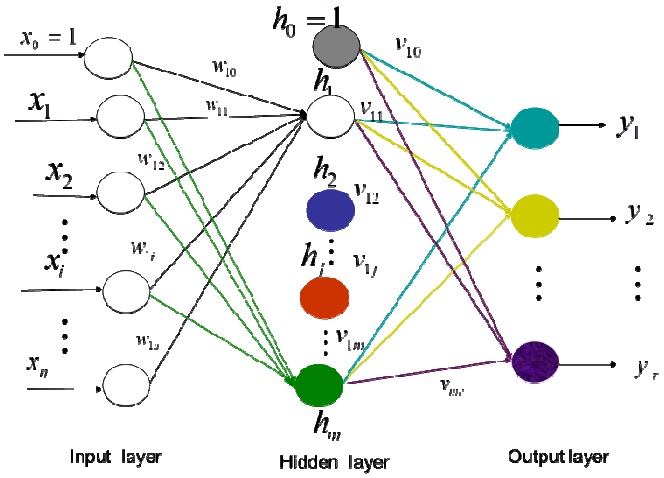


Figure 2.5.2: A full diagram of a MLP.

$y_k, 1 \leq k \leq r$, is the output with respect to the weights and inputs, or simply it is the input-output relation of the network. Note that the activation functions for both hidden and output layers, as seen in Figure 2.5.2, are the same. However, for designing a network it may be required to employ two different types of activation functions for the hidden layers and output layer. If the output activation function is either sigmoidal (logistic function) or linear, then the output of a sigmoidal neuron is constrained in the interval [0, 1] and (0,∞), respectively, whilst for a tangent hyperbolic and for a symmetric sigmoidal neuron the output interval is [-1 1]. Therefore, a linear output neuron is not constrained and can take a value of any magnitude. Such behaviour can be dangerous depending on the application and thus neural network should be used in conjunction with safety programs that check the validity of the neural network output.

Exercise 2.5.1: If the target values are within the interval (0,∞), is it possible to use a logistic function for the output layer? Note that the range of the logistic function is the interval [0,1].

Solution: In this case, the new target is defined as $t_1 = 1/t$ which within the interval (0,∞).

Proposition 2.12.1: The standard feedforward multilayer perceptron with a single nonlinear hidden layer (sigmoidal neurons) can approximate any continuous function to any desired degree of accuracy over a compact set (see Cybenko 1989, Hornik *et al.* 1989 for proof and details).

Based on the above proposition a MLP is called a universal approximator (Hornik *et al.* 1989).

Exercise 2.5.2

The following networks comprise three hidden neurons and two output neurons. It can be observed that the neural network outputs will be affected the selection of activation functions:

(a) The activation functions of the hidden and output layers are hyperbolic tangent and linear, respectively

$$y = \text{logsig}\left(w_2 \text{logsig}(w_1 x + b_1) + b_2\right)$$

$$= \text{logsig}\left(\begin{bmatrix} 0.8 & -0.2 & 1.4 \\ 0.6 & -0.9 & -0.4 \end{bmatrix} \text{logsig}\left(\begin{bmatrix} 0.6 & -0.4 & 0.5 \\ 0.3 & -0.6 & -0.4 \\ 1.1 & -2.1 & 0.3 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix} + \begin{bmatrix} 0.6 \\ 0.5 \\ -0.2 \end{bmatrix}\right) + \begin{bmatrix} 0.7 \\ -1.8 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.8453 \\ 0.2044 \end{bmatrix}$$

(b) The activation function of the hidden layer is the hyperbolic tangent while the activation functions of the output layer is the logistic function

$$y = \text{logsig}\left(w_2 \tanh(w_1 x + b_1) + b_2\right)$$

$$= \text{logsig}\left(\begin{bmatrix} 0.8 & -0.2 & 1.4 \\ 0.6 & -0.9 & -0.4 \end{bmatrix} \tanh\left(\begin{bmatrix} 0.6 & -0.4 & 0.5 \\ 0.3 & -0.6 & -0.4 \\ 1.1 & -2.1 & 0.3 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix} + \begin{bmatrix} 0.6 \\ 0.5 \\ -0.2 \end{bmatrix}\right) + \begin{bmatrix} 0.7 \\ -1.8 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.6048 \\ 0.5127 \end{bmatrix}$$

(c) The activation functions of the both hidden output layers of the network are a same logistic function

$$y = \tanh\left(w_2 \tanh(w_1 x + b_1) + b_2\right)$$

$$= \tanh\left(\begin{bmatrix} 0.8 & -0.2 & 1.4 \\ 0.6 & -0.9 & -0.4 \end{bmatrix} \tanh\left(\begin{bmatrix} 0.6 & -0.4 & 0.5 \\ 0.3 & -0.6 & -0.4 \\ 1.1 & -2.1 & 0.3 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix} + \begin{bmatrix} 0.6 \\ 0.5 \\ -0.2 \end{bmatrix}\right) + \begin{bmatrix} 0.7 \\ -1.8 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.4015 \\ 0.0510 \end{bmatrix}$$

(d) The activation of the both hidden and the output layers are the hyperbolic tangent

$$y = \tanh\left(w_2 \tanh(w_1 x + b_1) + b_2\right)$$

$$= \tanh\left(\begin{bmatrix} 0.8 & -0.2 & 1.4 \\ 0.6 & -0.9 & -0.4 \end{bmatrix} \tanh\left(\begin{bmatrix} 0.6 & -0.4 & 0.5 \\ 0.3 & -0.6 & -0.4 \\ 1.1 & -2.1 & 0.3 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 6 \end{bmatrix} + \begin{bmatrix} 0.6 \\ 0.5 \\ -0.2 \end{bmatrix}\right) + \begin{bmatrix} 0.7 \\ -1.8 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.4015 \\ 0.0510 \end{bmatrix}$$

Note that when the activation function of the output layer is linear the values of the output could be any real number, while when the activation function of this layer is the logistic function or hyperbolic tangent , the output values are within the intervals [0,1] and [-1, 1], respectively.

Example 2.5.1: Assume that inputs and initial weights are given by

$$\begin{bmatrix} 0.9 \\ -1.9 \end{bmatrix}$$

(a) Calculate the output of the network when the activation functions of the hidden and output layers are the logistic function (MATLAB® code is logsig).

$$\left(\sum w_{ji} x_i \right), \; j = 1,2,3$$

$$\begin{matrix} 0.6 & -0.4 & 0.9 \\ 0.1 & -0.6 & -0.4 \\ .1 & -2.1 & 0.7 \end{matrix} \left[\begin{matrix} 3 \\ 2 \\ 7 \end{matrix} \right] + \left[\begin{matrix} 0.6 \\ 0.4 \\ -0.9 \end{matrix} \right] \Bigg)$$

(b) Compute

$$U = \begin{bmatrix} 0.1 & -0.5 & 1.4 \\ 0.6 & -0.9 & -0.4 \end{bmatrix} H + \begin{bmatrix} 0.9 \\ -1.5 \end{bmatrix}$$

to obtain the output layer net, i.e. $u_k = \sum_{j=0}^{m} v_{kj} h_j$.

(c) Obtain e network output $Y = f(U) = \text{logsig}(U)$ where $y_k = f(u_k)$,

(d) Solutions: (a) $H = \begin{bmatrix} 0.9427 \\ 0.0573 \\ 0.1824 \end{bmatrix}$; (b)$U = \begin{bmatrix} 1.2210 \\ -1.0590 \end{bmatrix}$; $(c)\; Y = \begin{bmatrix} 0.7722 \\ 0.2575 \end{bmatrix}$.

(e)

Exercise 3.12.3: Using MATLAB® find the output of the above networks. Note that tanh and logsig MATLAB® functions indicate the hyperbolic tangent and logistic function, respectively.

To understand a MLP structure including hidden and output layers, consider the following simple network (Figure 2.5.3) with a hidden layer in which there are two neurons.

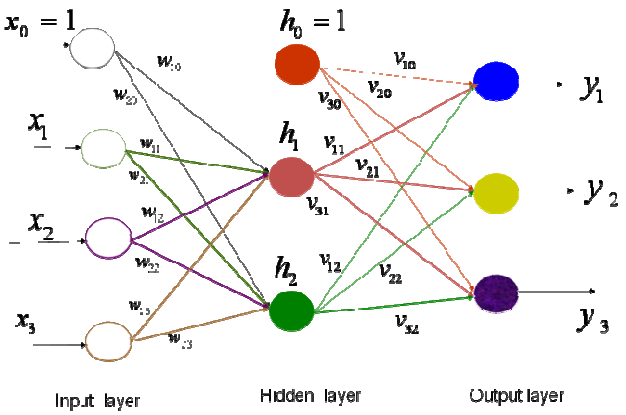


Figure 2.5.3: A simple MLP with one hidden layer and three outputs.

The output and hidden layers are given by

$$h_1 = f\left(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{10}\right)$$
$$h_2 = f\left(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + w_{20}\right)$$
$$y_1 = f\left(v_{11}h_1 + v_{12}h_2 + +v_{10}\right)$$
$$y_2 = f\left(v_{21}h_1 + v_{22}h_2 + v_{20}\right)$$
$$y_3 = f\left(v_{31}h_1 + v_{32}h_2 + v_{30}\right)$$

This network can also be shown as

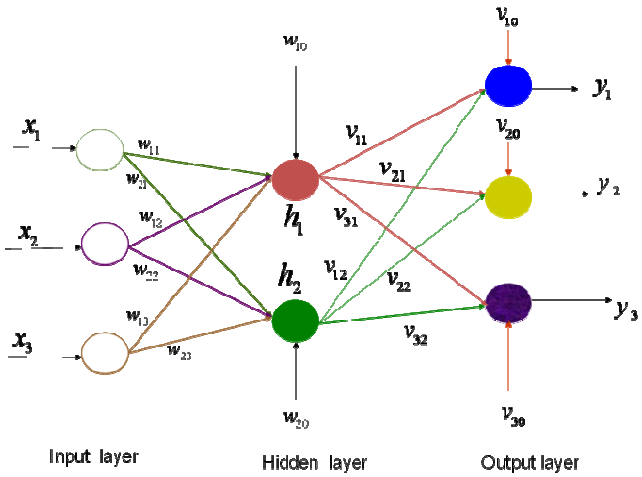


Figure 2.5.4: A simple MLP with one hidden layer and three outputs.

2.5.1 XOR problem

Recall the XOR problem, i.e.

Input		Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.12.2: XOR problem

The XOR problem can be solved using appropriate MLPs since the input can be separated by two straight lines into two regions. Assume that x_1 and x_2 are the inputs and $w_{ij}(i=1,2,0, j=1,2)$ are the weights. Then one can select two lines such that (0,0) and (1,1) are in one region (between two lines) and another two inputs lie outside of this region as shown in Figure 2.5.5.

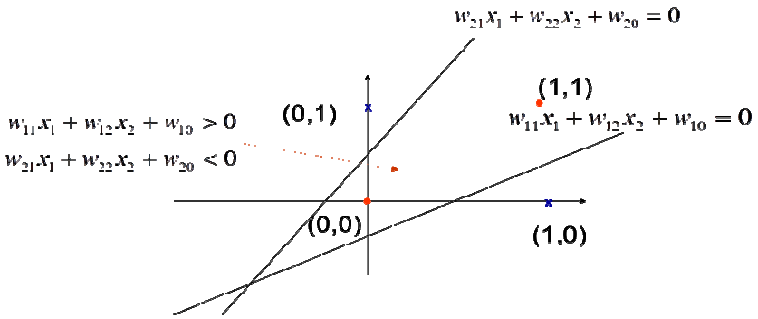


Figure 2.5.5: Separation of the XOR inputs using two lines.

The XOR problem can be solved in different ways. For example, select the step function and AND operation as the hidden and output layers activation functions, respectively, see Figure 2.5.6.

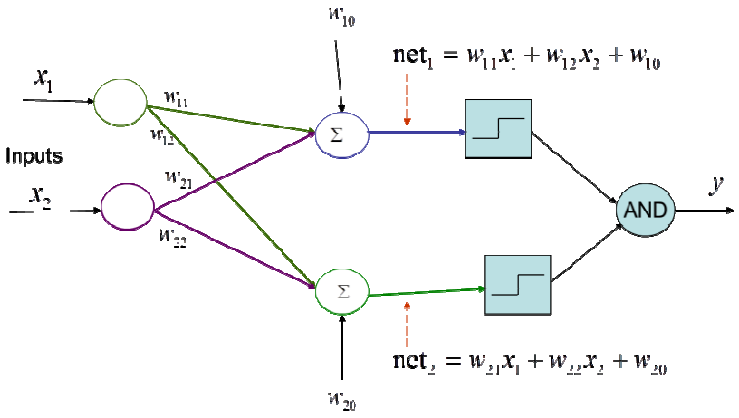


Figure 2.5.6: A XOR network.

One may select a the weights such that nets are

$$\text{net}_1 = -x_1 + x_2 + 0.5$$
$$\text{net}_2 = -x_1 + 2x_2 - 0.5$$

These two lines separate the plane into two regions such that (0,0) and (1,1) lie inside the region between the two lines and the two others are the outside of this region, see Figure 2.5.7**Błąd! Nie można odnaleźć źródła odwołania.**.

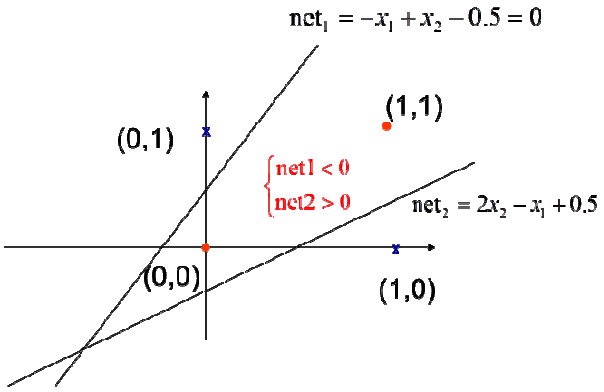


Figure 2.5.7: The net function associated to the XOR problem.

2.6 Backpropagation Method

Backpropagation (BP) is a general method for iteratively updating of a multilayer perceptron's weights and biases. BP is an optimisation technique based on minimising an objective function such as a squared error function. If the learning rate is small BP is stable, because it uses the steepest decent technique, however, its convergence is slow. Overall, it is a simple method compared with others. . In this section an algorithm is given to update the weights and then find a set of appropriate weights using an iterative process. In this regards, the rates of change of the weights are first calculated. Consider the MLP network as shown in Figure 2.5.2, and assume that

$$h_j = f\left(\sum_{i=0}^{n} w_{ji}x_i\right) = f(z_j), \quad y_k = f(u_k), \quad u_k = \sum_{j=0}^{m} v_{kj}h_j \text{'} \quad z_k = \sum_{i=0}^{n} w_{ji}x_i \tag{2.6.1}$$

It is required to minimise the total error given by

$$E = \frac{1}{2}\sum_{j=1}^{m}\left(t_j - y_j\right)^2 \tag{2.6.2}$$

to obtain the rate of updating of the weights. Therefore the partial derivative of E with respect to all weights and biases must be calculated. The partial derivative with respect to v_{kj} is

$$\frac{\partial E}{\partial v_{kj}} = \frac{\partial E}{\partial u_k}\frac{\partial u_k}{\partial v_{kj}} = \frac{\partial E}{\partial u_k}\left(\frac{\partial}{\partial v_{kj}}\sum_{j=0}^{m} v_{kj}h_j\right) \tag{2.6.3}$$

On the other hand,

$$\frac{\partial}{\partial v_{kj}}\sum_{j=0}^{m} v_{kj}h_j = h_k, \qquad (2.6.4)$$

$$\frac{\partial E}{\partial u_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial u_k} = -\left(t_k - y_k\right) f'(u_k)$$

since $\frac{\partial E}{\partial y_k} = -\left(t_k - y_k\right)$, $\frac{\partial y_k}{\partial u_k} = f'(u_k)$. Substituting (2.6.4) into (2.6.3) gives

$$\frac{\partial E}{\partial v_{kj}} = \frac{\partial E}{\partial u_k}\frac{\partial u_k}{\partial v_{kj}} = \frac{\partial E}{\partial u_k}\left(\frac{\partial}{\partial v_{kj}}\sum_{j=0}^{m} v_{kj}h_j\right) \qquad (2.6.5)$$

$$= -\left(t_k - y_k\right) f'(u_k)h_k$$

Therefore, $\Delta v_{kj} = \beta\left(t_k - y_k\right) f'(u_k)h_j$ where β is a small number. The partial derivatives E defined in (2.6.2) with respect to the weights w_{ji} are now calculated. Therefore,

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}}$$

On the other hand,

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial h_j}\frac{\partial h_j}{\partial z_j} = \frac{\partial E}{\partial h_j} f'(z_j)$$

$$\frac{\partial z_j}{\partial w_{ji}} = \sum_{i=0}^{n}\frac{\partial}{\partial w_{ji}}\left(w_{ji}x_i\right) = x_i$$

and

$$\frac{\partial E}{\partial h_j} = \frac{1}{2}\sum_{k=1}^{m}\frac{\partial\left(t_k - f\left(v_{kj}h_j\right)\right)^2}{\partial h_j}$$

Therefore,

$$= \sum_k (t_k - y_k) f'(u_k) v_{kj}$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}}$$

$$= x_i f'(z_j)\sum_k (t_k - y_k) f'(u_k) v_{kj}$$

and the rate of the change of w_{ji} is

$$\Delta w_{ji} = \eta x_i f'(z_j)\sum_k (t_k - y_k) f'(u_k) v_{kj}$$

where η is small number. Based on the above calculations, the hidden and output layers weights are tuned as follows:

First the learning rates β and η are selected and then the new output and hidden layers weights are calculated using the following rules:

$$\begin{aligned}\tilde{v}_{kj} &= v_{kj} + \Delta v_{kj} \\ &= v_{kj} + \beta\left(t_k - y_k\right) f'(u_k) h_j \\ \tilde{w}_{kj} &= w_{kj} + \Delta w_{kj} \\ &= w_{ji} + \eta x_i f'(z_j) \sum_k (t_k - y_k) f'(u_k) v_{kj}\end{aligned}$$

2.6.1 Backpropagation algorithm (summary)

Assume that k pairs of training patterns input x, target t are going to be used for training the network. The algorithm is as follows:

1. Initialise all weights to small random values within the range [-*a, a*] where *a*>0.

2. Select a pair of training patterns input *x*, target *t* (randomly) in feedforward direction and compute the all outputs.

3. Select the learning rates β and η and calculate
$$\begin{aligned}\Delta v_{kj} &= \beta\left(t_k - y_k\right) f'(u_k) h_j \\ \Delta w_{ji} &= \eta x_i f'(z_j) \sum_k (t_k - y_k) f'(u_k) v_{kj}\end{aligned}$$

4. Calculate the new weights
$$\begin{aligned}\tilde{v}_{kj} &= v_{kj} + \Delta v_{kj} \\ \tilde{w}_{kj} &= w_{kj} + \Delta w_{kj}\end{aligned}$$

5. Go to step 2, i.e. select the new pair of training patterns input x, target t and find all new outputs. Repeat this cycle for k times until all training patterns inputs and outputs are used. In this step one cycle (epoch) is completed.

6. Calculate the total error $E = \frac{1}{2}\sum_{j=1}^{m}\left(t_j - y_j\right)^2$. If E is sufficiently small, i.e. $E < \varepsilon$ where $\varepsilon > 0$ is a desired small value. The current weights tune the network and the neural network training process is complete. Otherwise, use the final weights obtained from step 5 after completion of the last epoch as initial values and repeat steps 2 to 6.

A training cycle or epoch is completed each time the initial weights and the all training patterns inputs and outputs are used. In addition, it is not always possible to obtain the desired results using only one training cycle. Several training cycles, the so-called epochs, may be required to obtain a set of appropriate weights and therefore the correct outputs.

2.6.2 The perceptron convergence theorem

If the inputs are linearly separable, the perceptron will find a decision boundary which correctly divides the inputs.

Example 2.6.1: Assume that the activation function is the logistic function

$$f(x)=\frac{1}{1+e^{-\alpha x}}$$

Then $\frac{df}{dx}=\alpha(1-f(x))f(x)$ and the updated output layer weights are obtained from

$$\tilde{v}_{kj}=v_{kj}+\Delta v_{kj}$$
$$\Delta v_{kj}=\alpha\beta\left(t_k-y_k\right)f(u_k)(1-f(u_k))h_j$$

and the updated hidden layer weights are

$$\tilde{w}_{ji}=\tilde{w}_{ji}+\Delta w_{ji}$$
$$\Delta w_{ji}=\alpha^2\eta x_i f(z_j)(1-f(z_j))\sum_k(t_k-y_k)f(u_k)(1-f(u_k))v_{kj}$$

These updating processes are completed when all pairs of training patterns inputs and targets are used and the total error $E=\frac{1}{2}\sum_{j=1}^{m}\left(t_j-y_j\right)^2$ is sufficiently small, i.e. it takes nearly its minimum value.

The following example illustrates how the backpropagation method is applied and presents the effect the number of neurons and the number of epochs has on tuning of the network.

Example 3.15.2: Using the backpropagation method for the input

$$x=\begin{bmatrix}0.2 & 0.6 & 1.7 & -1.6\\ 0.9 & 0.3 & 1.2 & -02\\ 0.2 & 0.1 & 0.4 & 2\end{bmatrix}$$

and the target $t = [0.1\ 0.3\ 0.6\ 0.2]^T$, the final appropriate weights are obtained after 187epochs, see Figure 2.6.1.

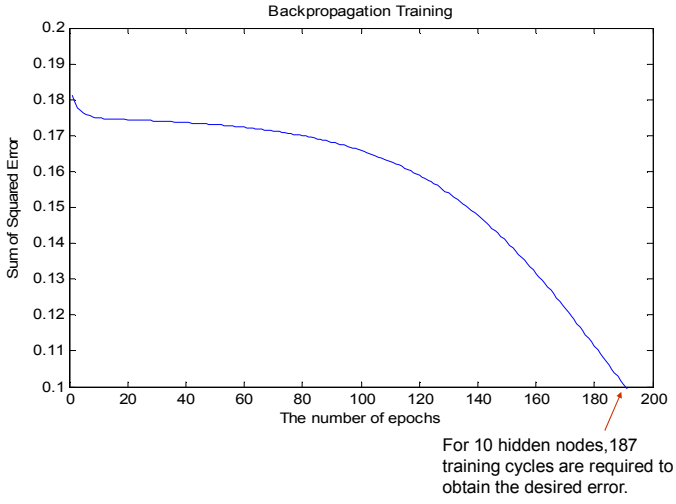


Figure 2.6.1: Using the backpropagation algorithm to tune the network.

The number of hidden nodes and the number of epochs for this example are given in Table 2.6.1 and it shows which when two nodes are selected, less epochs are required for the network training. Figure 2.6.2 depicts the process of a neural network algorithm.

The number of hidden nodes and the number of epochs for this example are given in Table 2.6.1. It shows that when too few or too many nodes are considered more epoch are required to reach the desired target.

Number of hidden neurons	1	2	3	4	5	6	7	8	9
The number of epochs required to reach the desired tuning	188	170	165	166	169	172	180	183	186

Table 2.6.1: The number of hidden nodes and the number of epochs that required completing the network learning.

Note on creating a MATLAB® function using the backpropagation algorithm

To define and train a MLP using the BP algorithm, activation functions should be first specified. The number of activation functions depends on the number of layers. The activation functions are normally logistic function (sigmoid function, **logsig** MATLAB® codes). This limits the network output to the interval [0 1].

Since most problems have targets outside of this range, a linear function is selected as the activation (training) function of the output layer. A linear output layer allows targets of any magnitude to be reached. A backpropagation MATLAB® m-file is given in Appendix A, named '**BP_training.m'**. .

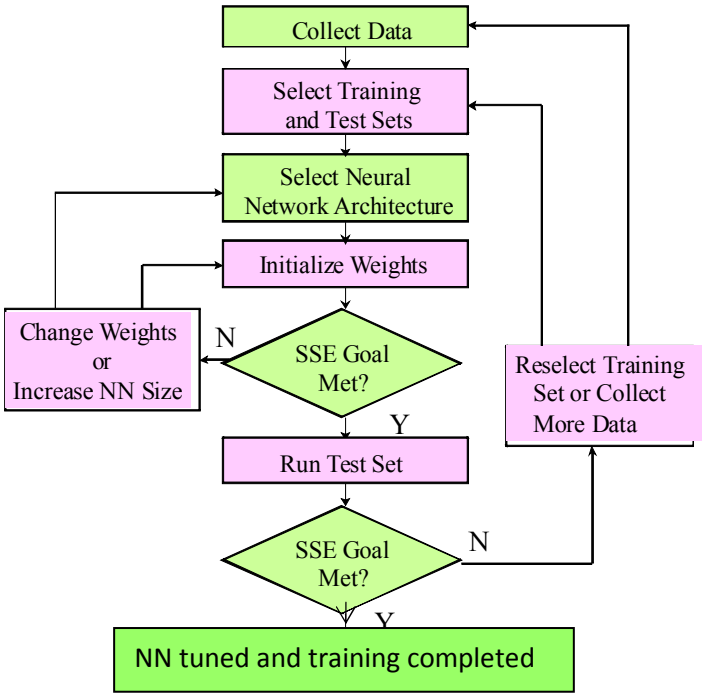


Figure 2.6.2: Overview of neural network training method

2.7 Gaussian radial basis function network

The radial basis function (RBF) network is an important neural network that has been widely used. The activation function of the hidden layer is normally a Gaussian function, in such case, the network is termed a Gaussian RBF (GRBF).

The characteristic of a RBF network can be summarised as follows:

1. a two layer network which has different types of neurons in the hidden layer and the output layer
2. The hidden layer which corresponds to a MLP hidden layer is a nonlinear mapping.
3. The hidden layer contains radial basis function neurons.
4. The activation function is normally a Gaussian function.
5. The activation functions are centred over areas in the input.

The Gaussian activation function of an RBF Network is

$$g_j(x) = e^{-\frac{(x-c_j)^2}{\sigma_j^2}}$$

where x is the input vector, c_j the centre of a region referred to as the receptive field and σ_j is the width of the receptive field. Note that $g_j(x)$ is the output of the j-th neuron of the GRBF. Figure 2.7.1 shows a Gaussian neuron with the centre at 0 and the width 1. A RBF is a local network that is trained in a supervised manner.

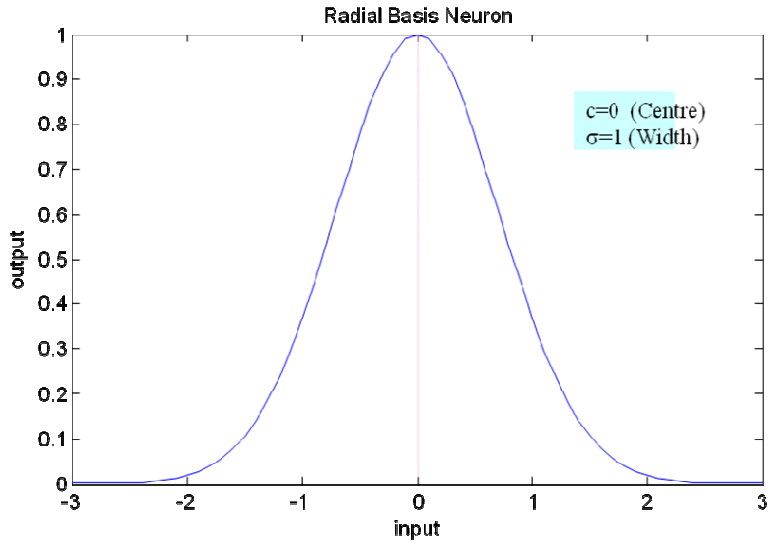


Figure 2.7.1: A Gaussian neuron.

Note that the input of a GRBF network constructs a set, the so-called a receptive field, which determines the range of the given data.

An RBF network has the following features:

1. The output layer is a layer of standard linear neurons.
2. Training algorithms are not based on any iterative method.
3. AS MLP, the output layer may, or may not, contain biases.
4. If an input vector x lies near the centre of a receptive field (c), then the RBF hidden layer neuron will be activated. So a suitable centre must be identified for each neuron.
5. If an input vector lies between two receptive field centres, but inside their receptive field width (σ) then the hidden neurons are both partially activated.
6. When the input vectors lie far from all receptive fields there is no hidden layer activation and the RBF output is equal to the output layer bias values.

The concepts of local and global networks will now be explained and the effect of networks with small neuron width and large neuron width will be discussed. In addition, the difference between an RBF and the standard MLP will be investigated. Note that a RBF performs a local mapping if only

inputs lie near a receptive field. In fact, the data which are the outside of the receptive field is not considered. The receptive filed for a Gaussian neuron shown in Figure 2.7.1 is [-3, 3].

A RBF is a local network while a MLP network is a global network. The concepts of local and global are defined based on the inputs. To train a network using a MLP, the training data are not classified while using GRBF the data should normally be classified.

Figure 2.7.2 illustrate the distinction between local and global mapping. Note that an MLP is a global network if all inputs yield only are output.

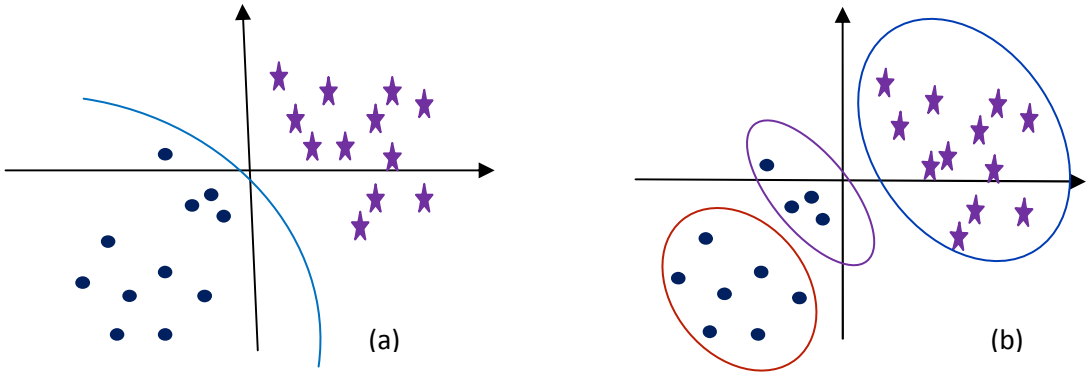


Figure 2.7.2: (a) Global mapping; (b) Local mapping.

2.8 Difference between a RBF and the standard MLP

Using an MLP, one cannot judge whether or not the input vector comes from the training set or an untrained regions. It is therefore not possible to assess if the output contains significant information, while an RBF is able to determine if the network is operating outside its training region. When a neural network is faced with unseen data, it has not been trained to recognise, its output will not be appropriate and should therefore be discarded. The RBF ability to determine if its inputs are from outside its training sets makes it suitable candidates for very sensitive problems in marketing, finance and safety critical applications. Two most important issue for using GRBF are the selection of the widths and centres (corresponding to mean and standard deviation in statistics) of the neurons. An RBF with a large width covers may cover a large section of the input space and is therefore very likely to be activated. Such behaviour may result in difficulty for the network to learn generalisable relations between its inputs and the target outputs. A RBF network with a small width has the following characteristics:

(a) There is no overlap between the neurons, and the number of the neurons may be highly increased.

(b) Since it is possible to have a number of receptive fields with insufficient inputs, the tuned network may be is not suitable when it is used for unseen data.

Therefore, the width should be selected to be neither too small nor too large.

Note that all inputs must lie within a receptive field and have a proper overlap with other receptive fields; the width needs to be at least equal to the distance between inputs of classes to guarantee that each input is member of at least one class (i.e. receptive field).

Błąd! Nie można odnaleźć źródła odwołania. and **Błąd! Nie można odnaleźć źródła odwołania.** show the modelled and actual input-output relationship for two networks. **Błąd! Nie można odnaleźć źródła odwołania.** shows illustrated that using a RBF with a large width of 20, which include all inputs, gives better fit between the data and the model whereas when small RBF width are used the network is not able to predict the output accurately. Therefore, the selection of a width for RBF network is an important issue and should be selected such that it covers of all inputs lie within the receptive field. Both large widths more than sufficient or small widths less than an appropriate lead to badly tuned networks with no guarantees that they will work for unseen data.

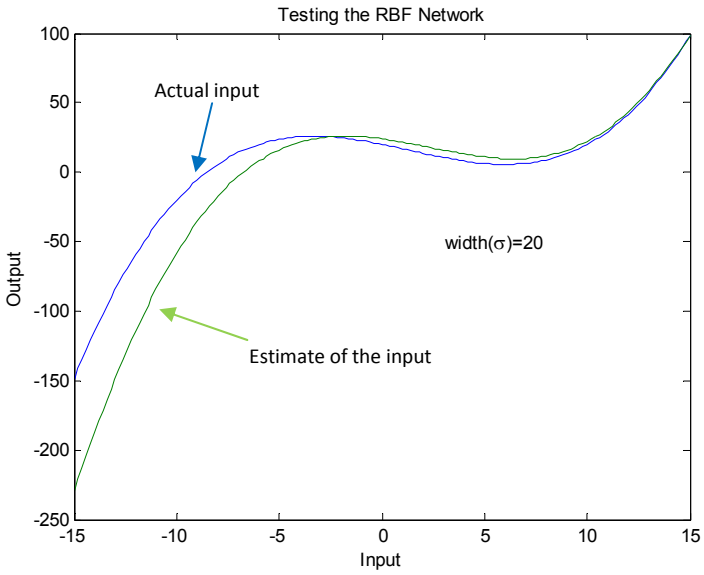


Figure 2.8.1: The RBF network with a width 20.

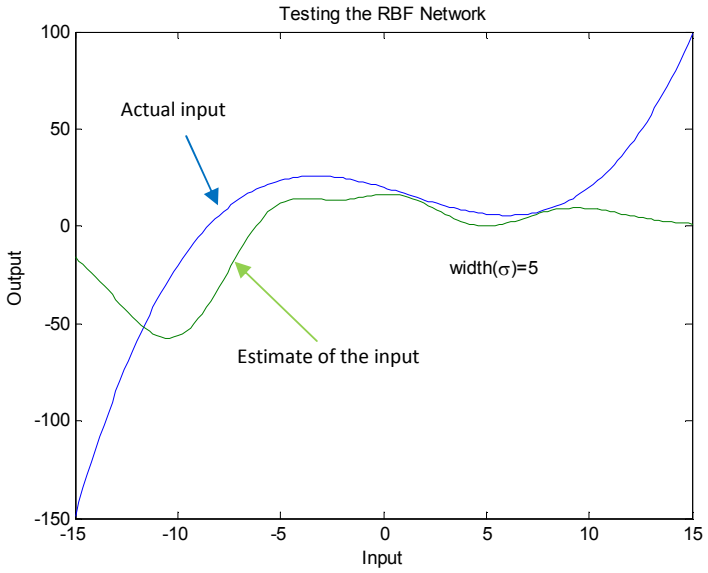


Figure 2.8.2: The RBF network with a width 5.

2.8.1 Gaussian radial basis function (GRBF) neural networks

The GRBF is a two-layer neural network in which the activation function of the hidden layer of the network is Gaussian. The relationship between the input and output of a RBF network is shown in Figure 2.8.3 in which $x_1, x_2, \ldots, x_M$ are inputs, $c_1, c_2, \cdots, c_M$ are the centres and $\|x-c\|$ indicates the distance between the input and the centres where

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}, \quad \mathrm{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_M \end{bmatrix}, \quad \|x-c\| = \sqrt{(x_1-c_1)^2 + (x_2-c_2)^2 + \cdots (x_M-c_M)^2}, \quad y = e^{-v^2} = e^{-\frac{\|x-c\|^2}{\sigma^2}}$$

Figure 2.8.4 depicts the GRBF network diagram in which $g_i = g_i(r_i, \sigma_i),\ i = 1, \cdots, N,$ are the Gaussian neurons of the hidden layer. The outputs of the GRBF are

$$\begin{aligned}
y_1 &= g_1(r_1,\sigma_1)w_{11} + g_2(r_2,\sigma_2)w_{12} + \cdots + g_j(r_j,\sigma_j)w_{1j} + \cdots + g_N(r_N,\sigma_N)w_{1N} + b_1 \\
y_2 &= g_1(r_1,\sigma_1)w_{21} + g_2(r_2,\sigma_2)w_{22} + \cdots + g_j(r_j,\sigma_j)w_{2j} + \cdots + g_N(r_N,\sigma_N)w_{2N} + b_2 \\
&\vdots \\
y_k &= g_1(r_1,\sigma_1)w_{k1} + g_2(r_2,\sigma_2)w_{k2} + \cdots + g_j(r_j,\sigma_j)w_{kj} + \cdots + g_N(r_N,\sigma_N)w_{kN} + b_k \\
&\vdots \\
y_P &= g_1(r_1,\sigma_1)w_{P1} + g_2(r_2,\sigma_2)w_{P2} + \cdots + g_j(r_j,\sigma_j)w_{Pj} + \cdots + g_N(r_N,\sigma_N)w_{PN} + b_p
\end{aligned}$$

which can be represented in compact form as:

$$y_k = \sum_{j=1}^{j=N} g_j(r_j,\sigma_j)w_{kj} + b_k, \quad (1 \le k \le p)$$

or in matrix form:

$$\begin{bmatrix} g_1(r_1,\sigma_1) & g_2(r_2,\sigma_2) & \cdots & g_N(r_N,\sigma_N) & 1 \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{P1} \\ w_{12} & w_{22} & \cdots & w_{P2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1N} & w_{2N} & \cdots & w_{PN} \\ b_1 & b_2 & \cdots & b_p \end{bmatrix} = \begin{bmatrix} y_1 & y_2 & \cdots & y_P \end{bmatrix}$$

The weights must be modified such that the outputs of the system become the targets. Therefore, the weights must satisfy the following relations:

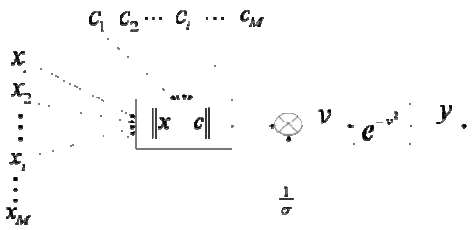


Figure 2.8.3: Input-output model of a GRBF neural network.

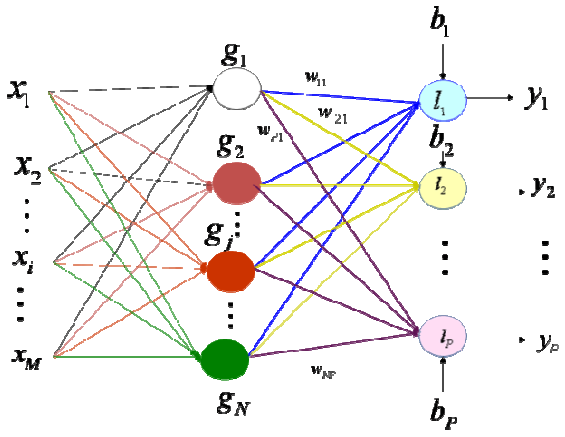


Figure 2.8.4: An GRBF neural network diagram.

$$\begin{bmatrix} g_1(r_1,\sigma_1) & g_2(r_2,\sigma_2) & \cdots & g_N(r_N,\sigma_N) & 1 \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{P1} \\ w_{12} & w_{22} & \cdots & w_{P2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1N} & w_{2N} & \cdots & w_{PN} \\ b_1 & b_2 & \cdots & b_p \end{bmatrix} = \begin{bmatrix} t_1 & t_2 & \cdots & t_P \end{bmatrix} \tag{2.8}$$

in which $\begin{bmatrix} t_1, & t_2 & \cdots & t_P \end{bmatrix}$ are the targets. The network model (000) can expressed as

$$\underbrace{\begin{bmatrix} g_1(r_{11},\sigma_1) & g_2(r_{12},\sigma_2) & \cdots & g_N(r_{1N},\sigma_N) & 1 \\ g_1(r_{21},\sigma_1) & g_2(r_{22},\sigma_2) & \cdots & g_N(r_{2N},\sigma_N) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ g_1(r_{Q1},\sigma_1) & g_2(r_{Q2},\sigma_2) & \cdots & g_N(r_{QN},\sigma_N) & 1 \end{bmatrix}}_{G} \underbrace{\begin{bmatrix} w_{11} & w_{21} & \cdots & w_{P1} \\ w_{12} & w_{22} & \cdots & w_{P2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1N} & w_{2N} & \cdots & w_{PN} \\ b_1 & b_2 & \cdots & b_p \end{bmatrix}}_{W} = \underbrace{\begin{bmatrix} t_1^T \\ t_2^T \\ \vdots \\ t_Q^T \end{bmatrix}}_{T}$$

or in the simplified form $GW = T$ where $G\varepsilon\mathbb{R}^{Q\times(N+1)}$. Note that, in general $Q \neq N+1$.

2.8.2 Estimation of the weight matrix

Now it is required to find the weight matrix W such that $GW = T$. Assume that the square matrix $\mathrm{G}^{\mathrm{T}}G$ is invertible. Since $\mathrm{G}^{\mathrm{T}}GW = \mathrm{G}^{\mathrm{T}}T$ then

$$\left(\mathrm{G}^{\mathrm{T}}G\right)^{-1}\mathrm{G}^{\mathrm{T}}GW = \left(\mathrm{G}^{\mathrm{T}}G\right)^{-1}\mathrm{G}^{\mathrm{T}}T$$

which implies that $W = \left(\mathrm{G}^{\mathrm{T}}G\right)^{-1}\mathrm{G}^{\mathrm{T}}T$. Now assume that $\hat{W}$ is an estimate of W. It required proving that $\varepsilon = G\hat{W} - T$ is zero. To this end, consider

$$\begin{aligned} \varepsilon^2 = \varepsilon^{\mathrm{T}}\varepsilon &= \left(G\hat{W} - T\right)^T\left(G\hat{W} - T\right) \\ &= \left(\hat{W}^TG^T - T^T\right)\left(G\hat{W} - T\right) \\ &= \hat{W}^TG^TG\hat{W} - T^TG\hat{W} - \hat{W}^TG^TT + T^TT \end{aligned}$$

Now the derivative ε^2 with respect $\hat{W}$ is calculated

$$\begin{aligned} \frac{\partial\varepsilon^2}{\partial\hat{W}} = \frac{\partial\varepsilon^{\mathrm{T}}\varepsilon}{\partial\hat{W}} &= 2G^TG\hat{W} - \left(T^TG\right)^T - G^TT \\ &= 2G^TG\hat{W} - 2G^TT \end{aligned}$$

To obtain the minimum, equate to zero

$$\frac{\partial\varepsilon^2}{\partial\hat{W}} = \frac{\partial\varepsilon^{\mathrm{T}}\varepsilon}{\partial\hat{W}} = 2G^TG\hat{W} - 2G^TT = 0$$

which yields

$$\hat{W} = \left(G^TG\right)^{-1}G^TT$$

Note that $\frac{\partial(\cdot)}{\partial\hat{W}} = \left[\frac{\partial(\cdot)}{\partial\hat{W}_1} \quad \frac{\partial(\cdot)}{\partial\hat{W}_2} \quad \cdots \quad \frac{\partial(\cdot)}{\partial\hat{W}_N}\right]$.

Example 3.20.1:

Using MATLAB, calculate the weight matrix $\hat{W} = \left(G^T G\right)^{-1} G^T T$ for the XOR problem.

Answer: See the end of Example 3.20.2.

The XOR Problem solved using a GRBF

The GRBF network diagram for XOR is presented in **Błąd! Nie można odnaleźć źródła odwołania.** .

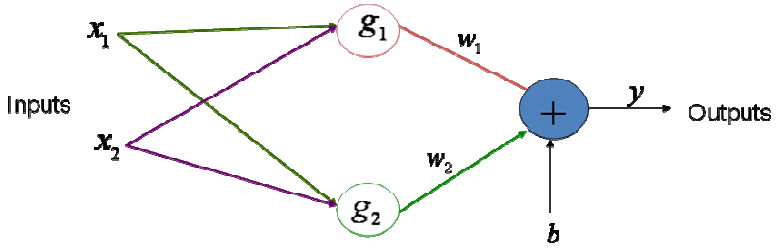


Figure 2.8.5: The GRBF neural network diagram for XOR.

The output network is obtained from the following relation:

$$y = w_1 g_1\left(r_1, \sigma_1\right) + w_2 g_2\left(r_2, \sigma_2\right) + b$$

in which

$$g_1\left(r_1, \sigma_1\right) = e^{-\frac{r_1^2}{\sigma_1^2}}, \quad r_1 = \left\| c_1 - [u_1 \; u_2] \right\|$$

$$g_2\left(r_2, \sigma_2\right) = e^{-\frac{r_2^2}{\sigma_2^2}}, \quad r_2 = \left\| c_2 - [u_1 \; u_2] \right\|$$

where c_1 and c_2 are the centres.

Select two of the four inputs as the potential centres, say, $c_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}$ and $c_2 = \begin{bmatrix} 0 & 0 \end{bmatrix}$. A suitable width, σ, is obtained from

$$\sigma = \frac{d_{\max}}{\sqrt{C}}$$

where C the number of the centres and $d_{\max}$ is the maximum distance between the selected centres**.** Since the number of the centres is $C = 2$ and the maximum distance between two centres is $d_{\max} = \sqrt{(0-1)^2 + (0-1)^2} = \sqrt{2}$. Therefore,

$$\sigma = \frac{d_{\max}}{\sqrt{C}} = \frac{\sqrt{2}}{\sqrt{2}} = 1$$

All quantities which are used for obtaining the weights are given in
Table 2.2.

x_1	x_2	r_1	$g_1(r_1, \sigma_1)$	r_2	$g_2(r_2, \sigma_2)$	t
0	0	$\sqrt{2}$	0.1353	0	1	0
0	1	1	0.3678	1	0.3678	1
1	0	1	0.3678	1	0.3678	1
1	2	0	1	$\sqrt{2}$	0.1353	0

Table 2.2: The inputs and hidden layers output of GRBF network for XOR.

Substituting the given values in table into $y = w_1 g_1(r_1, \sigma_1) + w_2 g_2(r_2, \sigma_2) + b$ yields

$$\begin{aligned}
0.1353w_1 + w_2 + b &= 0 \\
0.3678w_1 + 0.3678w_2 + b &= 1 \\
0.3678w_1 + 0.3678w_2 + b &= 1 \\
w_1 + 0.1353w_2 + b &= 0
\end{aligned}$$

or in the matrix form

$$\begin{bmatrix} 0.1353 & 1 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.3678 & 0.3678 & 1 \\ 1 & 0.1353 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Which can represented in the reduced form

$$\begin{bmatrix} 0.1353 & 1 & 1 \\ 0.3678 & 0.3678 & 1 \\ 1 & 0.1353 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

The solutions are

$$W = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} -2.5027 \\ -2.5027 \\ 2.8413 \end{bmatrix}$$

Therefore the network output is $y = -2.5027e^{-r_1^2} - 2.5027e^{-r_2^2} + 2.8413$ in which r_1 and r_2 are given in Table 3.20.1. Note that, the weights can be found using $\hat{W} = \left(G^T G\right)^{-1} G^T T$ where

$$G = \begin{bmatrix} 0.1353 & 1 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.3678 & 0.3678 & 1 \\ 1 & 0.1353 & 1 \end{bmatrix}$$

Therefore

$$\hat{W} = \left(G^T G\right)^{-1} G^T T = \begin{bmatrix} 0.6727 & -1.2509 & -1.2509 & 1.8292 \\ 1.8292 & -1.2509 & -1.2509 & 0.6727 \\ -0.9202 & 1.4202 & 1.4202 & -0.9202 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -2.5019 \\ -2.5019 \\ 2.8404 \end{bmatrix}$$

2.9 Generalised Regression Neural Network (GRNN)

A GRNN has three layers including hidden layer, summation and output layers, see .Figure 2.9.1 Assume that the activation function of the hidden layer is Gaussian the network as shown in Figure 2.9.1.

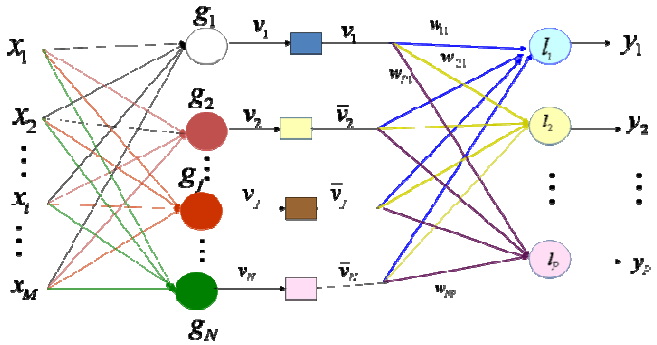


Figure 2.9.1: Gaussian generalised Regression Neural Network schematic.

The output of a GRNN is given by

$$\bar{v}_n = \frac{v_n}{\sum_{j=1}^{N} v_j} = \frac{e^{\frac{\|x_n - c_n\|^2}{\sigma_n^2}}}{\sum_{j=1}^{N} e^{\frac{\|x_j - c_j\|^2}{\sigma_j^2}}}$$

Then for $p = 1, \cdots, N$

$$y_p = \sum_{j=1}^{N} w_{pj} \bar{v}_j = \frac{\sum_{j=1}^{N} w_{pj} \, \mathrm{e}^{-\frac{\|x_j - c_j\|^2}{\sigma_j^2}}}{\sum_{j=1}^{N} \mathrm{e}^{-\frac{\|x_j - c_j\|^2}{\sigma_j^2}}} = \frac{S - \text{summation}}{D - \text{summation}}$$

2.10 K-means algorithm

2.10.1 Importance of k-means algorithms.

Since the centres of the receptive fields of data in GRBF and GRNN highly affect the neural network configuration, a systematic method is required to yield appropriate centres. The first step in identifying the centres is to classify the data. The centres of the radial neurons should then be determined such that the inputs within each class (cluster) are the members of a class whose distances from the class centre are less than other clusters. The set of cluster centres is required to represent the natural distribution of the training cases. The k-means algorithm assigns the radial centres of radial neurons to the first hidden layer in the network. See Duda and Hart (1973); Moody and Darkin (1989); Bishop (1995) for details. Let k be the number of radial neurons (also is called nodes or units). 'k-means' assigns each training inputs to one of the *k* clusters such that each cluster is represented by its centre of inputs whose are member of the cluster, and each input is closer to the centre of its cluster than to the centre of any other cluster.

k-means algorithms are based on an iterative procedures, and the following steps show the procedure for obtaining the final clusters and centres using any algorithms:

(a) First the k clusters are randomly assigned or are arbitrarily structured by selecting the first k cases.

(b) Then using an appropriate method the centre of each cluster is calculated.

(c) Each input is tested to see whether the centre of another cluster is closer to it than the centre of its own cluster; if so, the input is reassigned.

(d) The new clusters are formed and the new centres are recalculated.

(e) The algorithm repeats, i.e. it will start from step (c).

The iteration is completed if the reassignment of the cases is not possible, i.e. each input is closer to the centre of its cluster than to the centre of any other cluster.

Note that there is no formal proof of convergence for this algorithm, although in practice it usually converges quickly. Now a simple example is presented to show how the k-means algorithm works.

Example 7.1.1: Suppose there are six different chemical products A, B, C, D, F and E, and these must be classified based on the amount of two ingredient substances. The results of measurements are shown in the following table:

	A	B	C	D	F	E
Substance 1	2.0	3.0	3.5	2.1	3.2	3.8
Substance 2	1.5	2.4	5.0	3.0	1.0	6.0

First two centres are arbitrarily selected. For example the pairs C_1(2.0, 1.5) and C_2(3.5, 5.0) are chosen as the centres of the clusters.

	A	B	C	D	F	E
Substance 1	2.0	3.0	3.5	2.1	3.2	3.8
Substance 2	1.5	2.4	5.0	3.0	1.0	6.0

C_1 C_2

Now the distances of all points from these two centres are calculated. These distances are shown in following table:

Table D_0: Distance of each pair from the centres C_1 and C_2

	A	B	C	D	F	E
Distance from C_1	0.00	1.35	3.81	1.50	1.30	4.85
Distance from C_2	3.81	2.64	0.00	2.44	4.01	1.00

For example The distances of the pair corresponding to product B form the centres are calculated as follows:

$$\|(2.0,1.5)-(3.0,2.4)\| = \sqrt{(3.0-2.0)^2+(2.4-1.5)^2}$$
$$=1.35$$
$$\|(3.5,5.0)-(3.0,2.4)\| = \sqrt{(3.0-3.5)^2+(5.0-2.4)^2}$$
$$=2.64$$

Then assigning 0 and 1, it is determined whether the points are near to one of the centres. If the point is near the centre 0 is assigned, otherwise assign 1. See the following table which is termed the 'evaluation table'

Table G_0: The evaluation of the distances from the centres C_1 and C_2						
	A	B	C	D	F	E
Distance from C_1	0	0	1	0	0	1
Distance from C_2	1	1	0	1	1	0

Based on the distances from the centres C_1 and C_2, two clusters are assigned. The elements within the dashed lines belong to one cluster and the other one to another cluster. The products corresponding to 0 will belong to the same cluster and the products assigned to 1 will be in the other cluster.

	A	B	C	D	F	E
Substance 1	2.0	3	3.5	2.1	3.2	3.8
Substance 2	1.5	2.4	5.0	3.0	1.0	6.0
	C_1		C_2			

Now the new centres are reassigned which are the mean values (average) of the all points which are located in the same clusters. So the new centres are

$$\hat{C}_1 = \left(\frac{2+3+2.1+3.2}{4} \quad \frac{1.5+2.4+3+1}{4} \right) = (2.56 \quad 1.98)$$

$$\hat{C}_2 = \left(\frac{3.5+3.8}{2} \quad \frac{5+6}{2} \right) = (3.65 \quad 5.5)$$

Now the distance of the points from the new centres are computed as shown in the following table.

Table D_1 : Distance from of each pair from the centres $\hat{C}1$ and $\hat{C}2$						
	A	B	C	D	F	E
Distance from $\hat{C}1$	0.74	0.61	3.16	1.12	1.17	4.21
Distance from $\hat{C}2$	4.32	3.17	0.52	2.94	4.52	0.52

Then the evaluation table for new distance is

Table G_1: The evaluation of distance from the centres $\hat{C}1$ and $\hat{C}_2$						
	A	B	C	D	F	E
Distance from $\hat{C}1$	0	0	1	0	0	1
Distance from $\hat{C}2$	1	1	0	1	1	0

Since the new reassignment is not required, i.e. Tables G_1 and G_0 are the same, so no changes to the members of two clusters are needed and Table D_1 determines the two clusters. The first cluster comprises of the products A, B, D and F (see Table D_1 or G_1 in which the distances of these products from the centres and its evaluations have been enclosed by two rectangles) and the remaining products, i.e. products C and D belong to the second cluster. The centres and the two clusters associated with this example are shown in Figure 2.10.1.

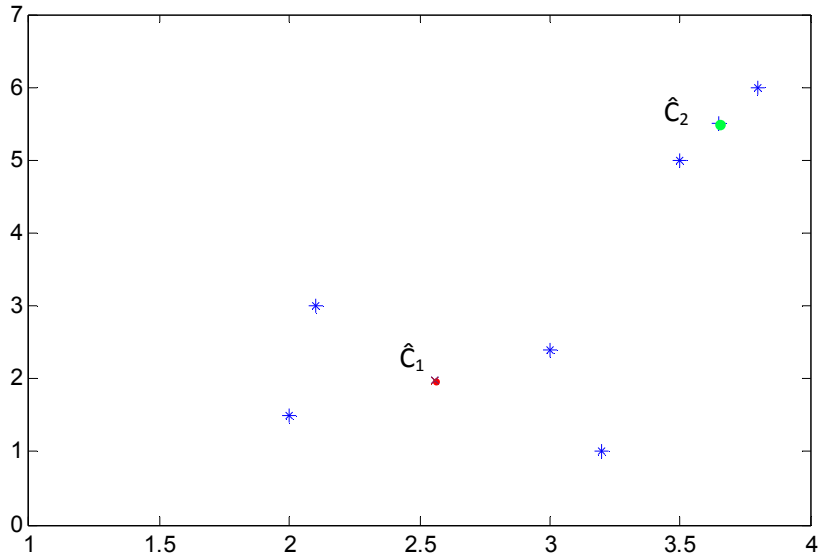


Figure 2.10.1: Two clusters and their centres.

2.10.2 The k-means algorithms

Here two different k-means algorithms are presented.

2.10.2.1 First k-means algorithm

1. *Initialise* - Select random, independent, vectors to be the initial centres for the RBF functions,

$$c_k(0); k = 1; 2; : : : ; K, \quad K < N$$

where N is the number of input vectors ,k is the number of centre and K is the number of clusters.

2. *Sample* - Select a vector at random from the set of input vectors

$$\{u_1, u_2, \cdots, u_N\}$$

3. *Centre allocation* - The centre best representing the cluster may be obtained using the minimum-distance Euclidean criterion:

$$k(u(i)) = \arg\min_k \|u(i) - c_k(i)\|$$

where $c_k(i)$ is the centre of the *k*th RBF at iteration i. $c_k(i)$ is positioned at the shortest distance from *u* and identifies the cluster to which *u* belongs to. Note that 'argmin' is used for cases in which there is no unique minimum. (argmin may return one of the minima randomly if there is more than one minimum.)

4. *Update* - Change the centres using the update rule.

$$c_k(i+1) = \begin{cases} c_k(i) + \alpha\left(u(i) - c_k(i)\right) & \text{if } k = k(u) \\ c_k(i) & \text{otherwise} \end{cases}$$

where is α suitable learning-rate in the range $0 < \alpha < 1$.

5. *Continue* - Increase i by 1, return to step 2, then repeat until no noticeable change occurs in the position of the centres. This is equivalent to minimising the objective function

$$F = \sum_{k=1}^{K} \sum_{q=1}^{Q_k} \left\| u_q^k - c_k \right\|^2$$

where K is the number of clusters and Q_k is the number of the input vectors with the elements u_q^k belonging to the cluster k.

2.10.2.2 Second k-means algorithm

The second k-means algorithm present a similar procedure for obtaining a set of appropriate centres:

1. *Initialise* - Choose random, independent, vectors to be the initial centres for the RBF functions, $c_k(0); k = 1; 2; \ldots, K, \quad K < N$

where *N* is the number of input vectors.

2. *Centre allocation* - The centre best representing the cluster within which each input vector, u, is obtained using the minimum-distance Euclidean criterion:

$$k(u(i)) = \arg\min_k \left\| u(i) - c_k(i) \right\|$$

where $c_k(i)$ is the centre of the k^{th} cluster at the iteration.

3. *Continue* - Increase i by 1, return to step 2, then repeat 2 until all the inputs vectors have been allocated to a cluster.

4. *Update*- Calculate the centroid. If is u_{lq}^k the l^{th} element of the q^{th} vector assigned to centre k, the $(i+1)^{\text{th}}$ centroid (new centre) of the k^{th} cluster is given by

$$c_k(i+1) = \frac{1}{Q_k(i)} \left[\sum_{q=1}^{Q_1(i)} u_{1q}^k(i) \quad \sum_{q=1}^{Q_2(i)} u_{2q}^k(i) \quad \cdots \quad \sum_{q=1}^{Q_k(i)} u_{mq}^k(i) \right], \quad \sum_{k=1}^{K} Q_k = N$$

where $Q_k(i)$ is the number of input vectors $u_{lq}^k(i)$ assigned to the k^{th} centre at the i^{th} iteration.

2.10.2.3 The fuzzy k-means algorithm

To use an algorithm for the fuzzy k-means, first the following aspects should be considered:

(a) The membership functions are $\mu_n^k(u_n) \in [0 \quad 1]$

(b) For all u_n^k, $\sum_{k=1}^{K} \mu_n^k(u_n) = 1$; u_n^k is obtained from

$$\mu_n^k(u_n) = \left(\sum_{j=1}^{K} \left[\frac{\left\| u_n(i) - c_k(i) \right\|}{\left\| u_n(i) - c_j(i) \right\|} \right]^{\frac{2}{\phi - 1}} \right)^{-1}$$

where $\phi > 1$ is the fuzzy exponent determining the degree of fuzziness.

(c) The centroid update equation (CUE) is obtained from

$$c_k(i+1) = \frac{1}{\sum_{n=1}^{N} \mu_n^{k\phi}(u_n)} \left[\sum_{n=1}^{N} \mu_n^{k\phi} u_{1n}^{k\phi} \quad \sum_{q=1}^{N} \mu_n^{k\phi} u_{2n}^{k\phi} \quad \cdots \quad \sum_{q=1}^{N} \mu_n^{k\phi} u_{Mn}^{k\phi} \right]$$

Now the fuzzy *k*-means algorithm can be expressed as follows:

1. Initialise - Choose random, independent, vectors to be the initial centres for the RBF functions

$$c_k(0); k = 1; 2; \ldots; K, \quad K < N$$

where *N* is the number of input vectors.

2. Select a value for $\phi > 1$.

3. Assign membership degree: u_n^k is calculated for each input vector using CUE.

4. *Update* - New centres are calculated using CUE.

5. *Continue* - Increase i by 1, return to step 2, then repeat until no noticeable change occurs in the position of the centres. This is equivalent to minimising the objective function

$$F = \sum_{k=1}^{K} \sum_{n=1}^{N} \mu_n^{k\phi} \left\| u_n - c_k \right\|^2$$

The fuzzy k-means algorithm has the following properties:

- a better performance than the standard k-means algorithm.
- It is implemented in MATLAB® using the fuzzy logic toolbox.
- Neither of the methods is guaranteed to find the optimal centres.
- Different initial conditions will yield different centres.
- Selecting *N* initial centres (i.e. the entire set of input vectors) guarantees a zero value for the objective function.
- A compromise solution with considerably less centres is generally desired.

Note that for all algorithms, a value for the stopping criterion function (for example when the error is less than 0.001) is selected to give reasonable convergence.

2.11 Supervised selection of centres

Define the error measure

$$\varepsilon = \frac{1}{2}(t - y)^2$$

Write in terms of the RBF functions

$$\varepsilon = \frac{1}{2}(t - y)^2$$

$$\frac{1}{2}\left(t - \sum_{n=1}^{N} w_n \, g_n(r_n, \sigma_n)\right)^2$$

Differentiate w.r.t. w_n

$$\frac{\partial \varepsilon}{\partial w_n} = -g_n(r_n, \sigma_n)\left(t - \sum_{n=1}^{N} w_n\, g_n(r_n, \sigma_n)\right)$$
$$= g_n(r_n, \sigma_n)\left(\sum_{n=1}^{N} w_n\, g_n(r_n, \sigma_n) - t\right)$$

$$\frac{\partial \varepsilon}{\partial w} = -g_n(r_n, \sigma_n)\left(t - \sum_{n=1}^{N} w_n\, g_n(r_n, \sigma_n)\right)^2$$
$$= g_n(r_n, \sigma_n)\left(\sum_{n=1}^{N} w_n\, g_n(r_n, \sigma_n) - t\right)^2$$

Obtain the weight update equation for the gradient descent algorithm

$$w_n(i+1) = w_n(i) - \eta_1 \frac{\partial \varepsilon}{\partial w_n(i)}$$
$$= w_n(i) - \eta_1 g_n(r_n(i), \sigma_n(i))\left(\sum_{n=1}^{N} w_n\,(i) g_n(r_n(i), \sigma_n(i)) - t\right)$$

Define the initial coordinates for the centres

$$C(0) = \begin{bmatrix} c_1^T(0) \\ c_2^T(0) \\ \vdots \\ c_N^T(0) \end{bmatrix} = \begin{bmatrix} c_{11}(0) & c_{12}(0) & \cdots & c_{1M}(0) \\ c_{21}(0) & c_{21}(0) & \cdots & c_{2M}(0) \\ \vdots & \vdots & \ddots & \vdots \\ c_{N1}(0) & c_{N2}(0) & \cdots & c_{NM}(0) \end{bmatrix}$$

Calculate the update equation for the centres

$$C(i+1) = c(i) - \eta_2 \frac{\partial \varepsilon}{\partial C(i)} = C(i) - \eta_2 \begin{bmatrix} \frac{\partial \varepsilon}{\partial C_{11}(i)} & \frac{\partial \varepsilon}{\partial C_{N1}(i)} & \cdots & \frac{\partial \varepsilon}{\partial C_{1M}(i)} \\ \frac{\partial \varepsilon}{\partial C_{21}(i)} & \frac{\partial \varepsilon}{\partial C_{22}(i)} & \cdots & \frac{\partial \varepsilon}{\partial C_{2M}(i)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon}{\partial C_{N1}(i)} & \frac{\partial \varepsilon}{\partial C_{N2}(i)} & \cdots & \frac{\partial \varepsilon}{\partial C_{NM}(i)} \end{bmatrix}$$

Differentiate with respect to c_n

$$\frac{\partial \varepsilon}{\partial c_n} = \frac{\partial \varepsilon}{\partial y}\frac{\partial y}{\partial c_n}$$
$$= \frac{\partial}{\partial y}\left(\frac{1}{2}(t-y)^2\right)\frac{\partial y}{\partial c_n}$$
$$= -(t-y)\frac{\partial y}{\partial c_n}$$

Differentiate with respect to σ_n

$$\frac{\partial \varepsilon}{\partial \sigma_n} = \frac{\partial \varepsilon}{\partial y}\frac{\partial y}{\partial \sigma_n}$$
$$= \frac{\partial}{\partial y}\left(\frac{1}{2}(t-y)^2\right)\frac{\partial y}{\partial \sigma_n}$$
$$= -(t-y)\frac{\partial y}{\partial \sigma_n}$$

where

$$\frac{\partial y}{\partial \sigma_n} = \frac{\partial}{\partial \sigma_n}\sum_{n=1}^{N} w_n\, g_n(r_n, \sigma_n)$$
$$= \frac{\partial}{\partial \sigma_n}\sum_{n=1}^{N} w_n\, g_n(r_n, \sigma_n) e^{-\|u-c_n\|^2/\sigma_n^2}$$
$$= \frac{2w_n\|u-c_n\|^2}{\sigma_n^3} e^{-\|u-c_n\|^2/\sigma_n^2}$$

Let

$$\alpha_n = \frac{2\eta_2}{\sigma_n^2} w_n g_n(r_n, \sigma_n)$$

Then

$$C(i+1) = c(i) + 2\eta_2(t-y)\begin{bmatrix} \frac{w_1(i)}{\sigma_1^2(i)} g_1\big(r_1(i),\sigma_1(i)\big)\big(u-c_1(i)\big)^T \\ \frac{w_1(i)}{\sigma_2^2(i)} g_2\big(r_2(i),\sigma_2(i)\big)\big(u-c_2(i)\big)^T \\ \vdots \\ \frac{w_N(i)}{\sigma_N^2(i)} g_N\big(r_N(i),\sigma_N(i)\big)\big(u-c_N(i)\big)^T \end{bmatrix}$$

$$= \begin{bmatrix} c_1^T(i)(1-\alpha_1(i)) + \alpha_1(i)u^T \\ c_2^T(i)(1-\alpha_2(i)) + \alpha_2(i)u^T \\ \vdots \\ c_N^T(i)(1-\alpha_N(i)) + \alpha_N(i)u^T \end{bmatrix}$$

Simplify

$$\frac{\partial \varepsilon}{\partial \sigma_n} = \frac{2w_n(t-y)\|u-c_n\|^2}{\sigma_n^3} e^{-\|u-c_n\|^2/\sigma_n^2}$$

The update equation

$$\sigma_n(i+1) = \sigma_n(i) + \eta_3 \frac{2w_n(i)(t-y(i))\|u-c_n(i)\|^2}{\sigma_n^3(i)} e^{-\frac{\|u-c_n\|^2}{\sigma_n^2(i)}}$$

$$= \sigma_n(i) + \eta_3 \frac{2w_n(i)(t-y(i))\|u-c_n(i)\|^2}{\sigma_n^3(i)} g_n\big(r_n(i),\sigma_n(i)\big)$$

To perform the iterations, the following equations are required:

$$w_n(i+1) = w_n(i) - \eta_1 \frac{\partial \varepsilon}{\partial w_n(i)}$$

$$= w_n(i) - \eta_1 g_n(r_n(i),\sigma_n(i))\left(\sum_{n=1}^{N} w_n\,(i) g_n(r_n(i),\sigma_n(i)) - t\right)$$

$$C(i+1) = c(i) - \eta_2 \frac{\partial \varepsilon}{\partial C(i)} = C(i) - \eta_2 \begin{bmatrix} \frac{\partial \varepsilon}{\partial C_{11}(i)} & \frac{\partial \varepsilon}{\partial C_{N1}(i)} & \cdots & \frac{\partial \varepsilon}{\partial C_{1M}(i)} \\ \frac{\partial \varepsilon}{\partial C_{21}(i)} & \frac{\partial \varepsilon}{\partial C_{22}(i)} & \cdots & \frac{\partial \varepsilon}{\partial C_{2M}(i)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon}{\partial C_{N1}(i)} & \frac{\partial \varepsilon}{\partial C_{N2}(i)} & \cdots & \frac{\partial \varepsilon}{\partial C_{NM}(i)} \end{bmatrix}$$

$$\sigma_n(i+1) = \sigma_n(i) + \eta_3 \frac{2w_n(i)(t - y(i))\|u - c_n(i)\|^2}{\sigma_n^3(i)} e^{-\frac{\|u-c_n\|^2}{\sigma_n^2(i)}}$$
$$= \sigma_n(i) + \eta_3 \frac{2w_n(i)(t - y(i))\|u - c_n(i)\|^2}{\sigma_n^3(i)} g_n\big(r_n(i), \sigma_n(i)\big)$$

The performance of the supervised selection of centres is better than the other algorithms in producing optimal RBF networks. To use the algorithm one should

(a) Select centres and spreads at random.

(b) Calculate the weights, then iterate.

Note that the outputs and weights are dependent both on the centres and the spreads, so the output and weights will be computed *twice* in each loop.

2.12 Neural networks for control systems

In this section the use of neural networks in control, Elman recurrent neural network, time delay neural networks, and methods for implementing neural network controllers are studied. Dynamic neural networks require memory which allows the network to exhibit temporal behaviour. In control system design using neural networks, the network behaviour is not only dependent on present inputs, but also on past inputs. There are two major classes of dynamic networks:

(a) Recurrent Neural Networks (RNN)

(b) Time Delayed Neural Networks (TDNN).

2.12.1 Recurrent neural networks (RNNs)

The two most common RNN are as follows:

(a) The Elman network (Elman 1990): The hidden layer outputs are fed back through a one step delay to dummy input nodes. See Figure 2.12.1.

(b) The Jordan network (Jordan 1986): It is similar to the Elman network but it feeds back the output layer rather than the hidden layer. See Figure 2.12.1.

Note that the RNNs are not easy to train due to the feedback connections.

2.12.2 Time Delay Neural Networks (TDNNs)

TDNNs can learn temporal behaviour by using the present and past inputs using delay(s) in the input signal. There are no feedback terms, so it is easily trained with standard algorithms. The neural network architecture is usually a standard MLP but it can also be a

- RBF
- PNN (Probabilistic Neural Network)
- GRNN (Generalised Regression Neural Network)
- other feedforward network architectures.

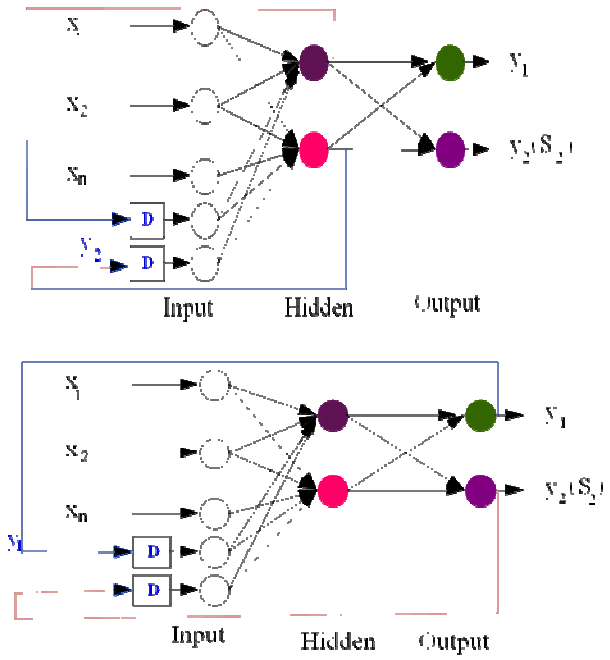


Figure 2.12.1: The block diagram of an Elman and Jordan recurrent neural Network, upper and lower diagrams, respectively.

A time delay neural network model is shown in Figure2.12.2.

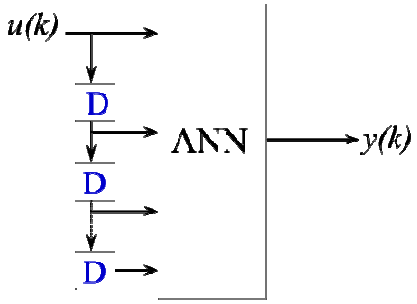


Figure2.12.2: A TDNN model.

Example 2.12.1: Let $x=[4,6,9,11,14,17,20,21,23,25]^T$ and time delay be 3. Then x_d is defined as

$$x_d=\begin{bmatrix} 4 & 6 & 9 & 11 \\ 6 & 9 & 11 & 14 \\ 9 & 11 & 14 & 17 \\ 11 & 14 & 17 & 20 \\ 14 & 17 & 20 & 21 \\ 17 & 20 & 21 & 23 \\ 20 & 21 & 23 & 25 \end{bmatrix}$$

Exercise 7.5.1: (a) Write a m-file MATLAB® to calculating x defined in Example 2.12.1 when delay are 2, 3 and 5 samples. (b) Assume that a noise signal is given by

$$f(x) = 0.05(1/2 - \text{rand}(1)) + 0.3\sin(2\pi t) + 0.5\cos(0.01\pi t)$$

for $100 \leq t \leq 200$. Consider $d = 5$ as a delay, $W = [0.1\ 0.3\ 0.5\ 0.1\ 0.3\ 0.2]^T$ and x_d the inputs obtained from signal $f(t)$. Using MATLAB® calculate $y = x_d W$ and plot(x_d , y) as well as plot(t_d , $f(x)$) in a single figure where $1 \leq t_d \leq \text{length}(f(x)-1)$. Note that this network filters the noise by calculating a linear weighted average. (c) Select two different delays and two appropriate weights

2.13 Neural network strategies

There are various methods for implementing neural network controllers including

(a) Supervised Control

(b) Direct Inverse Control

(c) Neural Adaptive Control

(d) Backpropagation Through Time

(e) Adaptive critic Methods

An appropriate method is selected based on available information on the system and desired performance. Supervised controllers are common and popular methods in which the neural network

is trained to perform the same actions as another controller (mechanical or human) for given inputs and plant conditions. Once the neural controller is well-trained, it replaces the controller. See Figure 2.13.1.

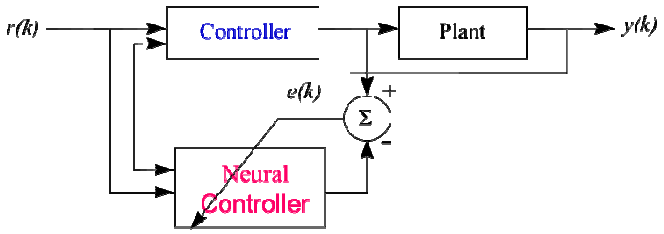


Figure 2.13.1: Supervised Neural Controller Training.

Neural network strategies and MATLAB® guideline for neural network and fuzzy logic can be found in text books such as Hines (1997).

In this section various neural networks, including single layer perceptron, MLP and GRBF have been introduced. The limitations of a single layer perceptron have been identified. A backpropagation algorithm has been presented to find appropriate weights which are required for tuning a network. The backpropagation method yields a global solution. To construct a GRB, a given data must be classified and the centre of each class (cluster) must be determined. This task can be achieved using a k-means algorithm. GRBF networks give a local solution to a problem.

References

[1] Simon Haykin, *Neural Networks and Learning Machines*, Third edition, Pearson *Prentice Hall: New Jersey, 2009.*

[2] Michael Negnevitsky, *Artificial Intelligence: A Guide to Intelligent Systems,* Second edition, Addison Wesley (Pearson Education Limited), England, 2005.

[3] Andries Engelbrecht*, Computation Intelligence: An Introduction, Second edition*, John Wiley & Sons, Ltd, England. 2007

[4] Michio Sugeno and Takehisa Onisawa, *Fuzzy Optimization and Decision Making* (Obituary: Professor Toshiro Terano), 4(4), 255-256, Netherlands: Springer Science+ Business Media, 2005.

[6] Hans-Jurgen Zimmermann, *Fuzzy Set Theory and Its Applications*, Massachusetts: Kluwer Academic Publishers, Fourth edition, 2001

[7] D Lowe, Adaptive RBF non-linearities, and the problem of generalisation, *The 1st IEE-International Conference on Artificial Neural Networks,* 171-175, London, 1989.

[8] C A Micchelli, Interpolation of scattered data: Distance matrices and conditionally positive definite functions, *Constructive Approximation*, 2, 11-22, 1986.

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

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

[11] D Wettschereck and T Dietterich, Improving the performance of RBF networks by learning centre locations, *Advances in Neural information Processing Systems*, 4, 1133-1140, San Mateo, CA: Morgan Kaufmann, 1992.

[12] L. Wesley Hines, *Fuzzy and Neural Approaches in Engineering,* New York: John Wiley & Sons, 1997

[13] J. Moody and C. Darken Fast learning in networks of locally-tuned processing units. Neural computation, vil1, pp 281-294. 1989.

[14] C. Bishop, *Neural Networks for Pattern Recognition,* Clarendon Press, Oxford, 1995

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

[16] Cybenko, Approximation by Superpositions of a Sigmoidal Function, *Math. Control Signals Systems*, 2, 303-314, 1989.

[17] J. W. Hines, MATLAB® supplement to Fuzzy and neural network in engineering, John Wiley & Sons, Inc, New York, 1997

2.14 Neural network using MATLAB

Neural network design is very much a heuristic science. The MATLAB® Neural Networks toolbox allows a reasonable selection of NN designs to be realised without resort to writing additional mfiles. However, design of an appropriate NN architecture for a particular problem may not always be straightforward. It is normally required to write mfiles which may help to find the best combination of number and type of neurons, the best weights and biases, and overall a learning algorithm.

2.14.1 Activation Functions

The MATLAB® NN toolbox consists of activation functions such that they can be used by a user readily. For example, a plot of the hardlim or heaviside function can be obtained as follows:

x= -5:0.1:5; plot(x, hardlim(x))

to . Similar graphs can be obtained for

purelin, logsig, tansig and radbas functions. MATLAB® assumes that the parameter α is equal to 1 in

$$y = \alpha x,\ \mathrm{y} = \tanh(\alpha\, x) = \frac{e^{\alpha x} - e^{-\alpha x}}{e^{\alpha x} + e^{-\alpha x}} =, y = \mathrm{logsig}(\alpha x) = \frac{1}{1+e^{-\alpha x}}$$

An adjustment to gradient must be made explicitly. For example, to plot

$$y = \mathrm{logsig}(2x) = \frac{1}{1+e^{-2x}}$$

the MATLAB® commands become: x=-5:0.1:5 ; plot(x,logsig(2*x)).

The synaptic weights are adjusted during the MATLAB® training process to optimise the effect of functions and therefore, the type of function must be appropriately selected. The situation is somewhat different for the RBF function.

2.14.2 MATLAB® demos

Demos which are available in the NN toolbox show that how one can use MATLAB® for designing a NN. To start the demo type:

demos

and click on the relevant tabs. Simple networks are presented in order to represent them graphically. The NN demos are appropriate for giving a favour of NN theory, especially the concept of local and global minima in weight space. Inputting a few different initialisation weights shows the degree to which NN training relies on stochastic i.e. random processes. In training a NN, if the first attempt is not successful, many successive trials may be required to obtain appropriate solutions.

The effect of optimising the number of hidden neurons is demonstrated well by ‘nnd11gn’. Selection of a suitable number of neurons is very important and it should be noted that a good network design should minimise the number of neurons.

2.14.3 MATLAB® nntool

A simple and user friendly GUI Network/Data Manager window can be generated with 'nntool'. Try it with the XOR function. Input

$$x = [0\ 0\ 1\ 1;\ 0\ 1\ 0\ 1], \qquad t = [0\ 1\ 1\ 0]$$

x is a 2×4 matrix, t a 2×4 row vector. The required NN must therefore be designed to have 2 input neurons and 1 output neuron. Click on 'Import' to import x as an input and t as a target. Then click on 'New Network' to name your network. Choose a network configuration using the 'Network Type' drop down menu. To MATLAB, a Perceptron is always a single-layer network with hardlim output neurons. MATLAB® calls MLP a Feed-Forward, Backpropagation Network. Each type of network has a limited number of Transfer and Learning functions associated with it. Click on the 'Get from input' drop down menu. Click on u gives an input range [0 1; 0 1]. The number of input neurons has automatically been set to 2. Clicking on View gives you a block diagram of your NN. Note, at this point, you have not indicated how many output neurons are required.

The default is 1, but you can change this to any (natural) number you wish. If it does not match the size of the target, however, training will not be allowed to commence. Return to the Network/Data Manager window, first click on your network name, then on Train. Select your input and target vectors. Click on 'Training Parameters'. Look at the default parameters and change some of these if required. Click Train Network and the training screen will appear. Check the response of the trained NN to x by returning to the Network/Data Manager window, clicking on Network Only: Simulate. Click on Simulate, specify u, then label the output with a new name. Clicking Simulate Network will produce a new output variable. Double clicking on it gives the output values of the trained NN. How close are they to the target values? You can export the outputs and errors to the MATLAB® command line workspace by first returning to the Network/Data Manager window then clicking on Export. Highlighting the output and error vectors followed by Export will copy them to the workspace. Experimenting with different NN architectures will soon convince you how important it is to gain practical experience in order to NN design effectively.

2.14.4 NN Design

A few of the NN architectures can be investigated using nntool. A MLP is first considered which MATLAB® calls this a Feed-forward, Backpropagation NN. To create this in MATLAB, input

$$x = [0\ 0\ 1\ 1;\ 0\ 1\ 0\ 1], \qquad t = [0\ 1\ 1\ 0]$$

```
net = newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingd')
```

This has created a new MLP with 2 inputs, 3 hidden tansig neurons and 1 linear output neuron. Input(1) has a range [-1, 2], input(2), [0, 5]. 'traingd' indicates that batch gradient descent i.e. standard BP is to be used for training. The semi-colon was deliberately omitted to print the list of network attributes. The same network can be generated using

```
net = newff(minmax(u),[3,1],{'tansig','purelin'},'traingd');
```

where minmax(u) finnds the input ranges of input u for you. Before training commences, it is good practice to randomise the pseudo-random number generator with the clock using

```
rand('state', sum(100*clock))
```

Otherwise, the same sequence of pseudorandom weights will be generated each time you start up MATLAB®. At this point, select the default settings and input

```
[net,tr] = train(net,u,t);
```

However, a more sophisticated approach is to adjust the training parameters first e.g.

```
net.trainParam.show = 50; % Shows training progress every 50 epochs

net.trainParam.lr = 0.9; % Learning rate parameter

net.trainParam.epochs = 1000; % Maximum number of epochs allowed for training

net.trainParam.goal = 1e-4; % Acceptable error at which training can be terminated
```

Once the above parameters have been set, type: traingd to display an explanation of the algorithm and show the defaults.

2.14.5 Training Algorithms

There is a large selection of training algorithms to choose from in MATLAB®. For MLP training (i.e. using newff, the most commonly used are listed below:

- traingd - Batch Gradient Descent. Standard BP algorithm.
- traingdm - Batch Gradient Descent with Momentum. BP with momentum.
- traingda - Variable Learning Rate. Adaptive learning rate attempts to keep the learning step size as large as possible while keeping learning stable.
- traingdx - Variable Learning Rate with Momentum. Does just what it says.
- trainrp - Resilient Backpropagation. Speeds up the BP process by focussing on the sign of the activation function derivatives rather than the magnitude. Only a modest increase in storage capacity is required.
- traincgf - Fletcher-Reeves Conjugate Gradient Algorithm. Searches by zigzagging towards minimum. Usually much faster than above algorithms and only require a little extra storage capacity.
- trainscg - Moller Scaled Conjugate Gradient. Combines the model-trust region approach with conjugate gradient algorithm. Less time consuming than traincgf.

- trainbfg - Broyden, Fletcher, Goldfarb, Shanno (BFGS) Algorithm. Achieves faster convergence using quasi-Newton (second-order secant) method. An approximate Hessian of size $N \times N$ is stored, where *N* is the number of free parameters in the NN.

- trainlm - Levenberg-Marquardt Algorithm. Achieves faster convergence by approximating Newton's method by a Jacobian matrix. Memory requirement increases with the number of NN parameters AND size of training data.

There are many more algorithms available. It is advised to try a linear model (LM) first which yield spectacularly better (1000 times faster than BP with momentum) results on networks of around 400 parameters using 3000 training pairs - any larger and LM is likely to fall down. BFGS should be tried next, with CG or RBP to follow. Try comparing the effectiveness of each algorithm with the same input/output data. Between each training run input

```
net=init(net);
```

in order to reinitialise net with a new set of random weights; alternatively go back to newff and give each NN a new label.

2.14.6 Parameter Optimisation

Note that the value of initialisation weights can make a big difference to the outcome of a training run. It is good practise, therefore, to repeat each run several times using the same training parameters, but with new initialisation weights, keeping a record of the mean squared error (MSE) for each run. The fitness of an architecture is then calculated as the average of the errors. If it is desired to find best combinations of values for learning rate (η) and momentum (α) parameters while keeping the NN architecture constant, it is necessary to use the same initialisation weights for each run, so that the training is only affected by the η and α values.

2.14.7 Pre- and Post-processing

Training is often made more efficient by re-scaling inputs and outputs to be within a range more suitable for the activation functions being used. The most common method is to normalise the inputs to be in a range [-1, 1], which is compatible with the values which are easily handled by logistic functions. The targets are re-scaled according to the functions used on the output layer. For example, if these are logsig, a rescale to [0.25, 0.75] ensures that the targets are well within the output region of the neurons. Use of an additional linear output layer obviates the requirement for output rescaling, but increases training times. MATLAB® has built in functions which carry out the normalisation calculations for you.

```
[pu,minpu,maxpu,pt,minpt,maxpt]=premnmx(x,t);
offset = 2;
gain = 0.25;
net = newff(minmax(u),[3,1],{'tansig','logsig'},'traingd');
net = train(net,pu,gain*(pt+offset));
```

net is now a network trained using the pre-processed input and output vectors. The offset and gain values have been chosen to suit the logsig output layer. To produce the network response to pu, type:

```
py = sim(net,pu)./gain-offset;

y = postmnmx(py,minpt,maxpt);
```

postmnmx post-processes the network output using the parameters created by premnmx. It is, of course essential that all of these are stored in order to rescale the outputs from net. Similarly, any new input vector must be pre-processed before being used as an input to net. The code is

```
punew = tramnmx(unew,minpu,maxpu);
```

2.14.8 Radial Basis Networks

MATLAB® provides two design functions for the basic RBF network: newrbe and newrb. The RBF NN is produced using the input

```
net = newrbe(u,t,s)
```

or

```
net = newrb(u,t,g,s)
```

The former assigns a RBF neuron to each input vector, thereby assuring zero training error. The value allotted to s determines the width of the area in input space to which each neuron responds. Each RBF function outputs the value 0.5 for an input vector u at a Euclidean distance of $s = \sqrt{\ln 2}/b \approx 0.8326/b$ from its centre, where b is the bias of the RBF. The larger the value for s, the greater the overlap between adjacent neurons. Finding the best value will be very much a matter of trial and error, inevitably depending on the magnitude of the elements of input vectors. Certainly, though, s should be greater than the average Euclidean distance between input vectors and less than the distance across the total input space. g is the error goal for newrb. newrb creates RBF neurons one at a time, using each input vector as a possible centre. The best centre i.e. the one producing the smallest network error, is chosen. If the resultant error is less than g, the NN is finished. Otherwise, another search through the remaining input vectors is made for the centre giving the next lowest error. The process is repeated until the error goal is achieved. If the goal is too small, you may end up with all vectors being used as centres, which is the same as newrbe, but taking much longer to design. newrbe has the advantage of being fast. The disadvantages are that the network has as many neurons as training vectors with the high probability of poor generalisation. newrb, conversely, can be very slow, due to the iterative process. As an alternative to the 'in-house' NN toolbox algorithm, try using the fcm command (which is located in the fuzzy toolbox) to implement a fuzzy k-means algorithm (MATLAB® prefers c- instead of k-). The difference between this and the standard k-means algorithm described in the lecture is that a fuzzy membership degree is assigned to each input vector based on its distance from a prospective centre. The vectors are still each assigned a cluster number, but with this algorithm, unless a vector is a centre, its membership degree within the cluster is < 1

(although maybe only to a very small degree). As the algorithm updates the centres, the membership degree for each vector is used as a weighting in the calculation. To implement, use

```
[cen,mem,ob] = fcm(u,k);
```

where k is the number of centres required. This command will output a matrix, cen, of centre co-ordinates, a matrix, mem, of membership degrees and a vector, ob, which records the objective function at each iteration. To calculate the output of the RBF layer due to an input u use

```
out{1} = radbas(netprod(dist(cen,u),s))
```

where s is a vector of spread values, calculated using $s = \sqrt{d_{\max}}/\sqrt{C}$.

The linear layer weights are calculated using

```
Wb = t/[out{1}; ones(1,lengthout{1}]
```

Investigate demorb1 and demorb3 for basic RBF network demos. demorgn1 and demopnn demonstrate the architecture of Generalised Regression Networks and Probabilistic ANNs, both of which are variations on the RBF network and can therefore be designed without the requirement for training. The former is useful for function approximation, the latter for classification of data.

Exercises

1. Investigate different networks to solve the AND, NAND, OR, NOR problems.
2. Create a 5×4000 matrix TT with elements from a random distribution. Calculate a second matrix U with a functional relationship to the first. Possibilities include exponential, sinusoidal or polynomial. Add a small amount of random Gaussian noise (use a suitable multiple of 'randn') to the second matrix, then use this as the input, with T as the target, to various NN configurations. Find the best architecture to describe the input-output mapping.

2.15 Creating a generalisable neural network using MATLAB®

NNs have become a popular addition to the control system arsenal. Part of their appeal is the ability to synthesize input - output relationship*s when l*ittle or no *a priori* knowledge of a system is available. However, if something *a*bout the mapping *is* known (typically an assumption of smoothness), it may be better to modify the NN learning to take this into account. Now you have been introduced to the concepts of generalisation and regularisation we can look at how to realise these concepts and optimise NN architectures using MATLAB®.

2.15.1 Bayesian Regularisation

Bayesian regularisation is best implemented in MATLAB® for networks with about 400 parameters using the training function trainbr, which combines the LM algorithm with Bayesian regularisation. This automatically changes the parameter in the equation

$$\mathrm{MSE} = \frac{\gamma}{M}\sum_{1}^{M}(t_i - y_i)^2 + \frac{1-\gamma}{N}\sum_{1}^{N} w_j^2$$

as training progresses, where *M* represents the number of training pairs and *N* the number of network weights. We will create a NN which approximates a noisy sine wave:

```
rand('state', sum(100*clock))

u = [-1:0.05:1];

t = sin(2*pi*u)+0.1*randn(size(u));

net = newff(minmax(u),[20 1],{'tansig','purelin'},'trainbr');

net.trainParam.show = 10;

net.trainParam.epochs = 100;

[net,tr] = train(net,u,t);

simu = sim(net,u);

figure

plot(u,t,'k *',u,simu,'k -')
```

If you run the above code, the training performance will be shown on 3 graphs. Errors are shown in terms of SSE and SSW (sum of squared error and sum of squared weights), respectively. The third graph shows the effective number of free parameters actually used in training the network. If this converges on a figure significantly less than the number used in your network design, it means you need to trim your network. The total error takes a little while to stabilise, as the algorithm initially minimises the NN weights at the expense of the output error, then increases the size of weights to minimise the overall error. The resulting output is notable for its smoothness.

For larger networks, it is required to replace parameters in existing MATLAB® training algorithms. This is implemented by first inputting

```
net.performFcn = 'msereg';
```

to change the MSE function to be of the form given at the start of this Section. The value for γ is then fixed for the complete training process by inputting

net.performParam.ratio = γ

where 0 < γ < 1 - e.g. for equal weighting between MSE and msw, set γ =0.5.

2.16 Cross validation: Early stopping

Cross validation is carried out by designing an architecture using an appropriate algorithm with a training set then afterwards calculating network error on a previously unseen validation set. For an algorithm using clustering, the optimal value for k is then considered to be that which produces the minimum error.
The original input data **u = [-1:0.05:1]** are re-sampled such that one in four points are selected **val.P = [-1:0.2:1]**.The validation output is then obtained by slightly altering the target using random variations: val.T = sin(2*pi*val.P) + 0.1*randn(size(val.P));
Training is implemented as usual, with the addition of the val structure. Levenberg-Marquart is used due to the small size of the network:

```
net = newff(minmax(u),[20 1],{'tansig','purelin'},'trainglm');

net.trainParam.show = 10;

net.trainParam.epochs = 100;

net.trainParam.mu = 1;

net.train.Param.mu_dec = 0.8;

net.train.Param.mu_inc = 1.5;

[net,tr] = train(net,u,t,[ ],[ ],val);

simu = sim(net,u);

figure

plot(u,t,'k *',u,simu,'k -')
```

The MSE value is low, but the resultant curve is not as smooth as that produced by Bayesian regularisation. Note that the validation set is sampled from across the whole range of the data. This is very important when using cross-validation. The parameter values chosen ensure that trainlm does not converge too rapidly. MATLAB® also recommends using trainscg and trainrp with early stopping, although in theory, any algorithm can be used.

2.16.1 Cross-validation for RBF ANNs

This is carried out by designing an architecture using an appropriate algorithm with a training set then afterwards calculating network error on a previously unseen validation set. For an algorithm using clustering, the optimal value for *k* is then considered to be that which produces the minimum error.

2.16.2 Principal Component Analysis

There are some cases where some or all of the input vector components are highly correlated. It is obviously advantageous in such circumstances to reduce the size of the vector, as this allows a corresponding reduction in the NN architecture. If the data is extensive, it will be time consuming to sift through each pair of components analysing covariance, so the method of Principal Component Analysis (PCA) is used instead. This uses eigentheory to calculate the relative importance of the orthogonalised input vectors - for a full explanation, see Haykin (2009). MATLAB® has a built in routine implementing PCA:

[un, meanu, stdu] = prestd(u);

con = 0.01;

[utrans, transMat] = prepca(un,con);

The first line normalises the input vectors to have zero mean and variance = 1. *con* is the contribution below which we require the vectors to be eliminated. If this is set e.g. to 0.01, principal components contributing less than 1% to the total variance of the data will be removed from the data. 'utrans' is the transformed input data, transMat the PCA transformation matrix, which must be used to transform all subsequent input data before applying this to the NN. The required code is

unewn = trastd(unew,meanu,stdu);

unewtrans = trapca(unewn,transMat);

y = sim(net,unewtrans);

PCA should only be used with input vectors to try to reduce the number of inputs used by the neural network in view to minimise the network size.

2.16.3 Optimising Network Size

The number of hidden neurons correct in a MLP is fundamental to the ability of the network to approximate a function. A simple but effective introduction to the concept can be found in MATLAB's nnd11gn command.

In many cases, architectures will be large and/or complex. To avoid tedious and repetitive human intervention, it is efficient to write an mfile which loops through the various architectures for you. The m-file should carry out at the least the following procedures:

1. Start with a small network, adding one neuron at a time to a hidden layer.
2. Repeat training a number of times with each architecture – Haykin (2009) suggests 20 repeats.
3. Save each trained NN to a structure.

4. Record the minimum MSE achieved during each training run OR the complete training run. A multi-dimensional matrix (*M by N* by *P by R)* should be used to save the result where *M* is the number of hidden layer neurons, *N* is the number of repeated training runs and *P* is the minimum MSE achieved and R is the number of training run.

5. Save the resultant matrix and the network structure to a .mat-file.

Time constraints may only allow a smaller number of training repeats, and, as the hidden layer grows, remember that each epoch will be correspondingly longer. If trainlm is first used on a small network, the computer memory may be insufficient and difficulties may be encountered as the network grows.

2.16.4 Statistical Analysis

The average of the minimum MSE achieved in each run is one of the possible indicator of a NN's suitability for use. The issue with such measure is that it may not accommodate for outliers and may lead to a solution wrongly affected by an unrepresentative large training error. This is especially the case with LM, as the training quite often terminates in a local optima. It is therefore informative to check the variance of each architecture's training results, and maybe reject the worst result if it is much worse than the rest. Using the minimum MSE from each ensemble run may work - it will depend on how many runs are used for each ensemble. The following MATLAB® command can perform regression analysis on the NN outputs compared with the targets for scalar outputs:

```
y = sim(net, u);

[a,b,r] = postreg(u,t);
```

where a, b are the slope and intercept of the linear regression line relating targets to NN outputs. For zero error, a =1 and b = 0. r is the correlation coefficient between outputs and targets. For zero error, r = 1.

2.17 NNs in Control

Plant identification for time-invariant plants is easily carried out using ANNs. The simplest design method is to use the sampled inputs and outputs from a plant, then train the plant model using these. If no time delays are implemented, this creates the familiar model of the form

$$y(t) = F(u(t))$$

this is what we have been working with so far. If, however, prediction of future plant output is required, so that the model is

$$y(t+\tau) = F(y(t),\ y(t-1), \ldots,\ y(t-n+1),\ u(t),\ u(t-1), \ldots,\ u(t-n+1))$$

with τ the control horizon, then the total number of inputs to the network is *n x m* + *n x q* where *m* is the size of an input vector and *q* the size of an output vector. The targets will need to be shifted by τ + *n* places in order to create a NN which predicts the plant output. If you choose to design a plant model where the output is only related to prior inputs, i.e.

$$y(t+\tau) = F(u(t),\ u(t-1), \ldots,\ u(t-n+1))$$

then MATLAB® is capable of building input time delays into the architecture. An example demonstrates the method. For instance,

```
net = newff(minmax(U),[5 1],{'tansig','purelin'},'trainbr');

net.inputWeights{1,1}.delays = [0 1 2 3];

net_tr=train(net, U, T)
```

produces a NN which models

$$y(t) = F(u(t),\ u(t-1),\ u(t-2),\ u(t-3))$$

If a single linear layer of neurons is used, this is equivalent to a finite impulse response (FIR) filter,

and is known as an ADALINE network. If the target data is shifted by τ places, a prediction horizon can be built into the NN, so that in the above example we would get a NN modelling

$$y(t+\tau) = F(u(t),\ u(t-1),\ u(t-2),\ u(t-3))$$

Note that MATLAB® cannot cope with inputting delays omitting a zero first entry e.g. [1 2 3] as a substitute for changing the targets. There is no reason for NN training to always be carried out online. As long as plant output can be sampled online, the only things limiting the NN weights from being constantly updated are the sampling and epoch periods. Online updating is useful when systems change with time. A common application of NN design is in modelling the plant inverse - the training input to the NN is a sequence of previous plant outputs. An identical NN is used to supply the input to the plant using the reference signal as input. When the output error between the two networks is zero, the plant output exactly equals the reference signal. see Figure 2.17.1 . The two ANNs are constantly updated on-line by monitoring the value of *e* and re-training them to avoid drift.

For those of you who wish to investigate MATLAB NN control facilities in more detail, try demos/Toolboxes/Neural Networks/Control Systems for some useful tutorials using Simulink.

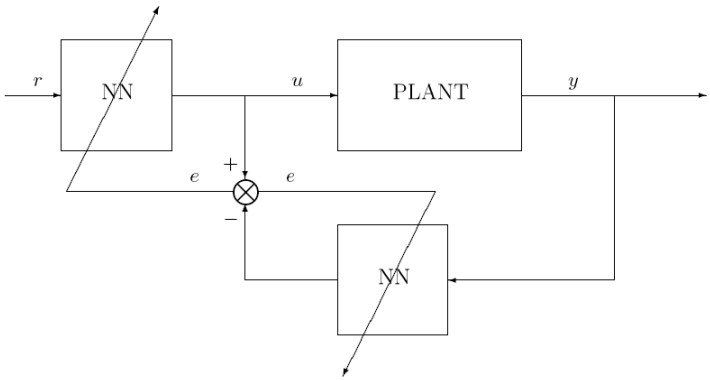


Figure 2.17.1: Plant controller using ANNs as inverse plant models

2.18 Appendix A M-files for neural network

2.18.1 Working with Backpropagation method using MATLAB

(BP_training.m)

```
% This file is used for training a neural network using the
% backpropagation method. In this mfile the learning rate is constant and
% can be selected in advance. If the desired solution is not obtained from
% the selected learning rate (lr), a loop must be included to obtain an
% appropriate learning rate and then the solution. For example
% lr=0.01
%for k=1:500
%here should be the backpropagation method, i.e. for i=1:maxepoch.... end;
 %lr=lr+0.01
 %end
% mfile inputs



% mfile output



% functions used


% algorithms - step by step description of what will be done



% constant and variable definitions

x = [0.2 0.6 1.7 -1.6;
  0.9 0.3 1.2 -02;
  0.2,0.1,0.4, 2];
t = [0.3 0.5 0.1 0.9];
[Nx,Dx] = size(x);        % x is the number of inputs
%Nx is the number of the avaible input set and Dx is the dimension of x,.
[Noutput,Dx]= size(t);
```

```
Nh=input('hidden_nodes=');     % Nh is the number of neurons (nodes) of
hidden
W=rand(Nh,Nx+1)/100;         % Initial weight W, the hiddeen layer weights
V=rand(Noutput,Nh+1)/100;       % Initial weight V, the oiutpt weights
maxepoch=1000;
Er=0.1;              % the totla errro E must be less than Er
lr=0.1;              %Learing rate
Te=zeros(1,maxepoch);
X=[ones(1,Dx); x];        % inputs including the bias
for i=1:maxepoch
 % The hiddenlayer output
  h = logsig(W*X);
  H=[ones(1,Dx);h];
  y=logsig(V*H);        %Th network output
  e=t-y;             % The error of the i iteration
  Te(i)= sum(e.^2)/2;      % The Total error up to i iteration.
  if Te(i)<Er; break; end
  y = logsig(V*H);
  delV= 2*lr*y.*(1-y).*e*H';
  V = V+ delV;        %updated weight
   del_W = 2*lr*( h.*(1-h).*(V(:,2:Nh+1)'*(y.*(1-y).*e)))*X';
  W = W+del_W;         %updated weight
end;clf
plot(nonzeros(Te));
title('Backpropagation Training');
xlabel('The number of epochs');ylabel('Sum of Squared Error')
if i<maxepoch;fprintf('Error goal reached in %i epochs.',i);end
```

3 Genetic algorithms

Olivier Haas, PhD, MSc, BEng, DUT, MIEEE

This document has been compiled from: Michalewicz Z., Genetic Algorithms + Data structures = Evolution programming, 2nd Ed, Springer Verlag, 1994; Daellenbach H.D., George J.A., *Introduction to Operations Research Techniques*, Allyn and Bacon, 1978, Reeves C. R., *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, 1993, Goldberg D.E., *Genetic algorithms in Search Optimization & Machine Learning*, Addison-Wesley, 1989 and Chipperfield, Fleming, Polheim, Fonseca, Genetic Algorithm toolbox for MATLAB, 1995 in addition to private communication with Colin Reeves, Coventry University, 1996.

The above references helped the author's research which was published in Haas O.C.L. (1999) Radiotherapy Treatment Planning: New System Approaches, Springer Verlag London, Advances in Industrial Control Monograph, ISBN 1-85233-063-5, and appears in various part of this document as examples and case studies. Readers interested in a more up to date review of applications of genetic algorithms in radiotherapy should consult Haas O.C.L. and Reeves C.R., (2006) Chapter 24, Genetic algorithms in Radiotherapy, Paton, Ray and Laura McNamara, eds. Multidisciplinary Approaches to Theory in Medicine, Volume 3. Elsevier Press. Book series is Studies in Multidisciplinarity Editor: L. McNamara, p447-482, 2006. Whilst (Haas, 1999) was the first to use multi objective genetic algorithm in radiotherapy, many researchers have since adopted the technique both in external beam radiotherapy and brachytherapy.

A selection of GAs books and e-resources at Coventry University Library that are of use include:

Coello Coello, Carlos A., Evolutionary algorithms for solving multi-objective problems /Carlos A. Coello Coello, Gary B. Lamont, David A. Van Veldhuizen. -- New York : Springer, 2007. xxi, 800 p. : ill. ; 24cm. (Genetic and evolutionary computation series)

Anil Menon (Ed), Frontiers of evolutionary computation [ELECTRONIC RESOURCE] Boston: Kluwer Academic, 2004. xxiii, 271 p. (Genetic algorithms and evolutionary computation)

Reeves, Colin R. Genetic algorithms [ELECTRONIC RESOURCE] :principles and perspectives: a guide to GAs theory /Colin R. Reeves, Jonathan E. Rowe. -- Boston: Kluwer Academic Publishers, c2003. vii, 332 p. : ill. ; 25 cm. (Operations research/computer science interfaces series ; ORCS 20)

The practical handbook of genetic algorithms, applications /edited by Lance Chambers -- Boca Raton, Fl. : Chapman & Hall, 2001

Lance Chambers (Ed) Practical handbook of genetic algorithms. 2,New frontiers -- Boca Raton, Fla. ; London : CRC Press, 2000 2 v. : ill. ; 25 cm.

Spears, William M. Evolutionary algorithms: the role of mutations and recombination /William M. Spears -- Berlin ; London : Springer, 2000 xiv, 222p. : ill. (cased) ; 24cm. (Natural computing series)

Michalewicz Z., Genetic Algorithms + Data structures = Evolution programming, 2nd Ed, Springer Verlag, 1994

Stephens C. R. and Poli R., 2004, EC Theory – "In theory" Towards a Unification of Evolutionary computation Theory in Frontiers of Evolutionary Computation**,** Kluwer Academic Publishers**,** 9781402077821

Recent advances in genetic algorithm research and other nature inspired search algorithms can be obtained from scientific papers published on data bases such as:
ScienceDirect http://www.sciencedirect.com/with full text access or
the engineering village 2 (without full text) http://www.engineeringvillage2.org

3.1 Introduction on heuristic techniques and current research

Heuristic methods are human-like in the sense that they are based on creativity, insight, intuition and learning.

Creativity can be perceived as the ability to create new solutions for a particular problem. This can be practically implemented via a purely random process that enables the exploration of new regions of the search space by introducing new solutions into the search procedure.

Insight enables the use of problem-specific features to improve the performances of the algorithms. For example, insight can be used to design problem-specific search operators to improve the performance of a heuristic technique.

Intuition can be viewed as an unconscious guided random process. Although it is difficult to build such a human notion into an algorithmic form, the intuition may be used to start from a particular region of the solution space where good if not optimal solutions are likely to be found. However this initial starting point is likely to result from the use of some problem-specific knowledge obtained via some form of learning.

The ability to learn is probably the last but not the least important feature of a heuristic process. Indeed, this ability to learn from regions of the solution space previously explored and feedback from the real world constitutes the most valuable component of a heuristic search strategy or indeed any other analytical search technique. This learning is usually implemented by making use of information concerning the 'appropriateness' of the solution to guide the search toward the most promising region of the solution space.

In order to create such heuristic techniques a set of rules that will help to discover a satisfactory solution to a specific problem is required. The design of these rules has been inspired by physical processes such as the annealing process for a technique called simulated annealing. Natural processes have inspired, and are still continuing to inspire a wide range of optimisation and problem solving algorithms. The natural evolution of species in a particular environment has led to the development of evolutionary computation (EC). Insect behaviour such as ant colony optimisation, honey bee optimisation, bee homing, and bee swarms have also been proposed. Particle swarm optimisation also takes inspiration from animal behaviour as prey such as fish schooling and bird flocking, or predators with hunting behaviour based algorithms. Human activities such as the musical processes of searching for a perfect state of harmony have also been employed with success.

This Chapter focuses on genetic algorithms (GAs), which are an example of EC, that explore the search space to select solutions to generate future candidate solutions by making use of a probabilistic process, where good solutions are very likely to be selected, but where bad solutions can still be selected in an attempt to find even better solutions.

3.1.1 Origin of Genetic algorithms

GAs belong to a subclass of guided random search techniques named evolutionary algorithms. GAs were first published by John Holland and his student Ken De Jong in 1975 at the University of Michigan. The aim of their research was to explain adaptive processes of natural systems and design artificial software systems that encompass the important mechanisms of natural systems. In order to design a robust method that would 'survive' in many different environments, i.e. be applicable to a wide variety of problems, a problem independent solving framework had to be developed. Getting some inspiration from the natural environment where the evolution of genes and chromosomes are used to determine the characteristics of many different species, the real problem was coded into a standard framework that was then used to solve the coded representation of the problem as opposed to the 'real' problem. Working on a coding problem implies that most of the information regarding a particular problem except for the cost or appropriateness of the solution is discarded. In addition the operators that provide the means to evolve or search for a solution are also problem independent.

This black box algorithm makes use of adaptive methods inspired by natural evolution and population genetics to solve various optimisation problems. GAs, similarly to natural evolution, evolve a group of individuals (called a population) corresponding to alternative solutions for a particular problem, by making use of concepts such as the 'survival of the fittest' whose principles were first stated by Charles Darwin in the *Origin of Species*. Although it is a random based process it is usually more efficient than purely randomised search. As such it can be considered as a broad efficient method for solving a wide range of problems as opposed to specialised methods that may perform better but with a much more limited scope. Because of its origin, the terminology used by the GAs community stems from the terminology used in genetics.

3.1.2 Genetic algorithms terminology

To understand a general description of GAs, it is necessary to explain some specific vocabulary that is employed in a field for which it was not created. Indeed GAs not only get their inspiration from the evolution of species and in particular genetics, they also employ a vocabulary usually reserved to genetics to describe their structures and operations. For example, an individual is a single element of a population referred to as a chromosome, which is in general a coded representation of a possible solution to the problem being researched. A chromosome is a structure that represents all the parameters involved in the solution of a particular problem. A chromosome or genome is composed of genes, where each gene encodes the value of a single parameter. Depending on the coding system used, a chromosome can be composed of binary bit strings, or arrays of integers or real numbers. In real coded GAs, each element of a chromosome is a gene composed of only one element. For binary coded GAs a gene is the coded representation of a variable and is usually composed of more than one element. A summary giving the meaning of the vocabulary borrowed from genetics is given in Table 3.1.

Table 3.1 Definitions of some genetic terms for an artificial genetic system such as GAs.

Genetics	Genetic Algorithms
Chromosome	Structure (i.e. string or vector) containing information on the set of features, variables, or parameters representing a possible solution to the problem considered.
Gene	Element of the chromosome that represents a single feature or variable of the problem considered (i.e. a binary digit in a binary string or a real valued number for real coded GA)
Allele	Value of the feature or variable represented by a gene (i.e. the value of the gene in the real world, a real number for example)
Genotype	Structure of the chromosome, a binary string for example
Phenotype	Decoded genotype, set of variables or parameters representing a physical structure that is a solution to the problem considered

Example

In radiotherapy, it is common practice to make use of several beam incidences that overlaps over the cancerous volume to be treated. Such technique allows an increase of the dose delivered to the cancerous tissues whilst keeping the dose to healthy tissues to an acceptable level. Radiotherapy treatment planning aims to find the best beam arrangement including beam modulation and direction.

Let us assume that it is aimed to find the best combination of beam directions using a genetic algorithm. Assuming that each beam can be rotated by 259 degrees with a resolution of one degree, the relation between the physical value of the beam direction (or angle) and its coded representation is illustrated in **Table 3.2**.

Table 3.2 Variables and coding for coplanar beam orientation optimisation in radiotherapy

Variable (gantry angle)	Beam 1	Beam 2	Beam 3	Beam 4
Physical value =phenotype	0.0°	130.0°	300.0°	270.0°
Encoding =genotype			Gene coding the variable 130	
Binary coding		010010	100101100	101110
Integer coding	0	130	300	270
Chromosome (binary coding)	01001010010110010111 0			Genotype
Chromosome (integer coding)	0 **130** 300 270			

Allele = value of element = 130

3.2 Common characteristics of GAs

Goldberg's definition of GAs states that GAs are 'an example of a search procedure that uses random choice to guide highly exploitative search through a coding of parameter space'. This refers to the original GAs also referred to as the canonical GAs (CGA). Since then, GAs have themselves been subject to evolution so that the original black box algorithm could be improved and/or adapted to more specific problems. These adaptations are referred to as hybridisation and include the use of real values instead of binary, and specialised operators designed to solve a particular problem. In particular, some GAs aimed at optimisation have suppressed the need to code the problem into discrete strings and instead work on the real variables themselves. For example some GAs use special recombination and mutation operators that work with the real variables to produce new solutions around and within the hypercube defined by the parents. Nevertheless, a common feature of all the different schemes is the population handling. Indeed, GAs maintain one or several populations of individuals that can be selectively replaced by individuals from a new generation. The use of a

population can be thought of as a search in parallel for a solution by investigating alternative solutions in a population. A description of the mechanisms involved in traditional GAs is given next.

Traditional Genetic Algorithm procedure

1. Generate an initial population of alternative solutions (chromosomes)

2. Evaluate the objective function for each chromosome and deduce the fitness values

3. Create new possible solutions (offspring)

a) Select parents from the population according to their fitness to produce children

b) Combine each pair of parents together to produce a pair of offspring

c) Calculate the objective function and fitness of the new solutions

4. Delete members of the population to insert some (or all) of the new solutions into the population

5. If the stopping criterion is satisfied (the population has converged or elements of the population are within acceptable limits), then stop and return the chromosome with the best ever fitness; otherwise go back to step 3

The most common methods involved in the population initialisation, the fitness calculation, and the recombination and mutation operators and finally the replacement methods are now described .

3.2.1 Fitness function

Optimisation algorithms require objective functions or cost functions to provide a measure of the quality of the solution in the solution space. GAs require the best solution to have the highest chance of being selected by the selection procedure. In order to comply with the requirements of a GA, it is necessary to transform the raw cost resulting from the evaluation of the objective(s) into a measure of relative fitness. For example in a minimisation problem the lower values of the objective are the better solutions. The lowest values of the objective function have to be transformed into highest probability of selection. Various schemes exist to deduce the fitness value from the raw value of the objective function.

3.2.1.1 Proportional fitness

Proportional fitness and $F(x_i)$ assignment is a common transformation used, it is given by

$$F(x_i) = \frac{f(x_i)}{\sum_{i=1}^{Npop} f(x_i)} \qquad 3.2.1$$

where N_{pop} is the size of the population, x_i is the vector representing a possible solution, also referred to as the phenotypic value of the individual and $f(x_i)$ is the objective function.

3.2.1.2 Linear scaling

Linear scaling is another commonly employed method which offsets the objective function value and scale it such that

$$F(x_i) = af(x_i) + b \qquad 3.2.2$$

Where a is a scaling coefficient, b is an offset used to ensure that the fitness value is non-negative, $F(x_i)$ is the resulting fitness value. In one version of fitness scaling, a maximum number of times an individual N_{TRIAL} can be expected to be selected for reproduction is chosen, typically N_{TRIAL}=2, to multiply the average fitness value and then the maximum fitness value is subtracted, such that in (**Błąd! Nie można odnaleźć źródła odwołania.**)

$$a = 1 \text{ and } b = N_{TRIAL}(\max f(x_i) - \min f(x_i)) - \max f(x_i) \qquad 3.2.3$$

From the expression given in (3.2.2) and(3.2.3) it can be shown that in the presence of a very fit individual and the selection process will tend to select only the best individual thus leading to premature convergence if the current best individual is not in a region of the search space where the global optimum is located. Conversely, if there is a very unfit individual, with a very low fitness, there will be a big difference in the scaled fitness between this very bad individual and all the other individuals, which in turn reduces the difference between good and average solutions in the current population. This may lead to stagnation or under-exploration of the search space.

3.2.1.3 Ranking methods

Rank based fitness assignment methods overcome the drawbacks of fitness scaling by minimising the reliance on extreme individuals in introducing a uniform scaling across the population. In fitness ranking, the individuals are sorted according to their objective value and the fitness is calculated from the position of the individual in the sorted list. To improve the focus of the search a so called 'Selective pressure' can be adopted to increase the probability of selection of the best individuals. This favours exploitation of current good solution at the expense of exploration of the solution space.

Genetic Algorithm Toolbox User's Guide p 1-9
One variable, MAX (or SP), is used to determine the bias, or selective pressure, towards the fittest individuals and the fitness of the others is determined by the following rules:
• MIN = 2.0 - MAX
• INC = 2.0 × (MAX -1.0) / Nind
• LOW = INC / 2.0
where MIN is the lower bound, INC is the difference between the fitness of adjacent individuals and LOW is the expected number of trials (number of times selected) of the least fit individual. MAX is typically chosen in the interval [1.1, 2.0]. Hence, for a population size of Nind = 40 and MAX = 1.1, we obtain MIN = 0.9, INC = 0.05 and LOW = 0.025. The fitness for linear ranking of individuals in the population may also be calculated directly as:

$$F(i) = 2 - SP + 2(SP - 1)\frac{i - 1}{N_{pop} - 1} \qquad 3.2.4$$

where i is the position of the individual in the sorted list, with i =1 is the worse individual and $i = N_{pop}$ is the fittest individual (with the lowest value of the objective when minimisation is considered). The

number of trials allocated to each individual depends only on its rank and varies linearly with the rank, thus the fittest individual may not have more that the pre-specified number of trials, ensuring that the risk of premature convergence is reduced.

Example:

Assuming that a population contains 5 individuals, and varying the selective pressure from 1.1 to 2 would lead to the following number of times each individual solution can be selected for reproduction, see Table 3.3. For example if the selective pressure is set to 1.5, then the best solution has 1.5 chances of being selected and the worse solution has only 0.5 chances of being selected, see Table 3.3.

Table 3.3 Illustrating the use of linear ranking with various selective pressures

Solution quality	Worse				best
Selective Pressure	1	2	3	4	5
1.1	0.9	0.95	1.0	1.05	1.1
1.2	0.8	0.9	1.0	1.1	1.2
1.3	0.7	0.85	1.0	1.15	1.3
1.4	0.6	0.8	1.0	1.2	1.4
1.5	0.5	0.75	1.0	1.25	1.5
1.6	0.4	0.7	1.0	1.3	1.6
1.7	0.3	0.65	1.0	1.35	1.7
1.8	0.2	0.6	1.0	1.4	1.8
1.9	0.1	0.55	1.0	1.45	1.9
2.0	0	0.5	1.0	1.5	2.0

For non linear ranking, the fitness is calculated using the following rule:

$$F(x_i) = \frac{N_{POP} p^{i-1}}{\sum_{j=1}^{N_{POP}} p^{j-1}} \qquad 3.2.5$$

where N_{POP} is the size of the population and p is the solution of roots of the following polynomial

$$0 = (\sigma - 1) p^{N_{POP}-1} + \sigma p^{N_{POP}-2} + \cdots + \sigma p + \sigma \qquad 3.2.6$$

with σ denoting the selective pressure.

Non-linear ranking enables the user to have a higher selective pressure than in linear ranking and thus increases the number of trials allocated to the fittest individuals with respect to the number of trials allocated to less fit individuals. The main difference with fitness scaling is that the number of reproductive trials is fixed with respect to the rank of the individual.

3.2.2 Pareto Ranking

Traditionally problems that are aiming to solve several objectives simultaneously have combined these objectives into a weighted sum of objectives. An alternative is to evaluate each of the objectives using different cost functions and then combine them using one of the techniques developed for multi objective genetic algorithm (MOGA) optimisation. Pareto ranking, is one of this method which makes use of the concept of 'dominance' to differentiate solutions according to the value or cost associated with each of their objectives. Pareto ranking aims to identify all the non dominated solutions which are all 'optimal', see **Błąd! Nie można odnaleźć źródła odwołania.** . A solution is said to be non dominated if no improvement can be achieved in the costs associated with any one of the objectives considered. In this work all the non dominated solutions are assigned an identical rank of 1. The rank of the dominated solutions is calculated as follows

$$r = 1 + N_{dominated} \qquad 3.2.7$$

where *r* is the rank, $N_{dominated}$ is the number of solutions by which the solution considered is dominated, i.e. the number of solutions which are better than the solution considered for at least one of the objective considered.

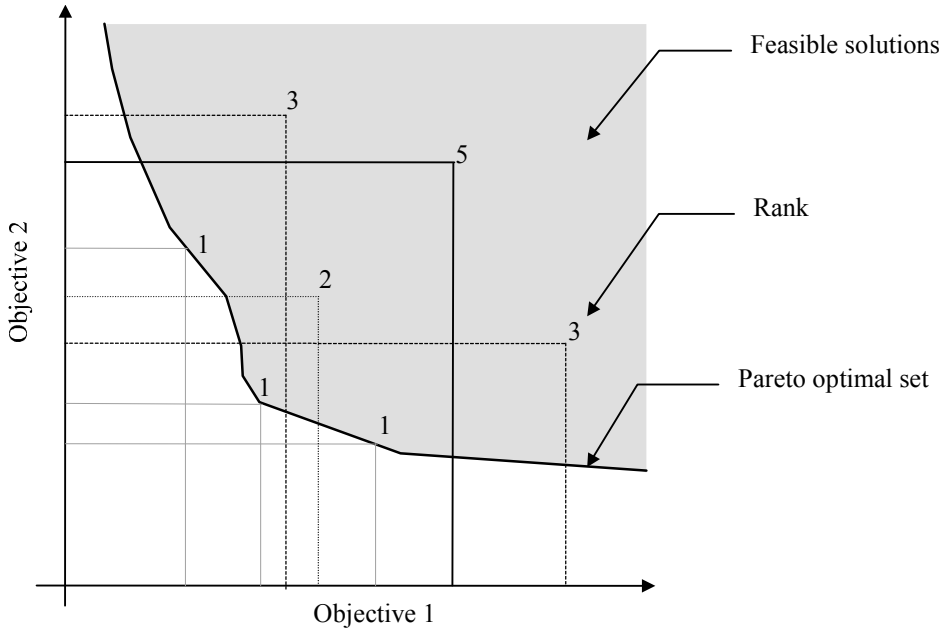


Figure 3.2.1.Illustrating Pareto-ranking approach for two objectives where the numbers 1 to 5 indicate the solution currently present in the population.

A consequence of the definition of rank adopted in this work is that all the solutions which are dominated by the same number of solutions (or all the non dominated solutions) are associated with an identical rank. For example, it can be observed in the illustrative example of Figure 3.2.1, that

there are two solutions of rank 3, which are dominated by two solutions that have a smaller cost for both objective 1 and objective 2. A solution of rank 5 is also present, which is dominated by four solutions, three of which happen to be of rank one and one of rank two. However, there is no solution of rank 4 because none of the solutions represented is dominated by three other solutions. This redundancy or absence of a particular rank is one of the main differences between Pareto ranking and traditional ranking methods.

3.2.2.1 Pareto ranking selection methods

The simplest method to select individuals from the current population is to use traditional rank based fitness assignment methods, with the sole difference being that the rank is not related to the position of an individual with respect to its objective but is now given by equation (1.7). Such an approach could lead, after a few generations, to a population composed solely of non dominated solutions and therefore to individuals having the same rank of unity, leading to an identical probability of being selected. At this stage, the selection of a new individual would become a random process. However, in most problems, solutions located at one extreme of the Pareto set may be very different to solutions located at the other. It is generally accepted that a combination of these two solutions is not likely to give rise to a better individual. Therefore it is necessary to differentiate individuals of similar rank, such that the combination of individuals that are more likely to produce improved solutions is favoured. This can be achieved, to some extent, by the so called 'niching' methods. Niching was first proposed by Goldberg and Richardson (1987)to reduce the number of candidate solutions in a region of the solution space by i) creating sub-populations that can evolve in parallel and ii) decreasing the fitness of individual that are close together in term of similarity of the chromosomes.

Radiotherapy treatment planning example

In the multiobjective optimisation work described in (Haas, 1999), the aim was to focus the search on region(s) of the search space that is/are likely to offer an acceptable solutions. However the shape of the solution space and, therefore, the trade-off that can be achieved is not common to all treatment plans in radiotherapy treatment planning (RTP). Consequently it may not be easy to identify the 'best', in some sense, region of the Pareto optimal set that is likely to offer the most favourable compromise. Therefore a scheme that enables a sufficient degree of freedom either to investigate all the Pareto optimal set (when no *a priori* information is available), or to concentrate on a particular region of this set has been adopted. It makes use of a decision maker (DM) to differentiate solutions with identical rank.

By keeping the preferences of the DM fixed, the search concentrates on the region indicated by the DM, whilst being able to produce non-dominated solutions in other regions of the search space; thus keeping a small level of exploration together with the exploitation of a particular region. Indeed, the selection being a heuristic process, although solutions favoured by the DM may have a higher chance of being selected, other non-dominated solutions can still be selected with a small probability.

The DM may also change its preferences as the search progresses in a deterministic or a heuristic manner. This enables the focus of the search on different regions of the solution space to change, therefore enabling a better exploration of the search space than the previous approach. The following approaches to articulate/combine the objectives were investigated:

Alternating the objectives randomly
: At each iteration, only one objective is considered by setting its objective weighting to unity, the other objectives weightings being set to zero. The objective considered at each iteration is selected randomly.

Alternating the objectives sequentially in an orderly manner
: Similar to the previous method, but where the objectives are chosen in a sequential manner.

Random combination of the objectives
: Each objective weighting is chosen randomly, such that the DM is a weighted sum of the objectives that evolves in a random manner. By making use of additional scalar weightings, it is possible to emphasise the relative importance of some objectives.

These techniques which have been used to direct the search as it progresses can also be used at a later stage to select the optimal solution from the Pareto optimal set. However, in many cases goals have been used to select the appropriate solution from the Pareto optimal set. The approach usually involves the setting of a minimal requirement for each objective as well as some desirable requirement for one objective and then selecting the optimal solution according to the cost for the other objectives. In radiotherapy treatment planning, there are many means to achieve very similar solutions. The availability of solutions on the Pareto set provides the ability for the clinician to add other criteria *a posteriori* to establish not only what is the optimal plan but also the most efficient, convenient and robust to deliver.

The DM used in conjunction with the MOGAs to guide the search combines the normalised cost of the various objectives, such that the fitness:

$$\text{Fitness} = r + \alpha\, f_{DM}(O_1, O_2, \ldots O_n) \qquad 3.2.8$$

where the scalar quantity $0 < \alpha < 1$ is chosen to emphasise the relative importance of the rank and the output of the decision maker f_{DM} which is of the form:

$$f_{DM}(O_1, O_2, \ldots O_n) = \sum_{i=1}^{n}\left(\lambda_i \frac{O_i}{M_i}\right) \Big/ \sum_{i=1}^{n}(\lambda_i) \qquad 3.2.9$$

where n, $\lambda_i \in \Re^+$, O_i and $M_i > 0$, $i = 1 \ldots n$, are, respectively, the number of objectives, the objective weightings, the values of each objective and the maximum value of a given objective in the current population.

An interesting feature of MOGAs is that objectives of a different nature that are difficult to combine into a single cost function can still be solved simultaneously. Such an approach may assist in the design of an optimal objective function. For example in radiotherapy treatment planning, selection of treatment plans were traditionally based on dose distributions. The development of radiological

testing enables to improve the prediction of the individual cell response to radiation and can thus be included as another objective to the dose based (referred to as physical) objective functions. In radiotherapy the evaluation of the cost function can take a long time if it is performed with a high degree of accuracy. It is therefore common to use a simplified cost function, quick to evaluate, such as using geometrical features to select the beam orientation (Haas, 1999) or using a simplified radiation beam model. To ensure that the solution is acceptable with a more accurate model and associated cost function, the more complex model, longer to evaluate can be used every few generations.

Having determined a measure of fitness it is now possible to enter the selection stage.

3.3 Selection

The selection process is one of the important processes for a GA. Originally the difference between GAs and other evolutionary algorithms such as evolution strategies was that the selection process in GAs was based on heuristic processes making use of probabilities of selection whereas evolution strategies used deterministic selection processes where the best elements of the actual population are always selected. The breeder genetic algorithm seems to be in between in the sense that a fixed percentage of the best elements of the population are always selected, but then mated randomly. The main traditional methods of selection are now reviewed.

The selection aims to determine the individuals from the current population that are most likely to produce good solutions. As opposed to evolutionary computation techniques that always select the best individuals, GAs introduce an element of randomness into the selection process by choosing individuals probabilistically. This implies that, for example, although the best individual may have a probability of 70% of being selected it may not be actually selected for breeding whereas a worse individual with a probability of 20% of being selected may be selected. This approach has been used to mimic natural behaviour of evolution that favours best individuals although it may not ensure their survival in the short term. Another way of approaching in the problem is to consider the selection as a method of avoiding convergence toward a sub-optimal solution, premature convergence or no convergence at all. Once these individuals have been selected they are used in a number of ways by the search operators to generate new solutions.

This phase is very important as if the most appropriate individuals are not selected, the algorithm may not find good solutions. The performance of selection schemes can be assessed with three measures: bias, spread and efficiency.

The bias is the difference between the individual's actual and expected selection probability. The aim of selection routines is to have zero bias such that the individual’s selection probability equals its expected number of appearance in the intermediate population.

The spread is the range in the possible number of times an individual is selected. Reducing the spread to a minimum may permit zero bias.

The spread is a measure of consistency and the bias a measure of accuracy of the selection method. To reduce GAs overall time complexity to a minimum, selection methods also need to be efficient.

There are several methods of selection available to select individuals from a population. The simplest approach to the selection mechanism is to consider each element of the population as a sector of a wheel. The sector size represents the probability that a particular individual has to be selected. This probability is calculated with a fitness function that transforms the values of the objective function into a probability of selection through a fitness value. For example in a minimisation problem, the best individuals have the lowest value of the objective function and the highest fitness value. The relative size of each sector with respect to the size of the wheel represents the percentage chance that a particular solution has to be selected for the next generation.

3.3.1 Roulette wheel selection

The simplest selection mechanism called roulette wheel selection (RWS) or stochastic sampling with replacement (SSR) can be view as spinning a wheel as many times as there are individuals to be selected and choosing one individual each time the wheel stops (see Figure 3.3.1).

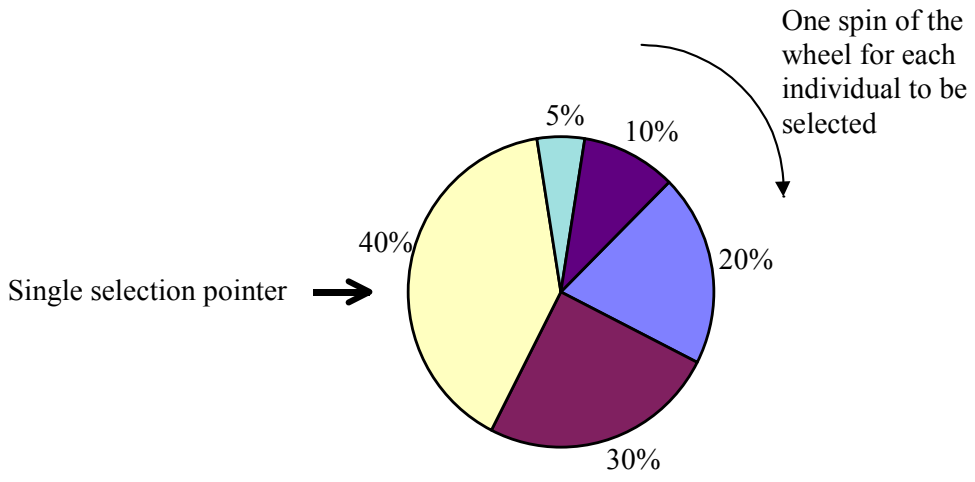


Figure 3.3.1: Illustrating the roulette wheel selection method

This method of selection is biased and the resulting intermediate population selected may be composed solely of the same individuals if such individuals has a probability of selection much greater that the rest of the population. This is not desirable as it may produce identical individuals with traditional combination operators such as crossover.

3.3.2 Remainder stochastic sampling with replacement

Remainder stochastic sampling with replacement (RSSR) selects most of the individuals deterministically according to the integer part *N* of their fitness value calculated such that

$$N = n\frac{f_i}{\sum_{i=1}^{n} f_i} \qquad 3.3.1$$

where N is a real number which integer part indicates the number of copies of the individual *i* is placed in the intermediate population. f_i is the fitness value of the i^{th} individual and n is the number of individuals in the intermediate population. Then in order to fill the remaining place in the intermediate population, the RWS is used with the fractional part of N.

This method of selection provides zero bias and a lower bounded spread.

3.3.3 Stochastic universal sampling

Stochastic universal sampling (SUS) is a single phase sampling algorithm using *n* equally spaced pointers starting from a position randomly generated to point to a particular individual. It can be visualised similarly to RWS as a number of pointers equally spaced around a wheel (see Figure 3.3.2). A single spin of the wheel can now provide simultaneously all *n* individuals to be assigned to the intermediate population. SUS has minimum spread and zero bias.

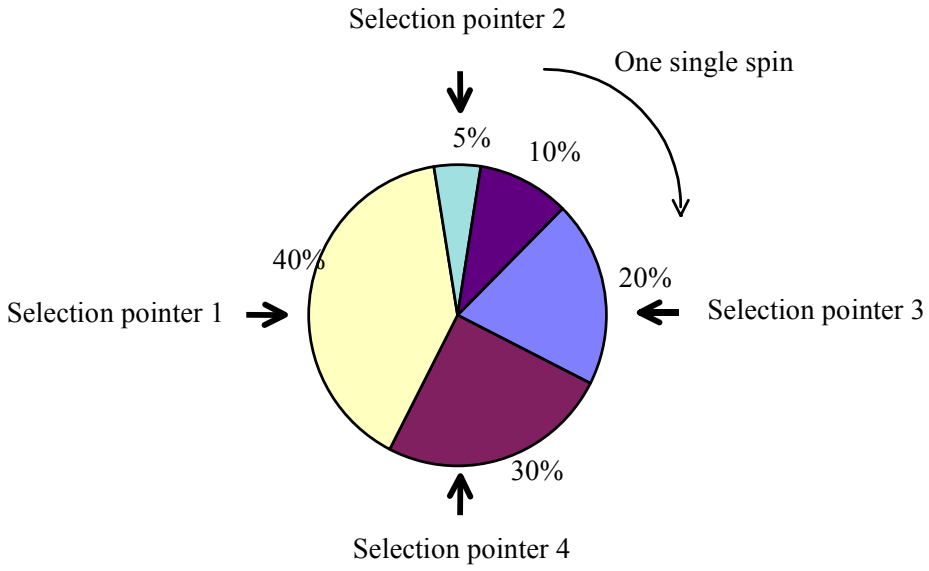


Figure 3.3.2: Illustrating stochastic universal sampling selection method

To ensure convergence to the optimal solution an elitist strategy should be adopted. This implies that the algorithm needs to keep the current best solution into the population. To save resources and minimise the likelihood of converging to a local optima and having the whole population filled with

the same individuals, it is possible to force each element in the population to be unique. Such a feature should be implemented. It normally requires checking the individual produce and if they are similar rejecting them before being evaluated by the cost function.

3.4 Genetic search operators

In the following Section, the main search operators used for canonical GAs as well as real or integer coded GAs are detailed. The genetic search operators can be divided into two categories: the recombination operators and the mutation operators.

3.4.1 Crossover operators

The crossover operator is used to recombine or exchange parts of the selected parents to produce new offspring.

3.4.1.1 Single point crossover

The simplest cross over operator usually represented with an 'X' exchanges parts of the parents C_{P1} and C_{P2} after a crossover point selected randomly to produce two offspring C_{O1} and C_{O2}. It can be schematically represented as follows:

Making use of vector notation, a simple crossover at the crossover point of index k between parents

$C_{P1} = [\, a_1\, a_2 \ldots a_n]$ and $C_{P2} = [\, b_1\, b_2 \ldots b_n]$ produces two offspring defined as follows:

$C_{O1} = [\, a_1 \ldots a_k\, b_{k+1} \ldots b_n]$ and $C_{O2} = [\, b_1 \ldots b_k\, a_{k+1} \ldots a_n]$, see Figure 3.4.1.

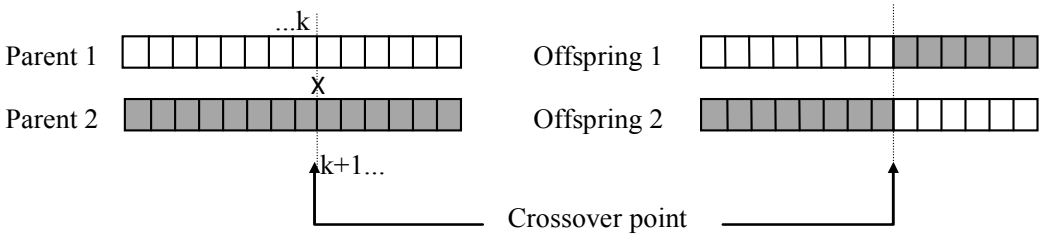


Figure 3.4.1 Illustrating single point crossover

3.4.1.2 Multi-point crossover

A multi point crossover can be seen as a generalisation of single point crossover, when each time a crossover point is encountered, the parts of the parents are exchanged after the crossover point.

In vector notation, a multi point crossover at the crossover points of index k, h and t between parents

$C_{P1} = [\, a_1\, a_2 \ldots a_n]$ and $C_{P2} = [\, b_1\, b_2 \ldots b_n]$ produces two offspring defined as follows:

$C_{O1} = [\, a_1 \ldots a_k\, b_{k+1} \ldots b_h\, a_{h+1} \ldots a_t\, b_{t+1} \ldots b_n]$

C_{O2} = [b_1 ... b_k a_{k+1} ... a_h b_{h+1} ... b_t a_{t+1} ... a_n], see Figure 3.4.2**Błąd! Nie można odnaleźć źródła odwołania.**.

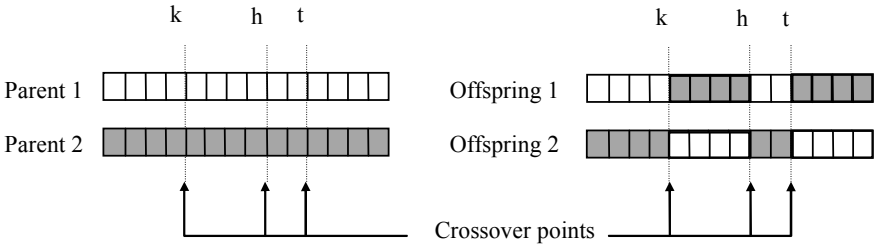


Figure 3.4.2 Illustrating multi-point crossover

3.4.1.3 Uniform crossover

Uniform crossover exchanges parts of each parent at positions randomly selected by a mask of the same length as the parents. The mask is composed of zeros and ones indicating which parent supplies the offspring with which element.

C_{O1} = [a_1 ... a_{k-1} b_k a_{k+1} ... a_{h-1} b_h a_{h+1} ... a_{t-1} b_t a_{t+1} ... a_{n-1} b_n]

C_{O2} = [b_1 ... b_{k-1} a_k b_{k+1} ... b_{h-1} a_h b_{h+1} ... b_{t-1} a_t b_{t+1} ... b_{n-1} a_n], see Figure 3.4.3

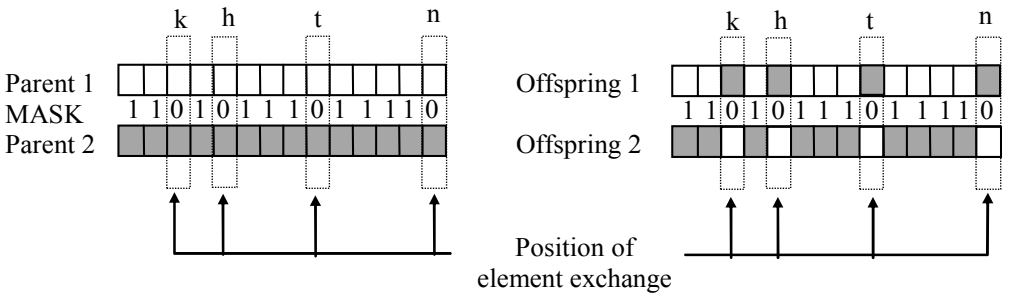


Figure 3.4.3: Illustrating uniform crossover

3.4.1.4 Shuffle crossover

The shuffle crossover acts similarly to the previous crossover operators except that prior to applying crossover all the components of the parents represented by the elements of the vectors C_{P1} and C_{P2} are shuffled randomly and the resulting offspring C_{O1} and C_{O2} are then subsequently un-shuffled. This reduces positional bias as elements of the vectors are randomly reassigned at each crossover operation.

3.4.1.5 Reduced surrogate

Reduced surrogate constrains the crossover to always produce new individuals (provided that the two parents are not identical) by restricting the location of the crossover points to positions where the successive components of C_{P1} and C_{P2} differs.

These crossover operators have been developed for canonical GAs where each individual is coded with a binary string/vector, but they can also be applied to integer or real number representations without any modification. In this case the reduced surrogate operator is also referred to as discrete recombination where the operator deals directly with the variables and exchanges the values of two variables between two individuals (parents).

3.4.1.6 Remarks on the crossover operators

The binary crossover operators outlined previously do not introduce any new information into individuals. Therefore, such crossovers alone may not guarantee that an optimal solution is found in an optimisation problem, especially when a high cardinality alphabet is used. For example in an alphabet containing 16 symbols it may be that each of the 16 symbols is not present in the population for each allele (each position in the chromosome). The crossover only changes part of the chromosomes, hence if a symbol corresponding to the optimal solution is not present in the population then the optimal solution will not be found using crossover alone. A chromosome does not introduce any new information into the population. If an element that is part of the optimal solution is not present in the actual population it cannot be created. This may explain why some have argued that the smallest value alphabet should be used to code a problem. Using a binary alphabet, both elements of the alphabet are likely to be present in the initial population of individual chosen at random. It is however good practice to make sure that this is the case when generating the initial population even if it means sacrificing some of the randomness aspect of the population generation.

3.4.2 Mutation operators

Whereas crossovers exploit the actual information contained in the elements of the population, traditional mutation introduces an element of chance by modifying heuristically elements of existing solutions. This is used to reach parts of the search space that may not be attainable by other means. The simplest mutation that is considered as being a traditional mutation operator is that of binary mutation.

3.4.2.1 Binary mutation

The binary mutation operator is the traditional mutation operator used in canonical GAs. Each component of the current solution represented by a binary string / vector composed of zeros and ones is examined and its value changed to its complement according to a small probability such that only a few elemental components of the individual selected for mutation are affected. The binary mutation is illustrated in Figure, where the selected parent C_{P1} = [1 1 0 1 0 1 1 1 0 1 1 1 1 0] is subject to a mutation of its 8th bit and becomes: C_{O1} = [1 1 0 1 0 1 1 0 0 1 1 1 1 0].

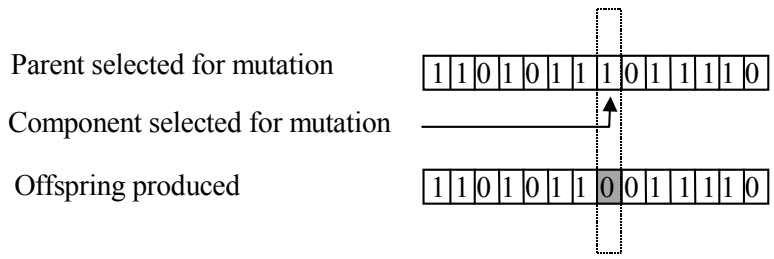


Figure 3.4.4 Illustrating binary mutation

3.4.2.2 Discrete mutation

Mutations operators are particularly useful for integer or real number representations, as the higher the cardinality of the alphabet *A* is, the smaller the chance of having all the possible values present in the initial population. Discrete mutation enables the introduction of new integer values into the current population. The effect is similar to binary mutation, except that the k^{th} component selected to be mutated takes a value randomly chosen between the range of possible values such that

$C_{P1} = [\ a_1 \ldots a_k \ldots a_n]$ produces $C_{O1} = [a_1 \ldots b_k \ldots a_n]$, with $b_k \in [0\ A\text{-}1]$.

For example for a parent C_{P1} = [1 45 25 30 96] coded with an alphabet *A* where |A| = 100, having its second element mutated may produce the following offspring

C_{O1} =[1 3 25 30 96], where the value of the mutated element is $3 \in [0\ 100]$.

So far, the operators considered have been designed for binary or integer valued populations. However, several GAs used for optimisation purposes use the real value of the physical variable directly and therefore require search operators specifically designed to accommodate real valued numbers. The searches operators described in the following Section have been specially designed for real coded GAs such as the breeder GA.

3.4.3 Search operators for real value GA

3.4.3.1 Line recombination

This operator combines each pair of parents C_{P1} and C_{P2} selected for recombination according to the following rule:

$C_{O1} = C_{P1} + \alpha\ (C_{P2} - C_{P1})$,

$C_{O2} = C_{P2} + \alpha\ (C_{P1} - C_{P2})$,

where a new value of α is randomly chosen for each pair of parents.

For example if

C_{P1} = [5.25 4.15 25.32 30.25 9.62] and C_{P1} = [4.25 27.25 5.32 3.25 19.62] and α=0.4 then the resulting offspring are:

C_{O1}=[4.85 13.39 17.32 19.45 13.62] and C_{O2}=[4.65 18.01 13.32 14.05 15.62]

3.4.3.2 Intermediate recombination

This operator combines each pair of parents C_{P1} and C_{P2} selected for recombination according to the following rule:

$C_{O1} = C_{P1} + \alpha_t\ (C_{P2} - C_{P1})$,

$C_{O2} = C_{P2} + \alpha_t\ (C_{P1} - C_{P2})$,

where a new value of α_t, $i \in [1, n]$ is a uniform random number chosen over the interval [-0.25, 1.25] for each component i contained in the parents of length n.

In Line recombination the scalar α remains constant for a pair of parents whereas for intermediate recombination α_t changes with each element of the chromosome.

3.4.3.3 Continuous mutation

Continuous mutation requires one parent C_{P1}, and operates according to the following rule:

$$C_{P1} = C_{P1} + r_i \sum_{i=0}^{i=m-1} \beta_i 2^{-i} \qquad 3.4.1$$

where r_i is the range of the mutation for the i^{th} variable, β_i is randomly chosen in the set {0,1}, with a probability $1/m$ of being 1 with m selected to be 20 in the GA toolbox used 'With m = 20, the mutation operator is able to locate the optimum up to a precision of the order of 2^{-19}.

3.5 GA: a theoretical perspective

The mechanism of binary coded GAs with binary mutation and single point crossover was originally explained with the schema theorem by Holland (1977). The schema theorem was developed to predict the propagation of schemata through generations and therefore the ability to reach different a region of the solution space containing the global optimum.

A schema (plural schemata) is a pattern of coding entities used in the representation of a chromosome. In addition to the entities used for coding (for example ones and zeros) a schema may include 'don't care' or wild card match symbols which can be represented by '*' and can take any values of the coding entities. In order to differentiate all the possible schemata, the order and the 'defining length' have to be introduced. The order is the number of fixed position (or non '*') in a schema. The defining length is the maximum distance between the outermost non-'don't care ' symbols.

Example: a chromosome composed of binary elements the string S = [* * 0 0 1 * 1 0 0] is referred to as a schema and the chromosome C = [1 0 0 0 1 0 1 0 0] matches the schema S. The order of S is $o(S)$ = 6 (there are six 1 or 0 in the string S. The defining length for S is $\delta(S)$ = 6, the first non * is at location 3 and the last at location 9, therefore $\delta(S)$ = 9-3 = 6..

The schema theorem has been referred to as the Fundamental Theorem of Genetic Algorithms, and it offers a prediction of the number of schemata expected in the next generation. Assuming that good individuals contain good schemata then these schemata are more likely to be passed on to the next generation. Making use of the schema theorem, Goldberg has derived the so called building block hypothesis which states that good solutions can be found by combining together 'building blocks', short length schemata, from different individuals to give an individual of higher fitness.

The Schema theorem has attracted both supporters and critics. A detailed review of GA theory is beyond the scope of this book, however, a nice overview of the issues raised by the schema theorem

and GA theory is given in Stephens and Poli, 2004. Readers interested in details of the schema theorem should consult Chapters 2 and 3 of Michalewicz, 1994 ‘GAs: How do they work’ and ‘GAs why do they work’.

The difficulty with the theory developed to date is that they assume simple GA and operators and do not account for the large number of operator that exists. The bottom line is that GAs have been shown to work to get to an acceptable solution if not the optimal one. From a practical perspective a good solution obtained within a reasonable amount of time is good enough. GAs should not be used for simple problems with single optima where a gradient descent will be more suitable. One of the advantages of genetic algorithm is that several cost functions can be designed and solved in parallel using multi objective genetic algorithms.

3.6 Worked example:

The following examples make use of the freely available Genetic Algorithm Toolbox for MATLAB® that was developed at the Department of Automatic Control and Systems Engineering of the University of Sheffield, UK. It was originally developed for MATLAB® v4.2, but has also been successfully used with subsequent versions up to and including MATLAB® 7 http://www.acse.dept.shef.ac.uk/cgi-bin/gatbx-download

Genetic algorithms are traditionally used to solve complex and large optimisation problems and are usually able to find good solutions relatively quickly. Finding an optimal solution is not guaranteed and this could take a significantly longer time. The following examples are probably too simple to tackle with GAs and could be solved more efficiently. The aim is to describe how GAs work using simple examples where the results are easy to replicate and analyse.

3.6.1 Optimisation of a quadratic cost function

Find the minimum of a function with one unknown variable:

$$f(x)= (x\text{-}52)^2 \qquad 3.6.1$$

Step 1: What are we trying to optimise?

We try to find the value of x which makes $f(x)$ equal to a minimum. In this case, it is obvious that the minimum of the function is $f(52)=0$. We will now see how the GAs goes about solving this problem. We will consider the initialisation of the GAs and only one iteration of the algorithm to obtain the value of the variable we are trying to optimise, namely in our case x.

The domain of the problem is arbitrarily chosen as $x \in [0\ 127]$. This means that x can take integer values between 0 and 127, corresponding to 128 distinct values that can be coded using 7 bits.

Remark

In most cases, it is necessary to design your own objective function to solve a problem. It is of the utmost importance to find the appropriate mathematical expression that actually solves the problem you are facing. If your objective function is incorrectly formulated you will obtain an incorrect answer. In this example, the cost function is $f(x)= (x\text{-}52)^2$.

To use a GAs we need to define

- the number of variable(s) to optimise, NoOfVar = 1;
- the range of values that the variable(s) can take MinMaxVar = [0;127];
- the resolution you want to achieve, i.e. how many decimal places RESOLUTION = 1;

Step 2: How do you code the variable: binary, Gray, k-nary, integer, real or other code?

What coding systems should be used and what are their relative advantages?

The coding system adopted depends on the problem considered and on the decision variables. Typical coding used are: binary, Gray, floating point, integer. There are also some problem-specific coding systems (usually in scheduling problems). Reflected Gray code, commonly referred to as Gray code, has the advantage over binary code that adjacent numbers differ only by one bit between. Hence an integer, coded using three bits, increasing from 3 to 4 will only have one bit different using Gray code whereas with binary coding all the bits are different. This means that solutions that are in the 'real world' close together may be far apart once coded. One method to convert Gray to binary is to XOR the bits one at a time, starting with the two highest bits, using the newly calculated bit in the next XOR. Gray to binary is obtained as follows and illustrated with the integers 0 to 3 in Table 3.4**Błąd! Nie można odnaleźć źródła odwołania.**:

$xg_1 = xb_1$, $\mathbf{xg_i = xg_{i-1}} \oplus xb_i$, for i > 1 where $\oplus$ is the addition modulo 2 (xor)

where the index i=1 to 3 in the example of Table 3.4, corresponds to the most significant bit (on the left) and index 3 the least significant bit on the right. For example for the binary code 100 the most significant bit is the first 1 and the least significant bit the last zero.

The inverse mapping from gray to binary is given as follows and illustrated with the integers 4 to 7 in Table 3.4:

$xb_1 = xg_1$, $\mathbf{xb_i = xb_{i-1}} \oplus xg_i$, for i > 1 where $\oplus$ is the addition modulo 2 (xor)

Table 3.4 Illustrating Binary and Gray coding

Binary	Integer	Binary reflective Gray
0**0**0	0	000 =[**0**, **0** $\oplus$0=**0**, **0**$\oplus$**0**=0]
001	1	**0**,0,1 =[**0**, **0** $\oplus$0=**0**, **0**$\oplus$**1**=1]
010	2	**0**11 = [**0**, **0** $\oplus$**1**=1, 1 $\oplus$ **0** = 1]
011	3	**0**10 = [**0**, **0** $\oplus$1=1, 1 $\oplus$ 1 = 0]
100 = [**1,1** $\oplus$1=**0, 0** $\oplus$0 = 0]	4	**1**10
101 = [**1**, **1** $\oplus$1= **0, 0** $\oplus$1=1]	5	**1**11
110= [**1, 1** $\oplus$0=**1,1**$\oplus$1=0]	6	**1**01
111 = [**1, 1** $\oplus$0=**1,1**$\oplus$0=1]	7	**1**00

Gray code ensures that adjacent genotypes correspond to adjacent phenotypes. Gray code has been shown to produce more optima than binary code. Hence it performs better in many cases.

The argument for using Gray or binary coding is losing strength owing to the improved numerical power of computers and the speed of floating point operations. Hence when a floating point encoding is required, floating point operator usually performs better.

In this simple illustrative example binary coding is used. The first task is to calculate the number of bits that are required to code our decision variables. Note that if more than one variable is used it is necessary to calculate the number of bits to code each variable. The length of the chromosome representing the variable depends on both the range of possible values that the variable can take and the precision or resolution of the variable.

In the example considered the solution lends itself well to using either integer, binary or Gray coded GA. Assuming binary coding is used the task is now to determine the number of bits required to code 128 different numbers (between 0 and 127), assuming a resolution of 1.

From a theoretical perspective the number of bits is given by

n = ceil (log2((Max - Min) / Precision + 1))

Note that ceil is used to round up to the next integer value to ensure that the entire variable can be coded. It may happen that some binary representations of this variable do not correspond to any possible solution. For example if you want to code a variable between 0 and 129 you do have to use 8 bits even though 125 values (integer between 130 and 255 will not correspond to a possible solution.

In our example:

n = log2(((127 - 0) / 1) + 1) = log2(128), meaning exactly 7 bits can be used.

The precision is then expressed as:

Precision = (Max – Min) / (2^n -1)

Checking the result we find that with n = 7, the precision is (127)/(2^7-1) = 127/(128-1) = 1

MinMaxVar = [0;127]; % x∈ [0 127].

% calculate the difference between the maximum and the minimum values = 127-0 = 127

diff(MinMaxVar) = 127

With a resolution of 1, exactly 128 different values are required to code the variables between 0 and 127.

The following MATLAB® code calculates the highest integer value you require to code your variable assuming you start at 0.

diff(MinMaxVar)/RESOLUTION) = 127 % = (127-0)/1

Remark:

If the resolution was 0.1 you would need 10 times more values: 127/0.1+1 = 1271, with the minimum value being 0.0 and the maximum value being 127.0

Using the MATLAB® command dec2bin gives:

dec2bin(127,7) = 1111111 % converts a decimal number to a binary string

dec2bin(diff(MinMaxVar)) = 1111111 % converts a decimal number to a binary string

dec2bin(diff(MinMaxVar)/RESOLUTION) = 1111111 % assuming a RESOLUTION of 1

Calculating the length or the maximum size of this vector of 1 gives the number of bits required to code the values between 0 and 127

NoOfBits = max(size(dec2bin(diff(MinMaxVar)/RESOLUTION)));

`str = dec2bin(d,n)` % produces a binary representation with at least `n` bits

To code 0 you just need to have all the bits equals to zero: dec2bin(0,7)= 0000000

To code 127 you need 7 bits all set to the value 1: dec2bin(127,7)= 1111111

To code 5 using 7 bits gives: dec2bin(5,7)= 0000101

Examples

i) How many different values would you need assuming that the range of your variables is between 0 and 64 and you need a resolution of 0.05?

Answer:

Number of values to code: 64/0.05 + 1 = 1281

Values can range from 0, 0.05, 0.1, …, 1279.95, 1280

length(dec2bin(1280)) = 11

ii) Using Matlab, calculate the size of the chromosome and the value of the chromosome for a variable x = 32.05 required to be coded with a resolution of 0.05 units.

Answer:

number of bits required assuming a resolution of 0.05:

Number of elements to code: (Max - Min) / Precision = 32.05/0.05 = 641

n = ceil (log2(641) + 1) = 11

Maximum value of variable with 11 bits $2^{11} = 2048$

dec2bin(32.05/0.05,11) = 01010000000

This coding process should be repeated for each variable to be optimised. If we have more than one variable the total chromosome length is then the sum of the lengths required to represent each variable. For example assuming that one variable requires 6 bits and another one requires 4 bits, the chromosome length would be 6+4=10 bits.

Implementation using the GAs toolbox

```
len = rep([NoOfBits],[1, NoOfVar]); % Len=7 in this example

% In the following code we specify the minimum and maximum value that can be coded.

lb = rep([MinMaxVar(1)],[1 NoOfVar]); % minimum value = 0 (in this example)

lu = rep([MinMaxVar(2)],[1 NoOfVar]); % maximum value = 127 (in this example)

 % Choose binary or gray code to represent the variables 0=binary, 1= Gray

code = 0*ones(1,NoOfVar);
```

Step 3: Initialise the population

What is the most appropriate size for the population (Npop)?

The population size is one of the most important parameters and will depend on the coding employed. The probability for all alleles to be present (1 and 0 at each position in the chromosome) can be calculated using: P = (1-(1/2)^(Npop-1))^ len

Binary or Gray coded GAs have typical population sizes in the range 30 to 200. However, it is still very much a trial and error process for new types of problems. I have found that small population sizes work well with real value coded operators. Some authors, however, suggest that micro GAs (population of 5 individuals) work well in combination with specific recombination operators with appropriate mutation and crossover rates. Note that the larger the population the more 'rejects' there are likely to be at the start of the search. This could significantly slow down the algorithm, especially if the objective function requires a long time to evaluate. The advantage of larger population sizes is that, for binary and Gray coded GAs, you can almost guarantee that you will have both a 1 and a 0 at each position in the Chromosome, therefore ensuring that all possible solutions can be reached using only crossover.

Create a population composed of a number of candidate solutions coded as chromosomes. This population will be the starting point of your search and care should be taken to initialise with appropriate estimates of the solution if possible. Inserting known good solutions is referred to as seeding. Note that if the initial solution contains a local optimum, the GAs may converge very quickly to this local optimum and fail to find a global optimum. Such behaviour is referred to as premature convergence. In the toolbox a standard approach adopted when the user has no knowledge about the problem to be solved is to randomly generate the values 1 and 0 for binary or Gray coding. I

would suggest starting with a population of 50 randomly selected and checking that you have a 1 and a zero at each location.

To illustrate the process whilst keeping the algorithm tractable a population of 10 individual was selected, representing a 10/127*100 = 7.9% of the possible solutions.

You can check that with a population of 10 there is 98.64% chance for both alleles to be present. P=(1-(1/2)^(10-1))^7 =98.64

Smaller populations can be generated if the presence of alleles is not left to 'chance' but designed by the user.

% Initialise population 10 different chromosomes in the population

Chrom = crtbp(NIND, NoOfVar*NoOfBits);

crtbp generates random numbers between 0 and 2 (for binary)

rand(Nind,Lind).*BaseV(ones(Nind,1),:)

1.93 0.85 1.89 0.03 0.11 1.81 1.01	1 0 1 0 0 1 1
1.24 1.67 1.56 1.72 0.11 0.21 0.54	1 1 1 1 0 0 0
1.39 1.46 1.41 0.15 0.30 1.03 0.20	1 1 1 0 0 1 0
1.44 0.72 0.21 1.33 0.03 0.28 1.01	1 0 0 1 0 0 1
0.69 0.90 0.77 1.00 0.87 1.11 1.17	0 0 0 1 0 1 1
1.03 0.77 1.18 0.43 1.66 0.01 1.52	1 0 1 0 1 0 1
1.11 1.55 0.91 1.14 1.23 1.53 0.16	1 1 0 1 1 1 1
0.31 1.46 0.10 0.24 1.04 1.69 1.32	0 1 0 0 1 1 1
1.12 0.86 0.45 1.34 1.72 1.83 1.03	1 0 0 1 1 1 1
1.38 1.38 1.66 1.19 0.19 1.97 0.34	1 1 1 1 0 1 0

then round down to the nearest integer floor(rand(Nind,Lind).*BaseV(ones(Nind,1),:))

Step 4: calculate the value of the objective for the initial population

Having generated a set of random solutions, the variables are then decoded so that they can be used in the objective function

MyVAR = bs2rv(Chrom,FieldD)

The objective function calculate the cost associated with each variable.

ObjV =ga_objectivefunction_m23mse(MyVAR)

Where x is represented by the variable MyVAR

f(x)= $(x-52)^2$ is implemented in MATLAB® as COST = (x-52).^2

The best value is calculated; in this particular run of the GAs it corresponded to Chromosome number 8 whose coded representation is equivalent to the integer value 46, recall that 52 is the optimal solution.

[MinValue,ChromosomeNo]=min(ObjV)

MinValue = 36, ChromosomeNo = 8

Here the minimum value of the objective function is 36 corresponding to x = 46 (closest to 52)

Step 5: Select the good solutions for recombination

Step 5.1 Calculate the fitness

Linear ranking and a selective pressure SP=2 is assumed. This means that there is a greater emphasise for selecting good individuals. The fitness value for linear ranking is calculated as follows:

$F(i) = 2 - SP + 2(SP-1)\frac{i-1}{N_{POP}-1}$ with i=1 being the worse individual and $i=N_{POP}$ the fittest

% Calculate the fitness using selective pressure of 2 and linear ranking [2 1]

% for 1 population

FitnV = ranking(ObjV,[2 0],1);

% calculate all possible fitness values

RFun = 2*[0:Nind-1]'/(Nind-1);

% Sort solutions in descending order

```
[ans,ix] = sort(-ObjVSub(Validix))

Sorted = ObjVSub(ix)

% Allocate the fitness according to rank in the sorted list
```

In the example considered the following chromosomes were evaluated and gave rise to the cost value f(x) as well as the fitness value FitnV. The underlined solutions indicate chromosomes selected for recombination.

	Binary string							x	f(x)	FitnV	Quality
(1)	1	**1**	**1**	**0**	**1**	**0**	1	117	4225	0	worse
(2)	**0**	0	**1**	1	**1**	1	**0**	30	484	1.11	
(3)	1	1	0	0	0	1	1	99	2209	0.44	
(4)	1	1	0	1	0	0	1	105	2809	0.22	
(5)	0	1	0	0	0	1	1	35	289	1.33	
(6)	1	0	0	0	0	1	0	66	196	1.66	
(7)	**0**	**1**	0	**0**	**1**	1	**0**	38	196	1.66	
(8)	**0**	**1**	0	1	**1**	1	**0**	46	36	2.00	best
(9)	1	0	0	1	**1**	**0**	**0**	76	576	0.88	
(10)	0	0	0	1	1	0	1	13	1521	0.66	

The best solution, which is 52, is represented by the following chromosome:

0 1 1 0 1 0 0

An inspection of the current solutions shows that the worse solution is in fact very close to the best solution. The difficulty will be to make both most significant and least significant bit change.

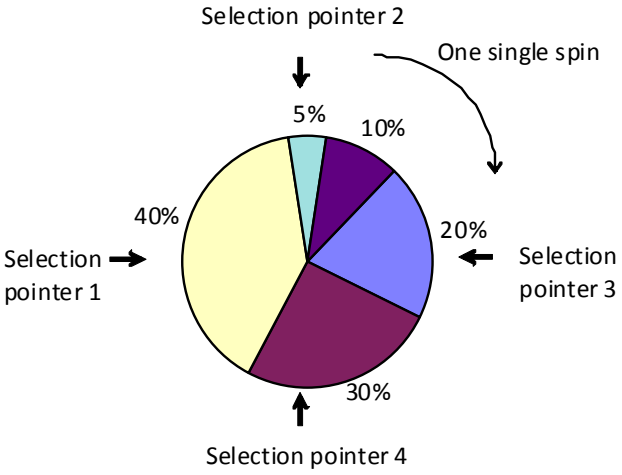
Manually it would be possible to operate a crossover between solutions (7) and (9) at location 5 giving: **0** **1** 0 **0** **1** **0** **0** and then perform a mutation at location 3 to reach the final solution **0** **1** 1 **0** **1** **0** **0**. Note that the use of the best solution would lead to a higher number of recombination to reach the best solution.

Step 5.2 Decide how many individuals to select based on the fitness

There are many options: keep the number constant, increase or decrease it. In the case of Pareto approaches where the aim is to find the non-dominated set of individuals it may be necessary to increase the population size until a maximum manageable limit.

In a 'discrete generation model' the entire population can be replaced at a time. In a 'continuous generation model', a few members can be replaced at a time, only one-at-the-time or many at each generation. In our case given that we are using a small population, e.g. 10, we can replace say 60%. This is relatively high and will promote a good exploration at the start of the search. The generation gap is selected to be: GGAP =0.6. Stochastic universal sampling was chosen due to its ability to select both good and not so good solutions for recombination.

SelCh = select('sus', Chrom, FitnV, GGAP);



The population is shuffled randomly and a single random number in the range [0 Sum/N] is generated, pointer. The N individuals are then chosen by generating the N pointers spaced by 1,

[pointer, pointer +1, ..., pointer +N-1],

and selecting the individuals whose fitnesses span the positions of the pointers.

Step 6: Recombine individuals selected for 'reproduction'

To generate new solutions known as offspring both crossover and mutation are applied on the set of solutions selected for recombination. It is also possible to apply mutation of the chromosomes that have not been recombined by crossover to prevent disturbing potentially good recombinations.

Use various crossover and mutation operator. There is no need to use only one type of operators. However the higher the number of operation on a chromosome the more likely it is to change significantly from the parents. It may be best to apply different operators on different offspring.

Typical crossover rate: 0.5 -1

Typical mutation rate: 0.001 - 0.05

There is nothing preventing you to change the recombination rates as the search progresses, but you should proceed methodically otherwise it may become difficult to compare the performances of your various implementations.

Single point crossover

SelCh = recombin('xovsp',SelCh,0.5); %'xovsp' single point crossover

0	1	0	0	0	1	1	35	289	1.33
1	*0*	*0*	*0*	*0*	*1*	*0*	66	196	1.66
0	*1*	*0*	*0*	*1*	*1*	*0*	38	196	1.66
0	*1*	*0*	*1*	*1*	*1*	*0*	*46*	*36*	*2.00* best

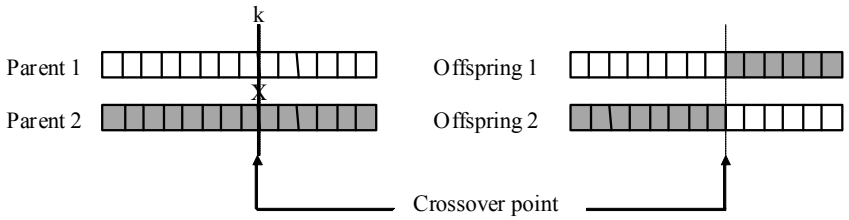


Figure 3.6.1 Illustrating the use of singe point cross over

The individuals selected for recombination are represented by the variable SelCh. Applying the above mutation operator give rise to the set of chromosomes denoted xov.

SelCh =	xov
0 1 0 1 1 1 0 (best solution)	0 1 0 1 1 1 0
0 0 1 1 1 1 0	0 0 1 1 1 1 0
1 0 0 0 0 1 0	**1 0 0 1 1 0 0**
1 0 0 **1 1 0 0**	1 0 0 0 0 1 0
0 1 0 0 *0 1 1*	**0 1 0 0 1 1 0**
0 1 0 0 **1 1 0**	*0 1 0 0 0 1 1*

It can be observed that these two crossovers did not help much in converging to the correct bit string: 0 1 1 0 1 0 0

Mutation

Having applied the crossover operation, mutation is now applied. The mutation operator implemented in the toolbox works as follows: generate a random matrix of the size of the population and find all the elements less than Pm: rand(Nind,Lind)<Pm then change their values from 1 to 0 and vice versa.

NewChrom = rem(OldChrom+(rand(Nind,Lind)<Pm).*ceil(rand(Nind,Lind).*(BaseM-1)),BaseM);

Where Pm is the mutation probability selected as 0.4/Lind.

SelCh = mut(SelCh,0.4/Lind)

0 1 0 1 1 1 0

0 0 1 1 1 **0** **1**

1 **1** 0 1 1 0 0

1 0 0 0 0 1 0

0 1 0 0 1 1 0

0 1 0 0 0 1 **0**

The highlighted bits indicate the bits that are in the optimal solution.

Step 7 Calculate the value of the objective for the new solutions

To calculate the objective value it is first required to decode the chromosomes into integer values (in our case) prior to evaluating them with the user specified objective function.

MyNewVar = bs2rv(SelCh,FieldD);

ObjVSel =ga_objectivefunction_m23mse_1var(MyNewVar);

From the binary string							deduce x and calculate	f(x)
0	1	0	1	1	1	0	46	36
0	0	1	1	1	0	1	29	529
1	1	0	1	1	0	0	108	3136
1	0	0	0	0	1	0	66	196
0	1	0	0	1	1	0	38	196

0	1	0	0	0	1	0	34	324
0	1	1	0	1	0	0	52	

It can be observed that the bit locations indicated in the best individuals are now more present in the new solutions produced than in the original solution, however the bit position 3 from the most significant position is still not very common being exhibited in only one chromosome.

Having calculated the cost from the objective function it is then required to deduce the fitness of each individual before reinserting them in the new population.

Step 8 Reinsert the new solutions in the population based on the fitness value of all the solutions

The best should always be selected for recombination. There is however a trade-off. Selection of the best individuals in the population may lead to 'inbreeding' and rapid but premature convergence to a local solution. If the selective pressure is high, and use mainly a subset of the best solutions, premature convergence may occur. If it is low there will be slower convergence but the ability to explore a wider search space is increased.

A reinsertion procedure that replaces the least fit parents has been adopted in this case.

```
[Chrom ObjV]=reins(Chrom,SelCh,1,1,ObjV,ObjVSel); %1 - fitness-based insertion
```

To evaluate the effect of the reinsertion consider the initial and new population. It can readily be observed that the overall quality of the solutions has been greatly increased. Note that in this toolbox implementation it is possible to obtain a population filled with the same individual. It is however possible to prevent duplicates in the population to improve the ability of the GAs to explore new parts of the solution space. Having many copies of the best individual may lead to quicker convergence but could also lead to a premature convergence towards a non optimal solution.

The shaded bits represent the bits that are identical to the optimal solution at in the new population. The shaded costs indicate the costs for the new individuals that have replaced the old ones. Note that in this algorithm implementation there is no check to prevent good individuals to be duplicated in the population. This is however to be avoided as it limits population diversity.

It can be observed that in this new population it is still not possible to obtain the best solution by using one recombination of a single or even a double point crossover. This reduces the likelihood of finding the best solution at the next iteration.

The reader should start with this population, evaluate the objective function and then perform another set of selection and recombination. It is left to the reader to investigate alternative crossover and mutation operators.

Note that the best solution was close to the worse one because binary coding was used. Using gray coding would have prevented this from happening.

Initial population

1	1	1	0	1	0	1	117	4225
0	0	1	1	1	1	0	30	484
1	1	0	0	0	1	1	99	2209
1	1	0	1	0	0	1	105	2809
0	1	0	0	0	1	1	35	289
1	0	0	0	0	1	0	66	196
0	1	0	0	1	1	0	38	196
0	1	0	1	1	1	0	46	36
1	0	0	1	1	0	0	76	576
0	0	0	1	1	0	1	13	1521

Total cost 12541

0	1	1	0	1	0	0	52	Best solution

New population

0	1	0	1	1	1	0	46	36
0	1	0	0	0	1	0	34	324
1	1	0	1	1	0	0	108	3136
0	0	1	1	1	0	1	29	529
0	1	0	0	0	1	1	35	289
1	0	0	0	0	1	0	66	196
0	1	0	0	1	1	0	38	196
0	1	0	1	1	1	0	46	36
0	1	0	0	1	1	0	38	196
1	0	0	0	0	1	0	66	196

Total cost 5134 244% improvement on average

3.7 Exercises

3.7.1 Revision of GAs terminology and functionality

i) Describe the relative merits of binary and Gray coding to represent a decision variable.

Solution:

Issue with binary: phenotypes that are close together may be far apart in terms of genotype, e.g. to change a value from 3 and 4 (in terms of phenotype which is only one unit different, requires the first three bits 011 to change to 100 (assuming a 3 bit representation).

Gray coding changes at most one bit between each incremented value in the phenotype (the real valued decision variable).

Gray coding has been shown to lead on average to more optimal solutions than binary

ii) Assuming that the range of each decision variable is [0, 50] and the minimum increment is 2, e.g. the variable can take the values [0, 2, 4,…, 50]. Calculate the number of bits required to code one variable

Solution:

n = ceil (log2((Max - Min) / Precision + 1))

4.7, hence 5 bits are required

iii) Calculate the genotype, using binary coding, of a decision variable equal to 26

Solution:

26 is in position 14,

hence 01110

iv) Explain the purpose and relative merits of crossover and mutation

Solution:

Crossover aims to recombine existing solutions to find better ones. It does not create any new alleles in binary or Gray coded chromosomes. Crossover exploits current information. It is mostly useful at the start of the search

Mutation changes the value of an allele and has an exploration function, allowing the creation of phenotypes that can be significantly different from the parents

v) Two chromosomes represented by the following strings are to be used for recombination:

A = 1 0 1 0 1 0 0 0 1 1

B = 1 0 0 0 1 0 1 0 1 1

Name and explain the procedure used to ensure that the selection of one crossover point will lead to an offspring which is different from the parents A and B. Illustrate your answer by applying a single point crossover to the chromosomes A and B.

Solution:

Reduced surrogate, perform an xor on the parents A and B and select a crossover point between the two extreme elements containing 1.

0 0 1 0 0 0 1 0 0 0

Positions 5-7 from Least significant bit, example

A = **1 0 1 0 1 0 0 0 1 1**

B = 1 0 0 0 1 0 1 0 1 1

C1 = **1 0 1 0** 1 0 1 0 1 1

C2= 1 0 0 0 **1 0 0 0 1 1**

vi) A genetic algorithm is used to solve a maximisation problem, where the highest values of the cost function represent better solutions. The values of the objective function for four candidate solutions are f1= 10, f2=100, f3= 5 and f4= 7.

Solution:

$$F(x_i) = \frac{f(\mathbf{x}_i)}{\sum_{i=1}^{N_{POP}} f(\mathbf{x}_i)}$$

FT1 = 10 / (10 +100+5+7) = 0.0820

FT2=100 / (10 +100+5+7) = 0.8197

FT3 = 5 /(10 +100+5+7) = 0.0410

FT4 = 5 /(10 +100+5+7) = 0.0574

Using proportional fitness the solutions 1, 3amd 4 have very little chance to be selected, hence the GAs could keep selecting the same chromosome for recombination, limiting the possibility of creating new solutions.

Using a ranking method would enable the chances of selecting solutions 1, 3 and 4 to be increased.

3.7.2 Solving an equation

i) Describe using pseudo code the genetic algorithms that you have adopted to find the minimum of the following function $y=(x^3-5.5^3)^2$

To implement the above function you will need to use dot operations to be able to evaluate the whole population implemented as a vector x: y= (x.^3 - 5.5^3).^2 within the cost function ga_objectivefunction_m23mse_1var.m

Your description should include a description and a justification (when applicable) of

1) the GAs flowchart

2) the coding system adopted

3) the recombination operators used and their parameters

4) an illustration of the use of such operator(s) for your solution

5) the selection method adopted

6) the insertion method adopted

ii) Find the best combination and values for NIND,MAXGEN, GGAP as well as probability of applying crossover and mutation. You can use 0.7 for crossover and 0.1 for mutation as an initial guess.

```
SelCh = recombin('xovsp',SelCh,0.7);

SelCh = mut(SelCh,0.1/Lind);
```

The main file is called: sga_m23mse.m

The objective function is: ga_objectivefunction_m23mse_1var.m

To determine the best set of parameters for this problem you will need to run the GAs a statistically significant number of times. Run it for 10 times, then 50 and finally 100.

iii) Comment on the average performance of the GAs and identify how many times it gives the best solutions and how quickly it does it.

iv) Explain the approach that you have used to tune your genetic algorithm and justify the choice of the GAs parameters that you have adopted such as the recombination rates, the size of the population, the number of new individual created at each generation. Give the parameter of the best solution and the average solution obtained.

3.7.3 System identification using genetic algorithms

Create the following Simulink model illustrated in **Błąd! Nie można odnaleźć źródła odwołania.** and find the values of the gain K and the system time constant T. Evaluate the effect of reducing the time the simulation runs and reducing the amount of noise in the reference. You should observe that estimating appropriate parameters becomes more difficult as you do not have sufficient excitation of the system.

The first two questions you need to ask yourself are:

What are the objectives of the problem?

How are we going to express them mathematically?

Attributes, criteria, objectives and goals are words used to define the user requirements when solving an optimisation problem.

- Attributes are usually concerned with properties or characteristics of alternative solutions.
- An objective is a specific desire to attain with some level a goal (e.g. be within the top 10%). Sometimes they refer to unattainable ideals.
- A criterion (plural criteria) is an evaluative measure, a dimension or a scale against which different alternative solutions will be evaluated.
- A goal can be any of the above but is considered in the operational research literature as something attainable by contrast to ideal objectives.

In this exercise the suggested cost is a function of the error between the plant output and the model output. The error criterion implemented is the sum of the squares of an inflated error (100 times the error). You should experiment with other cost functions such as the sum of the absolute value of the error.

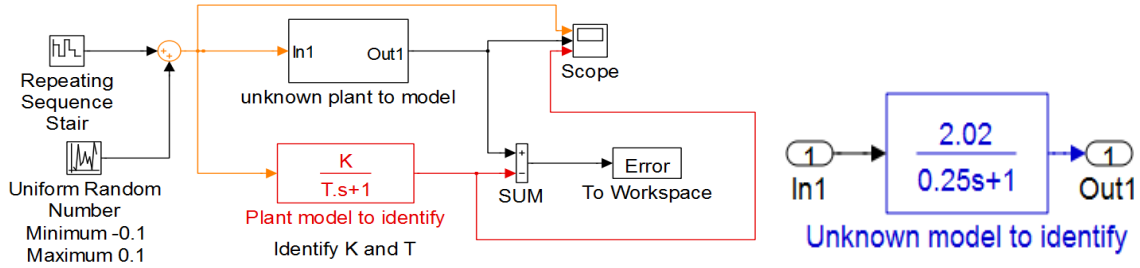


Figure 3.7.1 Simulink model to implement and to evaluate the fit/error between a model and an unknown plant first order transfer function

If you run the genetic algorithm for a number of times you should arrived a good solution such as K=2.0196 T= 0.2505. One of the issues with the genetic algorithm toolbox used is the difficulty in specifying the exact resolution of the requested solution when working with a real value phenotypes (variable to optimise) but binary coded gene. One method is to implement the rounding in the objective function: e.g. to round to x two decimal places use the following code 0.01*round(x*100).

Note that such an approach may however slow down the algorithm as small changes in cost may not be recognisable (i.e. several solutions that are different may have the same cost).

A GAs is not deterministic; hence, every time you run a simulation you may arrive at different results. When you do reach the same result it may well be by a different path, see Figure 3.7.2.

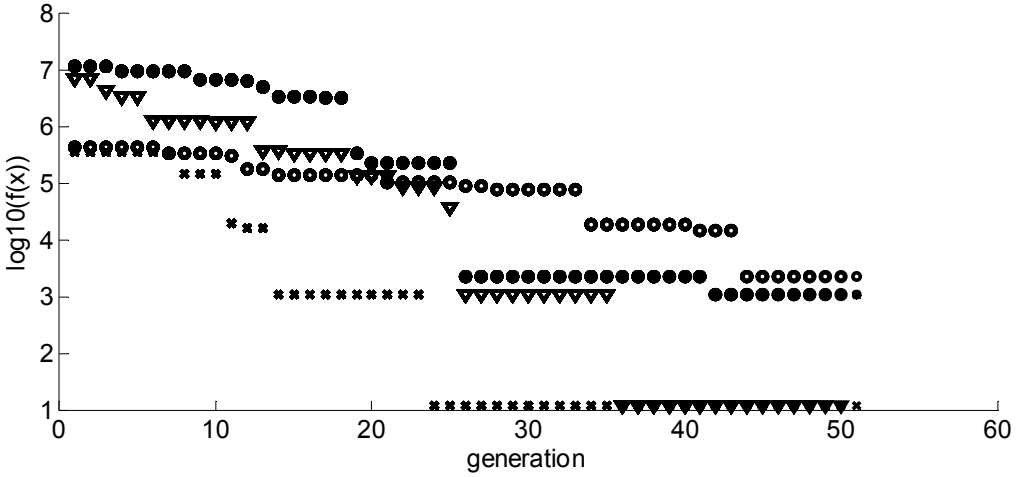


Figure 3.7.2 Illustrating the convergence of the genetic algorithm for four different runs. On two occasions a similar result is found.

Using two decimal places and considering the first order model from Figure, many solutions will be visually similar in terms of their response.

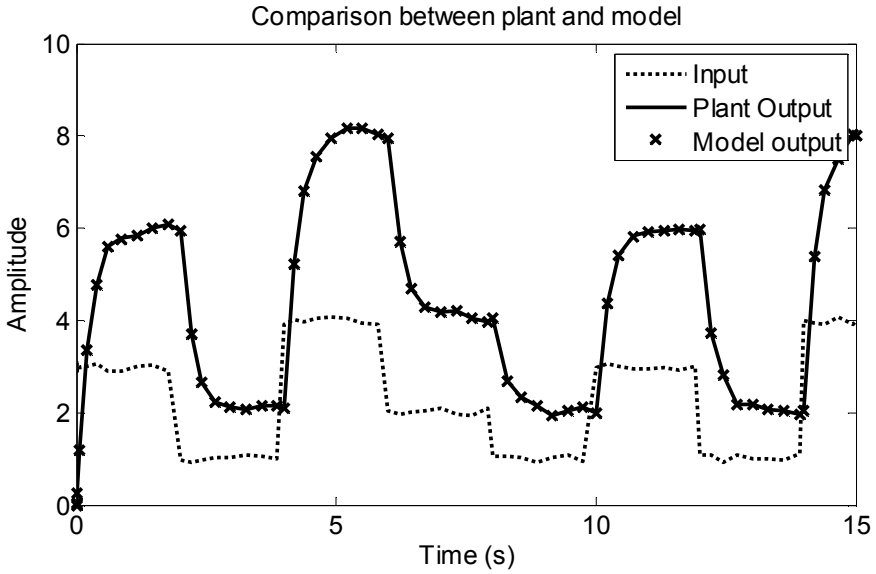


Figure 3.7.3 Illustrating the good model fit between the plant and the estimated model subject to an input represented by the dotted line.

3.7.4 Tuning PID controller with a GA

Create the Simulink model in Figure 3.7.4 and find the values of Kp, Ki and Kd the PID controller gains.

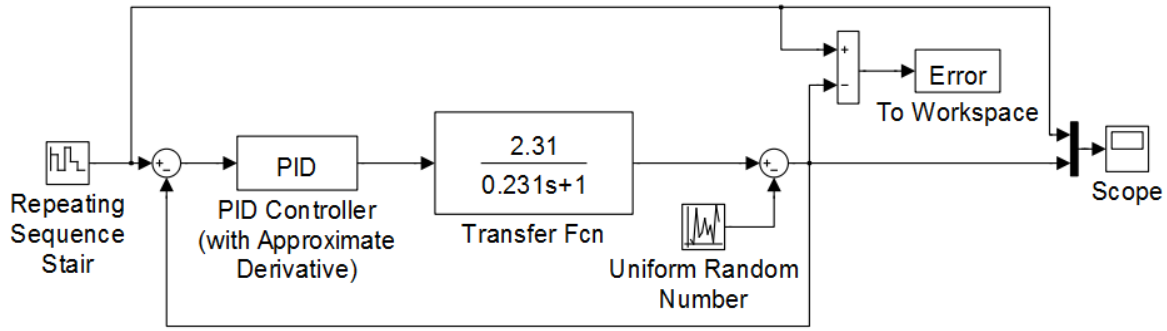


Figure 3.7.4 closed loop control system using a PID. The Error signal is used by the GAs to tune the PID controller

Use the genetic algorithm main file in Section 3.8together with the file implementing the objective or cost function to optimise the PID gains. Ideally the PID gains should not be too high, however the cost function implemented does not penalise high PID gains and focuses on the sum of the squares of the errors. It is strongly suggested to amend the objective function to be able to tune the PID using criteria such as the rise time, settling time and percentage overshoot. The aforementioned objectives, however, assume that you apply a single step to the system.

An example of closed loop control performance resulting from the tuning using GAs is given in Section 3.8

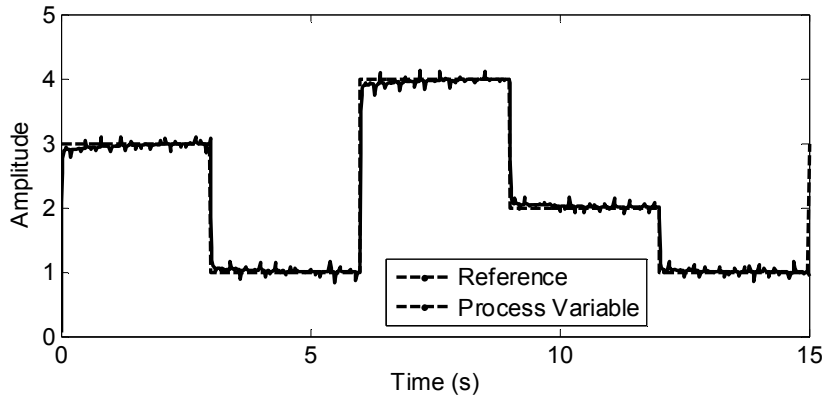


Figure 3.7.5 Input output response of the closed loop plant with a tuned PID with gains Kp = 8.6315 Ki = 8.4360 and Kd = 0.

3.8 MATLAB® code for GAs and cost function implementation

This appendix contains a sample of the code to be used with the Genetic Algorithm Toolbox for MATLAB® developed at the Department of Automatic Control and Systems Engineering of The University of Sheffield, UK. It was originally developed for MATLAB® v4.2, but has also been successfully used with subsequent versions up to and including MATLAB® v7. It was tested by the author for version 10a.

http://www.acse.dept.shef.ac.uk/cgi-bin/gatbx-download

ga_obj_fct_simulink_mode.m

sga_m23mse_sim.m

```
% function ga_obj_fct_simulink_mode
% Author:  Olivier Haas
% History: November 2008  file created
% Oct 2010: added both PID and model cost
% function.
%
% TO DO: implement different criteria
%
% Calculate the error for NoTrial different
solutions
% Need to set the PIDflag to:
%  0 for 1st order model identification
%  1 for PID gains estimation
%
% Case model identification
%  x = [GainK TimeConstantT]
%  x(1) = K; x(2) = T
%
% Case PID gains tuning
%  x = [Kp Ki Kd]
%  x(1) = Kp; x(2) = Ki; x(3) = Kd

function E = ga_obj_fct_simulink_model(x,
PIDflag )

global K T;
global Kp Ki Kd;

[NoTrial,Nvar]= size(x);
E=zeros(NoTrial,1);

for i=1:NoTrial
 K=x(i,1);
 T = x(i,2) ;
 opt = simset('SrcWorkspace','Current');
 if PIDflag==0
  K=x(i,1);
  T = x(i,2) ;
  sim('cw_model_m23mse',15,opt);
 else
  Kp=x(i,1);Ki=x(i,2);Kd=x(i,3);
  sim('m23cc_tf_tunePID',15,opt);
 end
 % if use scope returns time and Error data
 % if use To Workspace block returns only
 %Error data
 E(i) = sum((100*Error).^2);
end
```

```
% sga_m23mse_sim.m
%
% This script implements the Simple Genetic Algorithm described
% in the examples section of the GAs Toolbox manual.
%
% Author:  Andrew Chipperfield
% History: 23-Mar-94  file created
%
% tested under MATLAB® v6 by Alex Shenfield (22-Jan-03)
% tested under MATLAB® 2010a by O Haas
% Modified & renamed by O Haas Nov 2008 and Oct 2010
%%
warning('off','MATLAB:dispatcher:InexactMatch')
global K T; % only required to evaluate 1st order model
global Kp Ki Kd;% only required to evaluate PID gains

%% find the best parameters NIND,MAXGEN, GGAP
NIND = 20;   % Number of individuals per subpopulations
MAXGEN = 50;  % maximum Number of generations
GGAP = .4;   % Generation gap, how many new individuals are created

PIDflag=0;
if PIDflag==1
 NoOfVar = 3;   % Number of variables for PID
else
 NoOfVar = 2;   % Number of variables for plant model
end
% range of variable(s) to be optimised
if PIDflag==1
 MinMaxVar = [0;10];
else
 MinMaxVar = [0;4];
end
RESOLUTION = 0.01; % resolution
NoOfBits =
max(size(dec2bin(diff(MinMaxVar)/RESOLUTION)));

% Build field descriptor to specify size of
% chromosomes and resolution of variables

% len - row vector containing the length of
%   each substring in Chrom. sum(len)
%   should equal the individual length.
len = rep([NoOfBits],[1, NoOfVar]);

% lb, ub Lower and upper bounds for each
%   variable.
lb = rep([MinMaxVar(1)],[1 NoOfVar]);
lu = rep([MinMaxVar(2)],[1 NoOfVar]);

% code - binary row vector indicating how each
%   substring is to be decoded.
%   (0=binary  | 1=gray)
code = 1*ones(1,NoOfVar);

% scale - binary row vector indicating where to
%   use arithmetic and/or logarithmic
%   scaling.
%   (0=arithmetic | 1=logarithmic)
scale = 0*ones(1,NoOfVar);

% lbin,
% ubin - binary row vectors indicating whether
%   or not to include each bound in the
%   representation range
%   (0=excluded | 1=included)
lbin = ones(1,NoOfVar);
ubin = ones(1,NoOfVar);

FieldDR = [lb;lu];
Lind = sum(len);

FieldD = [rep([NoOfBits],[1, NoOfVar]); rep([-10;10],[1, NoOfVar]);...
 rep([1; 0; 1 ;1], [1, NoOfVar])];

FieldD=[len;lb;lu;code;scale;lbin;ubin];

% Initialise population
Chrom = crtbp(NIND, NoOfVar*NoOfBits);
```

```
% Reset counters
Best = NaN*ones(MAXGEN,1); % best in
current population
gen = 0;   % generational counter

% Evaluate initial population
% ObjV = objfun1(bs2rv(Chrom,FieldD));
MyVAR = bs2rv(Chrom,FieldD);
% ObjV
=ga_objectivefunction_m23mse_1var(MyVAR)
;
ObjV =ga_obj_fct_simulink_model(MyVAR,
PIDflag);

% Track best individual and display
convergence
Best(gen+1) = min(ObjV);
figure(1)
%clf
hold on

plot(log10(Best),'rv');xlabel('generation');
ylabel('log10(f(x))');
%text(0.3,0.5,['Best = ',
num2str(Best(gen+1))],'Units','normalized');
drawnow;
ALLBest=Best(gen+1);
% Generational loop
while gen < MAXGEN,

 % Assign fitness-value to entire population
 FitnV = ranking(ObjV);

 % Select individuals for breeding
 SelCh = select('sus', Chrom, FitnV, GGAP);

 %% find the best values for the Xov 0.7 and
mutation 0.7/Lind
 % Recombine selected individuals (crossover)
 SelCh = recombin('xovsp',SelCh,0.8); % first
no to change

 % Perform mutation on offspring
 SelCh = mut(SelCh,0.7/Lind); % second no to
change


 % Evaluate offspring, call objective function
 MyNewVar = bs2rv(SelCh,FieldD);
 %ObjVSel
=ga_objectivefunction_m23mse_1var(MyNew
Var);
 if PIDflag==1
  ObjVSel
=ga_obj_fct_simulink_model(MyNewVar,
PIDflag);
 else
  ObjVSel
=ga_obj_fct_simulink_model(MyNewVar,
PIDflag);
 end
 % Reinsert offspring into current population
 [Chrom
ObjV]=reins(Chrom,SelCh,1,1,ObjV,ObjVSel);

 % Increment generational counter
 gen = gen+1;

 % Update display and record current best
individual
 [Best(gen+1),Index] = min(ObjV);
 bs2rv(Chrom(Index,:),FieldD)
 plot(log10(Best),'rv'); xlabel('generation');
ylabel('log10(f(x))');
 %text(0.3,0.5,['Best = ',
num2str(Best(gen+1))],'Units','normalized');
 ALLBest = [ALLBest,Best(gen+1)];
 drawnow;
end


%% calculate best solution from fminsearch
X=bs2rv(Chrom(Index,:),FieldD)
if PIDflag==1
 Kp=X(1); Ki=X(2);Kd=X(3);
 opt = simset('SrcWorkspace','Current');
 sim('m23cc_tf_tunePID',15,opt);
 % extract data from structure
 t_ga=ScopeData.time;
 ref_ga=ScopeData.signals.values(:,1);
 pv_ga=ScopeData.signals.values(:,2);
 % display response of closed loop system
 figure(1)
 hold on
 plot(t_ga,pv_ga,'k')
 legend('pv ga')

else
```

```
K=X(1);  T = X(2) ;
opt = simset('SrcWorkspace','Current');
sim('model2optimise',15,opt);
% extract data from structure
t=ScopeData.time;
Input=ScopeData.signals(1).values;
Plant=ScopeData.signals(2).values;
Model=ScopeData.signals(3).values;
% compare model and plant output
figure(2),
clf
plot(t,Input,'g')
 hold on
 plot(t,Plant,'k+')
 plot(t,Model,'kv')
 xlabel('Time (s)')
 ylabel('Amplitude')
 title('Comparison between plant and model')
 legend('Input', 'Plant Output', 'Model output')
end
% End of GA
```

3.9 Bibliography on Genetic algorithm and Evolution strategies

Goldberg D. E. and Richardson J. 1987. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, John J. Grefenstette (Ed.). L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 41-4

Holland J.H., 1989, Searching nonlinear functions for high values, Applied Mathematics and Computation, Volume 32, Issues 2-3, August 1989, Pages 255-274

Michalewicz Z., Genetic Algorithms + Data structures = Evolution programming, 2nd Ed, Springer Verlag, 1994

Daellenbach H.D., George J.A., *Introduction to Operations Research Techniques*, Allyn and Bacon, 1978,

Reeves C. R., *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, 1993,

Goldberg D.E., *Genetic algorithms in Search Optimization & Machine Learning*, Addison-Wesley, 1989

Chipperfield, Fleming, Polheim, Fonseca, 1995, Genetic Algorithm toolbox for MATLAB®.

Stephens C. R. and Poli R., 2004, EC Theory – "In theory" Towards a Unification of Evolutionary computation Theory in Frontiers of Evolutionary Computation**,** Kluwer Academic Publishers**,** 9781402077821