NEAR EAST UNIVERSITY

Faculty Of Engineering

Department Of Computer Engineering

SOFT COMPUTING THEORY AND ITS APPLICATIONS

GRADUATION PROJECT
COM-400

STUDENT: Hafiz Zulfiqar Ali

SUPERVISOR: Asst.Prof.Dr Rahib Abiyev

Nicosia-2001

ACKNOWLEDGEMENTS

I am happy that Allah Taa'la the Almighty Supreme Being and Hazrat Muhammad (peace be upon him) has provided me with the strength and courage to complete the task.

I'm extremely grateful to my parents who nurtured me well and inculcated in me the sprit and enthusiasm to learn more and more.

Furthermore, I wish to thank my honorable teacher Asst. Prof. Dr Rahib Abiyev, in Computer Engineering Department at Near East University, who has given me help and encouragement in my project. He has also provided me with the best environment to learn about soft computing. I especially want to thank him for his continuing support. Also I am thankful to Mr. Babar, Awais, Naveed, Faisal, and Shahid who helped us a lot in this project.

TABLE OF CONTENTS

ABSTRACT

Soft computing is a collection of the intelligent paradigms such as Fuzzy logic, Neural Networks and Genetic Algorithms which deal with pervasive imprecision and all defending of the real world. Lutfi Zadeh noted that unlike traditional hard computing, soft computing aims at an accommodation with the pervasive imprecision of the real world.

The project is devoted one of the actual problem of the soft computing elements to industrial processes. For this reason the structure of Soft Computing, description of it's main elements are given.

The structure of Fuzzy System, its main blocks and their functioning principles are given. The different architectures of Neural Networks, their operating principles and learning algorithms are described. Also Genetic Algorithm description, its main functioning principle and genetic operators are given.

In the last chapter using Fuzzy Logic, Neural Networks and Genetic Algorithms, the construction of the hybrid systems are considered. The development of the Neuro Fuzzy, Neuro Genetic systems are given, the application of these systems to technological processes and obtained results are discussed.

INTRODUCTION

Soft computing differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty and partial truth. In effect, the role model for soft computing is the human mind. The guiding principle of soft computing is: Exploit the tolerance for imprecision, uncertainty and partial truth to achieve tractability, robustness and low solution cost.

At this juncture, the principal constituents of soft computing (SC) are fuzzy logic (FL), neural network theory (NN) and probabilistic reasoning (PR), with the latter subsuming belief networks, genetic algorithms, chaos theory and parts of learning theory. What is important to note is that SC is not a combination of FL, NN and PR. Rather, it is a partnership in which each of the partners contributes a distinct methodology for addressing problems in its domain. In this perspective, the principal contributions of FL, NN and PR are complementary rather than competitive.

GA is reminiscent of sexual reproduction in which the genes of two parents combine to form those of their children. When it is applied to problem solving, the basic premise is that we can create an initial population of individuals representing possible solutions to a problem we are trying to solve. Each of these individual has certain characteristics that make them more or less fit as members of the population. The most fit members will have a higher probability of mating than the less fit members, to produce offspring that have a significant chance of retaining the desirable characteristics of their parents. This method is very effective at finding optimal or near optimal solutions to a wide variety of problems, because it does not impose many of the limitations required by traditional methods. It is an elegant generate and test strategy that can identify and exploit regularities in the environment, and converges on solutions that were globally optimal or nearly so.

Fuzzy logic has been applied very successfully in many areas where conventional model based approaches are difficult or not cost-effective to implement. However, as system complexity increases, reliable fuzzy rules and membership functions used to describe the system behavior are difficult to determine. Furthermore, due to the dynamic nature of economic and financial applications, rules and membership functions must be adaptive to the changing environment in order to continue to be useful.

Neuro-Fuzzy hybrid systems combine the advantages of fuzzy systems, which deal with explicit knowledge which can be explained and understood, and neural networks which deal with implicit knowledge which can be acquired by learning. Neural network learning provides a good way to adjust the expert's knowledge and automatically generate additional fuzzy rules and membership functions, to meet certain specifications and reduce design time and costs. On the other hand, fuzzy logic enhances the generalization capability of a neural network system by providing more reliable output when extrapolation is needed beyond the limits of the training data.

CHAPTER ONE

SOFT COMPUTING AND ITS ROLE IN ARTIFICIAL INTELLIGENCE

1.1 Introduction

Artificial intelligence as a science has existing for about 40 years now. The main problem of this science is recreation of human's reasoning processes and behavior with aid of computers and other hand-made devices as well as construction of machines simulating the decision making by human in case of imprecise and uncertain environment. In most cases these various areas are attributed to the artificial intelligence scope, where precise models, methods and algorithms for solving the problem are not available, the problem being characterized by uncertainty. Methods of artificial intelligence are based on two characteristic features:

1. Use of information in symbolic form i.e. letters, words, phrases, signs, figures.

2. Search with aid of symbolic logic. When processing symbolic information, the computers convert the words and phrases to the form of binary digits. Then the computer recognizes or compares consequences of such symbols (converted to digits).

The classics of artificial intelligence stated that the abilities of Computers to manipulate symbols as easily as numbers, to compare consequences of symbols, and then, depending on the results of comparison, do or don't perform further operations, will allow realization in the machine the functions typical for the human mind, i.e. functions of deductive logical reasoning. It may seem that the potential abilities of a computer on the way of creation of artificial intelligence based on the symbolic information processing are unlimited. Despite huge successes of artificial intelligence (in the classical sense) in developing a wide range of systems for solving problems, automatically proving theorems, recognizing patterns as well as in constructing game systems,

expert systems, natural language understanding systems, the expectations have not been approved. The traditional artificial intelligence is not capable of solving problems like a man does with his common sense, and does not accord with the procedures, which are similar to human abilities of understanding and reasoning. The traditional artificial intelligence has not managed to exhibit itself in solving problems for intelligent robotics, computer vision, recognition of speech and hand-written, machine translation, learning through experience and many other important real-world problems. The pointed problems as well as many others have intrinsic imprecision and uncertainty that cannot be neglected. As noted professor L.Zadeh, the traditional artificial intelligence could achieve more successes with its goals if it did not limit itself by processing symbolic information only and using the first order logic. All traditional artificial intelligence systems have been realized by using the Hard Computing technology, which restricts considerably abilities of those systems. Moreover, the traditional artificial intelligence due to the regularities shown above does not consider the computational methods important for accounting uncertainty and imprecision. In this conditions MIQ for traditional artificial intelligent systems appeared to be not so high. There was a need to increase MIQ for intelligent systems. Thereat Soft Computing methodology appears implying cooperative activity rather than autonome one for such new computational approaches as fuzzy logic, neural networks, evolutionary computation and so on. These approaches allow solving many important real-world problems, which was impossible using traditional artificial intelligence methods.

The collection of such intelligent paradigms (used as computational techniques) as Fuzzy Logic (FL), Neural Networks (NN), Probabilistic Reasoning (PR), Genetic Algorithms (GA), Chaos Theory (CT) dealing with pervasive imprecision and ill definedness of the real world is named Soft Computing (SC). Unlike traditional Hard Computing (HC), SC can tolerate imprecision and uncertainty and partial truth without the loss of performance and effectiveness for the end use. It is the matter of time after no more than a decade

we will see that Artificial Intelligence is based on Soft Computing not on traditional Hard Computing. L.Zadeh noted that unlike the traditional Hard Computing, Soft Computing aims at an accommodation with the pervasive imprecisions of the real world. The guiding principle of Soft Computing is: exploit the tolerance for imprecision, uncertainty and partial truth to achieve tractability, robustness and low solution cost. We can easily come to the conclusion that precision has a cost (unfortunately, this obvious principle often is neglected). So, in order to solve the problem with an acceptable cost we need to aim at a decision with only the necessary degree of precision not going over the requirements. The impressing examples of the aforesaid are problems of landing a helicopter or parking a car. Let's consider the second case. One can park a car without doing any distance and angle measurements because the final position of the car is not specified clearly. If though it is, then the measurements are necessary, say, in the range of fractions of millimeter or a few seconds of arc. This will require many hours of manoeuvres and measurements from the devices for solving the problem. Moreover, the cost of decision will increase exponentially as the precision increases. Soft Computing technology is of great importance for data compression, especially, in HDTV, audio recording, speech recognition, image understanding and related fields. Actually, soft-computing-based concepts and techniques are already playing an essential role in the conception, design and manufacturing of high MIQ products and systems. As noted Zadeh the perfect model of SC is human brain.

1.2 Structure And Constituents Of Soft Computing

As was mentioned above all traditional artificial intelligent systems including expert systems widely used in various areas of human activity, have been realized on the base of Hard Computing, often using computers. But this base, obviously limits the efficiency and, generally, the possibility of creating systems of artificial intelligence for different purposes. Currently the significant increase can be noticed in number of applied artificial intelligence systems based

not on numerical (not symbolic) computation and traditional Hard Computing, but on neural networks, fuzzy computing, evolutionary programming, belief networks. There is as well a certain increase in number of publications, presented in proceedings of scientific conferences which are devoted to fuzzy logic, genetic algorithms, artificial life, biological computing, neural computing etc. This increase gives the evidence that the focus of the investigations and implementations of real artificial intelligence systems makes a shift nearer to Soft Computing. Figure 1.1 shows the structure of Soft Computing technology forming the basis for computational intelligence.

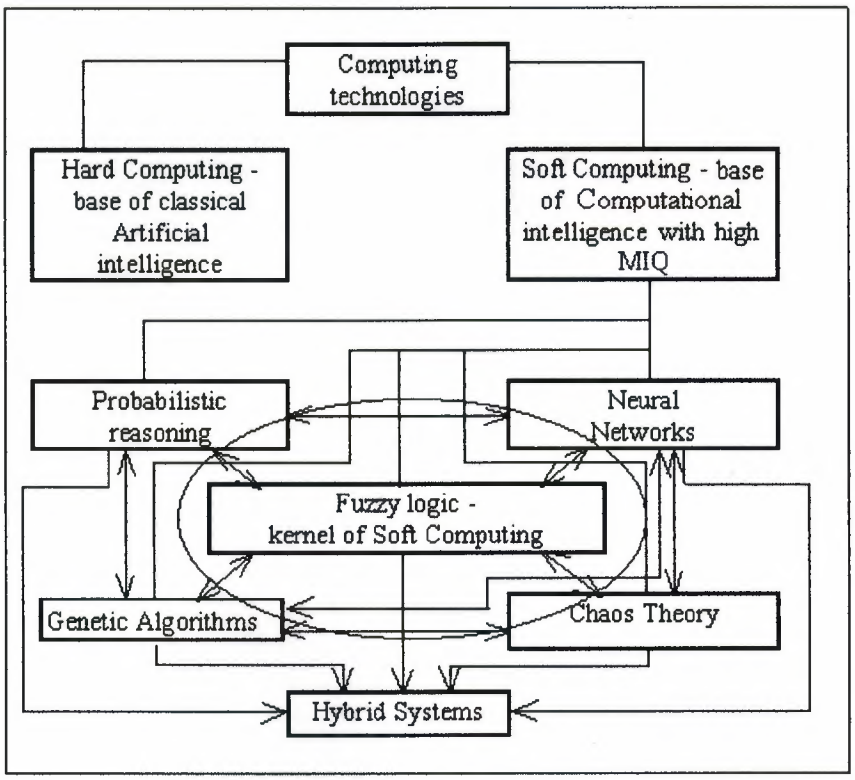


Figure 1.1. The Main Components Of Soft Computing.

The following main components of Soft Computing are known by now: fuzzy logic (FL), neural networks theory (NN), probabilistic reasoning (PR),

Genetic Algorithms (GA), and chaos theory (CT) (Figure 1.1). In SC FL is concerned in the main with imprecision and approximate reasoning, NN - with learning, PR with uncertainty and propagation of belief, GA with global optimization and searching and CT with nonlinear dynamics. In large measure FL, NN and PR are complementary rather that competitive.

CHAPTER TWO

FUZZY SYSTEMS

2.1 Structure of a Fuzzy System

There are specific components characteristic of a fuzzy controller to support a design procedure. In the block diagram in figure2.0, the controller is between a preprocessing block and a post-processing block. The following explains the diagram block by block.

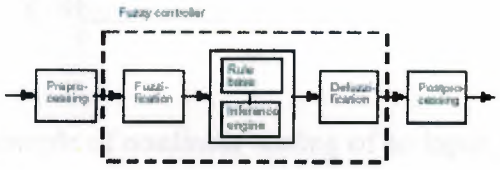


Figure 2.1. Blocks of a fuzzy controller

2.2 Preprocessing

The inputs are most often hard or crisp measurements from some measuring equipment, rather than linguistic. A preprocessor, the first block in Fig. 2.0, conditions the measurements before they enter the controller. Examples of preprocessing are:

- Quantisation in connection with sampling or rounding to integers;
- Normalisation or scaling onto a particular, standard range;
- Filtering in order to remove noise;
- Averaging to obtain long term or short term tendencies;
- A combination of several measurements to obtain key indicators; and
- Differentiation and integration or their discrete equivalences.

A quantiser is necessary to convert the incoming values in order to find the best level in a discrete universe. Assume, for instance, that the variable error has the value 4.5, but the universe is u = (-5, -4…0…4,5). The quantiser rounds to 5 to fit it to the nearest level. Quantisation is a means to reduce data, but if the quantisation is too coarse the controller may oscillate around the reference or even become unstable. Nonlinear scaling is an option (Fig. 2.1). In the FL smidth controller the operator is

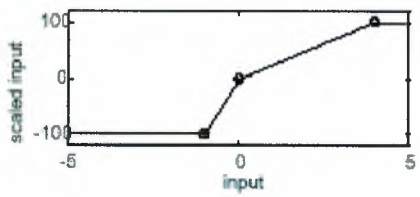


Figure 2.2. Example of nonlinear scaling of an input measurement

asked to enter three typical numbers for a small, medium and large measurement respectively (Holmblad & Stergaard, 1982). They become break points on a curve that scales the incoming measurements (circled in the figure). The overall effect can be interpreted as a distortion of the primary fuzzy sets. It can be confusing with both scaling and gain factors in a controller, and it makes tuning difficult.

When the input to the controller is error, the control strategy is a static mapping between input and control signal. A dynamic controller would have additional inputs, for example derivatives, integrals, or previous values of measurements backwards in time. These are created in the preprocessor thus making the controller multi-dimensional, which requires many rules and makes it more difficult to design.

The preprocessor then passes the data on to the controller.

2.3 Fuzzification

The first block inside the controller is fuzzification, which converts each piece of input data to degrees of membership by a lookup in one or several membership functions. The fuzzification block thus matches the input data with the conditions of the rules to determine how well the condition of each rule matches that particular input instance. There is a degree of membership for each linguistic term that applies to that input variable.

2.4 Rule Base

The rules may use several variables both in the condition and the conclusion of the rules. The controllers can therefore be applied to both multi-input-multi-output (MIMO) problems and single-input-single-output (SISO) problems. The typical SISO problem is to regulate a control signal based on an error signal. The controller may actually need the error, the change in air, and the accumulated error as inputs, but we will call it single-loop control, because in principle all three are formed from the error measurement. To simplify, this section assumes that the control objective is to regulate some process output around a prescribed set point or reference. The presentation is thus limited to single-loop control.

2.4.1 Rule Formates

Basically a linguistic controller contains rules in the if then format, but they can be presented in different formats. In many systems, the rules are presented to the end-user in a format similar to the one below:

1. If error is Neg and change in error is Neg then output is NB

2. If error is Neg and change in error is Zero then output is NM
3. If error is Neg and change in error is Pos then output is Zero
4. If error is Zero and change in error is Neg then output is NM
5. If error is Zero and change in error is Zero then output is Zero (2)
6. If error is Zero and change in error is Pos then output is PM
7. If error is Pos and change in error is Neg then output is Zero
8. If error is Pos and change in error is Zero then output is PM
9. If error is Pos and change in error is Pos then output is PB

The names Zero, Pos, Neg are labels of fuzzy sets as well as NB, NM, PB and PM (negative big, negative medium, positive big, and positive medium respectively). The same set of rules could be presented in a relational format, a more compact representation.

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

Figure 2.3

The top row is the heading, with the names of the variables. It is understood that the two leftmost columns are inputs, the rightmost is the output, and each row represents a rule. This format is perhaps better suited for an experienced user who wants to get an overview of the rule base quickly. The relational format is certainly suited for storing in a relational database. It should be emphasised, though, that the relational format implicitly assumes that the connective between the inputs is always logical and or logical or for that matter as long as it is the same operation for all rules and not a mixture of connectives. Incidentally, a fuzzy rule with an or

combination of terms can be converted into an equivalent and combination of terms using laws of logic (DeMorgan's laws among others). A third format is the tabular linguistic format. Change in error is given by:

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

Figure 2.4

This is even more compact. The input variables are laid out along the axes, and the output variable is inside the table. In case the table has an empty cell, it is an indication of a missing rule, and this format is useful for checking completeness. When the input variables are error and change in air, as they are here, that format is also called a linguistic phase plane. In case there are $n > 2$ input variables involved, the table grows to an þ-dimensional array; rather user-un friendly.

To accommodate several outputs, a nested arrangement is conceivable. A rule with several outputs could also be broken down into several rules with one output. Lastly, a graphical format which shows the fuzzy membership curves is also possible (Fig. 2.4). This graphical user-interface can display the inference process better than the other formats, but takes more space on a monitor.

2.4.2 Connectives

In mathematics, sentences are connected with the words and, or, if-then (or implies), and if and only if, or modifications with the word not. These five are called connectives. It also makes a difference how the connectives are implemented. The most prominent is probably multiplication for fuzzy and instead of minimum. So far

most of the examples have only contained and operations, but a rule like " If error is very neg and not zero or change in error is zero then ..." is also possible.
The connectives and or are always defined in pairs, for example,

a and b = min (a,b) minimum

a or b = max (a,b)= maximum

or

a and b = a * b algebraic product

a or b = a+ b - a * b algebraic or probabilistic sum

2.4.3 Modifiers

A linguistic modifier is an operation that modifies the meaning of a term. For example, in the sentence " very close to 0", the word very modifies close to 0 which is a fuzzy set. A modifier is thus an operation on a fuzzy set. The modifier very can be defined as squaring the subsequent membership function, that is

$$\textbf{very a} = a^2$$

Some examples of other modifiers are

$$\textbf{Extremely a} = a^3$$

$$\textbf{Slightly a} = a$$

Some what a = moreorless a and not slightly a

A whole family of modifiers is generated by a^p where p is any power between zero and infinity. With $p = \infty$ the modifier could be named exactly, because it would suppress all memberships lower than 1.0.

2.4.4 Universe

Elements of a fuzzy set are taken from a universe of discourse or just universe. The universe contains all elements that can come into consideration. Before designing the membership functions it is necessary to consider the universes for the inputs and outputs. Take for example the rule.

If error is Neg and change in error is Pos then output is 0

Naturally, the membership functions for Neg and Pos must be defined for all possible values of error and change in error; and a standard universe may be convenient.

Another consideration is whether the input membership functions should be continuous or discrete. A continuous membership function is defined on a continuous universe by means of parameters. A discrete membership function is defined in terms of a vector with a finite number of elements. In the latter case it is necessary to specify the range of the universe and the value at each point. The choice between fine and coarse resolution is a trade off between accuracy, speed and space demands. The quantiser takes time to execute, and if this time is too precious, continuous membership functions will make the quantiser obsolete.

Example 1 (standard universes) Many authors and several commercial controllers use standard universes.

- The FL smidth controller, for instances, uses the real number interval [-1,1]
- Authors of the earlier papers on fuzzy control used the integers in [-6,6]
- Another possibility is the interval [-100,100] corresponding to the percentages of full scale.

- Yet another is the integer range [0,5095] corresponding to the output from a 12 bit analog to digital converter
- A variant is [-2047,2048], where the interval is shifted in order to accommodate negative numbers

The choice of data types may govern the choices of universe. For example. The voltage range [-5,5] could be represented as an integer range [-50,50], or as a floating point range [-5.0,5.0]; a signed byte data type has an allowable integer range [-128,127]

A way to exploit the range of the universes better is scaling. If a controller input mostly uses just one term, the scaling factor can be turned up such that the whole range is used. An advantage is that this allows a standard universe and it eliminates the need for adding more terms.

2.4.5 Membership Functions

Every element in the universe of discourse is a member of a fuzzy set to some grade, maybe even zero. The grade of membership for all its members describes a fuzzy set, such as Neg. In fuzzy sets elements are assigned a grade of membership, such that the transition from membership to non-membership is gradual rather than abrupt. The set of elements that have a non-zero membership is called the support of the fuzzy set. The function that ties a number to each element χ of the universe is called the membership function $\mu(\chi)$.

The designer is inevitably faced with the question of how to build the term sets. There are two specific questions to consider; (i) How does one determine the shape of the sets? and (ii) How many sets are necessary and sufficient? For example, the error in the position controller uses the family of terms Neg, Zero, and Pos.

According to fuzzy set theory the choice of the shape and width is subjective, but a few rules of thumb apply.

- A term set should be sufficiently wide to allow for noise in the measurement.

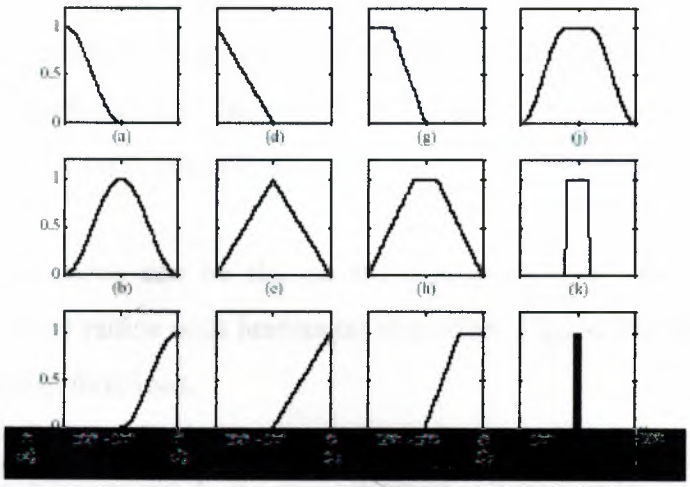


Figure 2.5: Examples of membership functions. Read from top to bottom, left to right: (a) s - function, (b) π – function, (c) z - function, (d-f) triangular versions, (g-i) trapezoidal versions, (j) flat π – function, (k) rectangle, (l) singleton.

- A certain amount of overlap is desirable; otherwise the controller may run into poorly defined states, where it does not return a well defined output.

A preliminary answer to questions (i) and (ii) is that the necessary and sufficient number of sets in a family depends on the width of the sets, and vice versa. A solution could be to ask the process operators to enter their personal preferences for the membership curves; but operators also find it difficult to settle on particular curves.

The manual for the TIL Shell product recommends the following (Hill, Horstkotte &

Teichrow, 1990).

- Start with triangular sets. All membership functions for a particular input or output should be symmetrical triangles of the same width. The leftmost and the rightmost should be shouldered ramps.
- The overlap should be at least 50%.The widths should initially be chosen so that each value of the universe is a member of at least two sets, except possibly for elements at the extreme ends. If, on the other hand, there is a gap between two sets no rules fire for values in the gap. Consequently the controller function is not defined.

Membership functions can be flat on the top, piece-wise linear and triangle shaped, rectangular, or ramps with horizontal shoulders. Fig2.4 shows some typical shapes of membership functions.

Strictly speaking, a fuzzy set A is a collection of ordered pairs

$$A = \{(\chi, \mu(\chi))\} \qquad (2.1)$$

Item χ belongs to the universe and $\mu(\chi)$ is its grade of membership in A. A single pair $\{\chi, \mu(\chi)\}$ is a fuzzy singleton; singleton output means replacing the fuzzy sets in the conclusion by numbers (scalars). For example

1. If error is Pos then output is 10 volts
2. If error is Zero then output is 0 volts
3. If error is Neg then output is - 10 volts

There are at least three advantages to this:

- The computations are simpler;

- it is possible to drive the control signal to its extreme values; and
- it may actually be a more intuitive way to write rules.

The scalar can be a fuzzy set with the singleton placed in a proper position. For example 10 volts, would be equivalent to the fuzzy set (0,0,0,0,1) defined on the universe (-10,-5,0,5,10) volts.

Example 2(membership functions) Fuzzy controllers use a variety of membership functions. A common example of a function that produces a bell curve is based on the exponential function.

$$\mu(\chi) = \exp\left[\frac{-(\chi - \chi_0)^2}{2\sigma^2}\right] \tag{2.2}$$

This is a standard Gaussian curve with a maximum value of 1,x is the independent variable on the universe, χ_0 is the position of the peak relative to the universe, and σ is the standard deviation. Another definition that does not use the exponential is

$$\mu(\chi) = \exp\left[\frac{-(\chi - \chi_0)^2}{\sigma^2}\right]^{-1} \tag{2.3}$$

The FL Smidth controller uses the equation

$$\mu(\chi) = 1 - \exp\left[-\left(\frac{\sigma}{\chi - \chi_0}\right)^a\right] \tag{2.4}$$

The extra parameter a controls the gradient of the sloping sides. It is also possible to use other functions, for example the sigmoid known from neural networks.

A cosine function can be used to generate a variety of membership functions. The s- curve can be implemented as

$$s(\chi,\chi_r,\chi)=\begin{cases}0 & ,\chi<\chi_l\\ \frac{1}{2}+\frac{1}{2}\cos\left(\frac{\chi-\chi_r}{\chi_r-\chi_l}\pi\right) & ,\chi_l\le\chi\le\chi_r\\ 1 & ,\chi>\chi_r\end{cases} \qquad (2.5)$$

Where χ_l is the left breakpoint and χ_r is the right break point. The z- curve is just a reflection

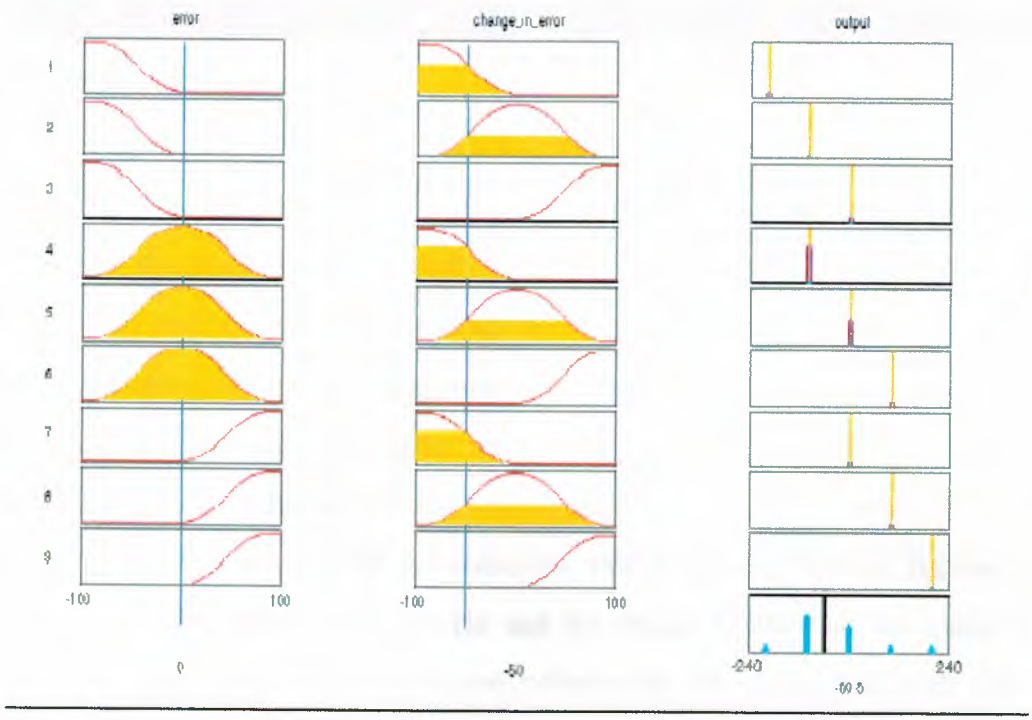


Figure 2.6 Graphical construction of the control signal in a fuzzy PD controller (generated in the Matlab Fuzzy Logic Toolbox).

$$z(\chi,\chi_r,\chi)=\begin{cases}0 & ,\chi<\chi_l\\ \frac{1}{2}+\frac{1}{2}\cos\left(\frac{\chi-\chi_r}{\chi_r-\chi_l}\pi\right) & ,\chi_l\le\chi\le\chi_r\\ 1 & ,\chi>\chi_r\end{cases} \qquad (2.6)$$

Then the π - curve can be implemented as a combination of the s- curve and the z-curve, such that the peak is flat over the interval $[\chi_2,\chi_3]$

$$\pi(\chi_1, \chi_2, \chi_3, \chi_4, \chi) = \min(s(\chi_1, \chi_2, \chi), z(\chi_3, \chi_4, \chi)) \quad (2.7)$$

2.5 Inference Engine

Figures 2.6 and 2.7are both a graphical construction of the algorithm in the core of the controller. In Fig. 2.7, each of the nine rows refers to one rule. For example, the first row says that if the error is negative (row 1, column 1) and the change in error is negative (row 1, column 2) then the output should be negative big (row 1, column 3). The picture corresponds to the rule base in (2). The rules reflect the strategy that the control signal should be a combination of the reference error and the change in error, a fuzzy proportional-derivative controller. We shall refer to that figure in the following. The instances of the error and the change in error are indicated by the vertical lines on the first and second columns of the chart. For each rule, the inference engine looks up the membership values in the condition of the rule.

2.5.1 Aggregation

The aggregation operation is used when calculating the degree of fulfillment or firing strength α_k of the condition of a rule k. A rule, say rule 1, will generate a fuzzy membership value μ_{e1} coming from the error and a membership value μ_{ce1} coming

from the change in error measurement. The aggregation is their combination,

$$\mu_{e1} \text{ and } \mu_{ce1} \quad (2.8)$$

Similarly for the other rules. Aggregation is equivalent to fuzzification, when there is only one input to the controller. Aggregation is sometimes also called fulfillment of the rule or firing strength

2.5.2 Activation

The activation of a rule is the deduction of the conclusion, possibly reduced by its firing strength. Thickened lines in the third column indicate the firing strength of each rule. Only the thickened part of the singletons are activated, and min or product (*) is used as the activation operator. It makes no difference in this case, since the output membership functions are singletons, but in the general case of s- , π - ,and z – functions in the third column, the multiplication scales the membership curves, thus preserving the initial shape, rather than clipping them as the min operation does. Both methods work well in general, although the multiplication results in a slightly smoother control signal. In Fig. 2.4 only rules four and five are active.

A rule k can be weighted a priori by a weighting factor $w_k \in [0,1]$, which is its degree of confidence. In that case the firing strength is modified to

$$\alpha_k = w_k * \alpha_k \qquad (2.9)$$

The designer, or a learning program trying determines the degree of confidence to adapt the rules to some input-output relationship.

2.5.3 Accumulation

All activated conclusions are accumulated, using the max operation, to the final graph on the bottom right (Fig. 2.5). Alternatively, sum accumulation counts overlapping areas more than once (Fig.2.6). Singleton output (Fig. 2.5) and sum accumulation results in the simple output

$$\alpha_1 * s_1 + \alpha_2 * s_2 + \ldots\ldots\ldots + \alpha_n * s_n \qquad (2.10)$$

The alpha's are the firing strengths from the n rules and $s_1 \ldots\ldots\ldots s_n$ are the output singletons. Since this can be computed as a vector product, this type of inference is relatively fast in a matrix oriented language.

There could actually have been several conclusion sets. An example of a one-input-two outputs rule is " If e_a is a then σ_1 is x and σ_2 is y ". The inference engine can treat two (or several) columns on the conclusion side in parallel by applying the firing strength to both conclusion sets. In practice, one would often implement this situation as two rules rather than one, that is, " If e_a is a then σ_1 is x, " If e_a is a then σ_2 is y " .

2.6 Defuzzification

The resulting fuzzy set (Fig. 2.6, bottom right; Fig. 2.7, extreme right) must be converted to a number that can be sent to the process as a control signal. This operation is called defuzzification, and in Fig. 2.6 the x-coordinate marked by a white, vertical dividing line becomes the control signal. The resulting fuzzy set is thus defuzzified into a crisp control signal. There are several defuzzification methods.

2.6.1 Centre of gravity (COG)

The crisp output value þ (white line in Fig. 2.6) is the abscissa under the centre of gravity of the fuzzy set,

$$u = \frac{\sum_i \mu(x_i) x_i}{\sum_i \mu(x_i)} \qquad (2.11)$$

Here x_i is a running point in a discrete universe, and $\mu(x_i)$ is its membership value in the membership function. The expression can be interpreted as the weighted average of the elements in the support set. For the continuous case, replace the summations by integrals. It is a much used method although its computational complexity is relatively high. This method is also called centroid of area.

2.6.2 Center of gravity method fro singletons (COGS)

If the membership functions of the conclusions are singletons (Fig. 7), the output value is

$$u = \frac{\sum_i \mu(s_i) s_i}{\sum_i \mu(s_i)} \qquad (2.12)$$

Here s_i is the position of singleton i in the universe, and $s(x_i)$ is equal to the firing strength α_i of rule i. This method has a relatively good computational complexity, and u is differentiable With respect to the singletons þþ, which is useful in neurofuzzy systems.

$$u = \left\{ \chi \mid \int_{Min}^{x} \mu(\chi) d\chi = \int_{x}^{Max} \mu(x) dx \right\} \qquad (2.13)$$

Here x is the running point in the universe, $\mu(\chi)$ is its membership, Min is the leftmost value of the universe, and Max is the rightmost value. Its computational complexity is relatively high, and it can be ambiguous. For example, if the fuzzy set consists of two singletons any point between the two would divide the area in two halves; consequently it is safer to say that in the discrete case, BOA is not defined.

2.6.3 Mean of maxima (MOM)

An intuitive approach is to choose the point with the strongest possibility, i.e. maximal membership. It may happen, though, that several such points exist, and a common practice is to take the mean of maxima (MOM). This method disregards the shape of the fuzzy set, but the computational complexity is relatively good.

2.6.4 Leftmost maximum (LM), and rightmost maximum (RM)

Another possibility is to choose the leftmost maximum (LM), or the rightmost maximum (RM). In the case of a robot, for instance, it must choose between left or right to avoid an obstacle in front of it.

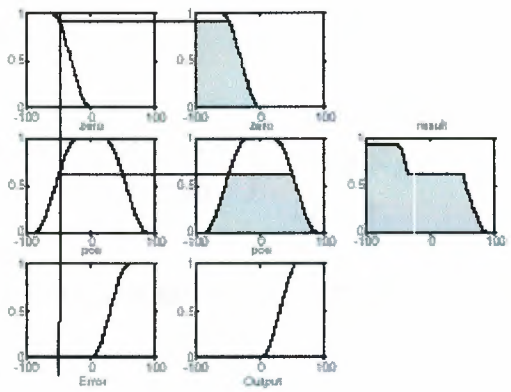


Figure 2.7. One input, one output rule base with non-singleton output sets.

The defuzzifier must then choose one or the other, not something in between. These methods are indifferent to the shape of the fuzzy set, but the computational complexity is relatively small.

2.7 Post processing

Output scaling is also relevant. In case the output is defined on a standard universe this must be scaled to engineering units for instance, volts, meters, or tons per hour. An example is the scaling from the standard universe [-1,1] to the physical units [-10,10] volts.
The postprocessing block often contains an output gain that can be tuned, and sometimes also an integrator.

Example 3(inference) How is the inference in fig 8 implemented using discrete fuzzy sets?

Behind the scene all universes were divided into 201 points from -100 to 100.But for brevity, let us just use five points. Assume the universe **u**, common to all variables, is the vector

U =

-100	-50	0	50	100

A cosine function can be used to generate a variety of membership functions. The s-curve can be implemented as

$$s(x_l, x_r, x) = \begin{cases} 0 & , x < x_l \\ \frac{1}{2} + \frac{1}{2}\cos\left(\frac{x - x_r}{x_r - x_l}\pi\right) & , x_l \le x \le x_r \\ 1 & , x > x_r \end{cases} \quad (2.14)$$

where x_l is the left break point, and x_r is the right breakpoint.The z-curve is just a reflection

$$z(x, x_r, x) = \begin{cases} 1 & , x < x_l \\ \frac{1}{2} + \frac{1}{2}\cos\left(\frac{x - x_r}{x_r - x_l}\pi\right) & , x_l \le x \le x_r \\ 0 & , x > x_r \end{cases} \quad (2.15)$$

Then the π-curve can be implemented as a combination of the s- curve and the z-curve, such that the peak is flat over the interval $[x_2, x_3]$

$$\pi(x_1, x_2, x_3, x_4, x) = \min(s(x_1, x_2, x), z(x_3, x_4, x)) \quad (2.16)$$

A family of terms is defined by means of the π – function, such that

Neg = π (-100, -100, -60,10,**u**) =

1	0.95	0.05	0	0

Zero= π (-90, -20,20,90,**u**)=

0	0.61	1	0.61	0

Pos = π(-10,60,100,100,**u**) =

0	0	0.05	0.95	1

Above we inserted the whole vector u in place of the running point x; the result is thus a vector. The figure assumes that error = -50 (the unit is percentages of full range). This corresponds to the second position in the universe, and the first rule contributes with a membership **neg**(2) = 0.95. This firing strength is propagated to the conclusion side of the rule using **min**, such that the contribution from this rule is

0.95	0.95	0.05	0	0

CHAPTER THREE
NEURAL NETWORKS

3.1 The Artificial Neuron

The basic unit of neural networks, the artificial neurons, simulates the four basic functions of natural neurons. Artificial neurons are much simpler than the biological neuron; the figure below shows the basics of an artificial neuron.

Note that various inputs to the network are represented by the mathematical symbol. x(n). Each of these inputs are multiplied by a connection weight, these weights are represented by w(n). In the simplest case, these products are simply summed, fed through a transfer function to generate a result, and then output.

$$I=\sum w_j x_i$$

$$Y=f(I) \text{ Transfer}$$

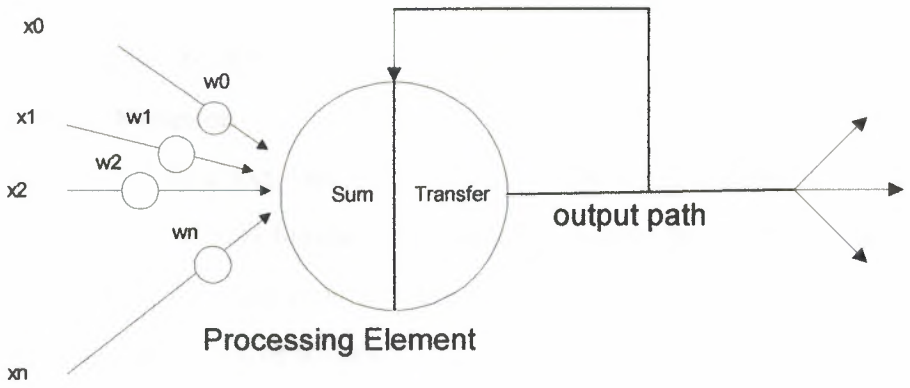


Figure 3.1. Artificial Neuron

Even though all artificial neural networks are constructed from this basic building block the fundamentals may vary in these building blocks and there are differences.

3.2 Threshold functions

Threshold functions also referred to as activation functions, squashing functions, or signal functions, map a PE's (possibly) infinite domain to a prespecified range. Although the number of threshold functions possible is quite varied, there are five that are regularly employed by the majority of neural- networks:

(1) Linear, (2) Step, (3) ramp, (4) Sigmoid, and (5) Gaussian. With the exception of the linear threshold function, all of these introduce a non-linearity in the network dynamics by bounding a PE's output values to a fixed range.

3.2.1 Linear Threshold Function

The linear threshold function (see Figure 2.2(a)), produces a linearly modulated output from the input x as described by the equation

$$f(x)=x$$

Where x ranges over the real numbers and a is a positive scalar, if a=l, it is equivalent to removing the threshold function completely.

3.2.2 Step Threshold Function

The step threshold function, (see Figure 3.2(b)), produces only two values, β and δ. If the input to the threshold function, x, equals or exceeds the threshold value, 0, then the step threshold function produces the value β, otherwise it produces the value -δ, where β and α are positive scalars.

3.2.3 Ramp Threshold Function

The ramp threshold function, (see Figure 3.2(c)), is a combination of the linear and step threshold functions. The ramp threshold function places an upper and lower bound on the values that the threshold function produces and allows a linear

response between the bounds. These saturation points are symmetric around the origin and are discontinuous at the points of saturation.

3.2.4 Sigmoid Threshold Function

The sigmoid threshold function, (see Figure 3.2(d)), is a continuous version of the ramp threshold function. The sigmoid (S-shaped) function is a bounded, monotonic, non-decreasing function that provides a graded, nonlinear response within a prespecified range.

The most common sigmoid function is the logistic function

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

Where ($\alpha > 0$ (usually $\alpha = 1$). which provides an output value from 0 to 1.

3.2.5 Gaussian Threshold Function

The Gaussian threshold function, (see Figure 3.1(e)), is a radial function (symmetric about the origin) that requires a variance value, $v > 0$, to shape the Gaussian function. In some networks the Gaussian function is used in conjunction with a dual set of connections.

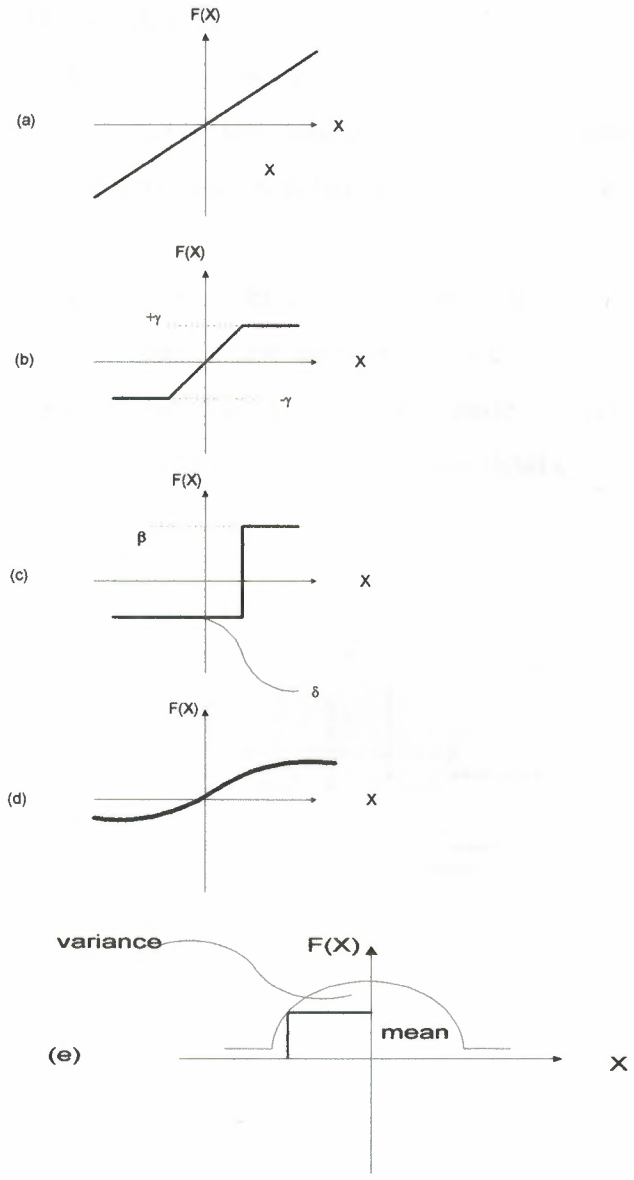


Figure 3.2. Threshold Functions

3.3 Neural Network Topologies

The building blocks for neural networks are in place. Neural networks consist of layer(s) of PEs interconnected by weighted connections. The arrangement of the PEs, connections and patterns into a neural network is referred to as a topology.

3.3.1 Layers

Biologically, neural networks are constructed in a three dimensional way from microscopic components. These neurons seem capable of nearly unrestricted interconnections. This is not true in any man-made network. Artificial neural networks are the simple clustering of the primitive artificial neurons

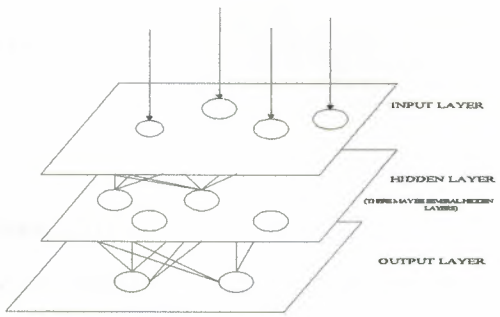


Figure 3.3. Layers structure

As the figure above shows, the neurons are grouped into layers. The input layer consists of neurons that receive input form the external environment.

3.3.2 Communication and types of connections

Neurons are connected via a network of paths carrying the output of one neuron as input to another neuron. These paths is normally unidirectional, there

might however be a two-way connection between two neurons, because there may be an another path in reverse direction. A neuron receives input from many neurons, but produce a single output, which is communicated to other neurons.

The neuron in a layer may communicate with each other, or they may not have any connections. The neurons of one layer are always connected to the neurons of at least another layer.

3.3.2.1 Inter-layer connections

There are different types of connections used between layers; these connections between layers are called inter-layer connections.

- **Fully connected**
- **Partially connected**
- **Feed forward**
- **Bi-directional.**
- **Hierarchical**
- **Resonance**

3.3.2.2 Intra-layer connections

In more complex structures the neurons communicate among themselves within a layer, this is known as intra-layer connections. There are two types of infra-layer connections.

• **Recurrent** The neurons within a layer are fully- or partially connected to one another. After these neurons receive input form another layer, they communicate their outputs with one another a number of times before they are allowed to send their outputs to another layer.

• **On-center/off surround** A neuron within a layer has excitatory connections to itself and its immediate neighbors, and has inhibitory connections to other neurons. One can imagine this type of connection as a competitive gang of neurons. Each

gang excites itself and its gang members and inhibits all members of other gangs. After a few rounds of signal interchange, the neurons with an active output value will win, and is allowed to update its and its gang member's weights.

3.4 Single-layer Networks: Auto association, Optimization, and Contrast Enhancement

Beyond the instarloutstar neural networks are the single layer intraconnected neural networks. Figure 3.4 shows the topology of a one-layer neural network, which consists of nF_X PEs. The connections from each F_X PE to every other Fx PE and itself, yielding a connection matrix with n^2 entries.

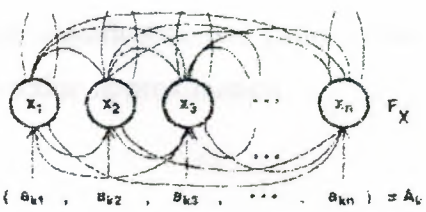


Figure 3.4. Single layer Neural Network

One-layer neural networks are used for pattern completion, noise remove optimization, and contrast enhancement.

3.5 Multi-layer Networks: Heteroassociation and Function approximation

A multi-layer neural network has more than two layers, possibly many more. A general description of a multi-layer neural network is shown in Figure 2.4, where there is an input layer of PEs, Fx, L hidden layers of Fy PEs and a final output layer. Fy, The *Fy* layers are called hidden layers because there are no direct connections between the input/output patterns to these PEs, rather they are always accessed through another set of PEs such as the input and output PEs. The added benefit of these PEs is not fully understood, but many applications such as prediction and classification are employing these types of topologies.

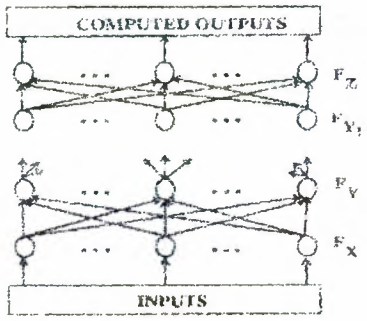


Figure 3.5. General Multi-layer Neural Network

Multi-layer neural networks are used for pattern classification, pattern matching and function approximation. This capability allows some very complex decision regions to be performed for classification and pattern matching problems, as well as applications that require function approximation.

3.6 Neural Network Learning

Perhaps the most appealing quality of neural networks is their ability to learn. Learning, in this context, is defined as a change in connection weight values that results in the capture of information that can later be recalled. There are several different procedures available for changing the values of connection weights.

3.6.1 Supervised vs. Unsupervised Learning

All learning methods can be classified into two categories- supervised learning and unsupervised learning. Supervised learning is a process that incorporates an external teacher and/or global information. The supervised learning algorithms that will be discussed in the following sections include error correction learning, reinforcement learning, stochastic learning, and hardwired systems. Examples of supervised learning include; deciding when to turn off the learning, deciding how long and how often to present each association for training, and supplying performance (error) information. Supervised learning is further classified

into two subcategories; structural learning and temporal learning. Structural learning is concerned with finding the best possible input/output relationship for each individual pattern pair.

Unsupervised learning, also referred to as self-organization, is a process that incorporates no external teacher and relies upon only local information during the entire learning process. Supervised learning organizes presented data and discovers its emergent collective properties. Examples of unsupervised learning that will be discussed in the following sections includes Hebbian learning, principle component learning, differential Hebbian learning, min-max learning, and competitive learning.

3.6.2 Off-line vs. On-line Learning

Most learning techniques utilize off-line learning. When the entire pattern set is used to condition the connections prior to the use of the network, it is called off-line learning. As an example, the back propagation training algorithm is used to adjust connections in multilayer neural network, but it requires thousands of cycles through all the pattern pairs until the desired performance of the network has been achieved. Once the network is performing adequately, the weights are frozen and the resulting network is used in recall mode thereafter. Off-line learning systems have the intrinsic requirement that all the patterns have to be resident for training. Such a requirement does not make it possible to have new patterns automatically incorporated into the network as they occur, rather these new patterns must be added to the entire set of patterns and a retraining of the neural network must be done again.

3.6.3 Hebbian Correlations

The simplest form of adjusting connection weight values in a neural network is based upon the correlation of PE activation values. The motivation for correlation-based adjustments has been attributed to Hebb (1949) who hypothesized that the change in a synapses efficacy (its ability to fire, or as we are simulating it in our

neural networks, the connection weight) is prompted by a neuron's ability to produce an output signal. If a neuron. A, was active, and A's activity caused a connected neuron, B, to fire, then the efficacy of the synaptic connection between A and B should be increased.

3.6.4 Principle Component Learning

There are some neural networks that have learning algorithms designed to produce, as a set of weights, the principle components of the put data patterns. I he principle components of a set of data are found by forming the covariance (or correlation) matrix of a set of patterns and then finding the minimal set of orthogonal vectors that span the space of the covanance matrix.

3.6.5 Differential Hebbian Learning

Hebbian learning has been extended to capture the temporal changes that occur in pattern sequences. This learning law, entitled Differential Hebbian Learning, has been independently derived by Klopf (1986) in the discrete time form and by Kosko (1986) in the continuous time form. The general form, some variants, and some similar learning laws are outlined in the following sections.

3.6.6 Competitive Learning

Competitive learning is a method of automatically creating classes for a set of input patterns. Competitive learning is a two step procedure that couples the recall process with the learning process in a two layer neural network (see Figure 3.6).

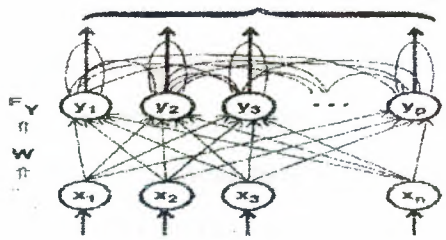


Figure 3.6 Competitive learning neural network

3.6.7 Min-Max Learning

Min-max classifier systems utilize a pair of vectors for each class For the class j, represented by the PE y, and defined by the abutting vectors V, (the min vector) and

(the max vector). Learning in a min-max neural system is done using the equation

for the min vector and $V_{ij}^{new} = \min(a_k i, V_{ij}^{old})$

for the max vector. $V_{ij}^{new} = \max(a_k i, V_{ij}^{old})$

3.6.8 Error Correction Learning

Error correction learning adjusts the connection weights between PEs in proportion to the difference between the desired and computed values of each output layer PE. Two layer error correction learning is able to capture linear mappings between input and output patterns. Multi-layer error correction learning is able to capture nonlinear mappings between the inputs and outputs.

3.6.9 Reinforcement Learning

Reinforcement learning is similar to error correction learning in that weights are reinforced for properly performed actions and punished for poorly performed actions. The difference between these two supervised learning techniques is that error correction learning utilizes more specific error information by collecting error values from each output layer PE, while reinforcement learning uses non-specific error information to determine the performance of the network. Where error-correction learning has a whole vector of values that it uses for error correction, only one value is used to describe the output layer's performance during reinforcement learning. This form of learning is ideal in situations where specific error information is not available, but overall performance information is, such as prediction and control.

A two-layer neural network such as the one found in Figure 2.6 serves as a good framework for the reinforcement learning algorithm. The general reinforcement learning equation is

$$W_{ij}^{new} = W_{ij}^{old} + \alpha(r-\theta_j)e_{ij}$$

where, r is the scalar success/failure value provided by the environment, 0, is the reinforcement threshold value for the j'th F_y PE, e,,; is the canonical eligibility of the weight from the i'th F., PE to the j'th F_Y PE, and $0<a< 1$ is a constant-valued learning rate. In error correction learning, gradient descent in error space controlled learning. In reinforcement learning it is gradient descent in probability space.

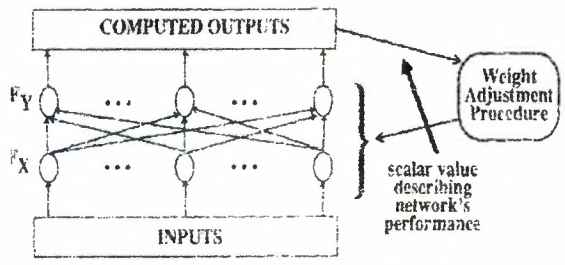


Figure 3.7. Reinforcement learning Neural Network

The canonical eligibility of w_{ij} is dependant on a previously selected probability distribution that is used to determine if the computed output value equals the desired output value and is defined as

$$E_{ij}=\delta/(\delta w_{ij})\ln g_i$$

where g, is the probability of the desired output equaling the computed output, defined as

$$g_i = Pr(y_j=b_{kj}/W_j,A_k)$$

which is read as the probability that yj equals b_{kj} given the input, A_k, and the corresponding weight vector, W_j.

3.7 Error Backpropagation

We have already seen how to train linear networks by gradient descent. In trying to do the same for multi-layer networks we encounter a difficulty: we don't have any target values for the hidden units. This unsolved question was in fact the reason why neural networks fell out of favor after an initial period of high popularity in the 1950s. It took 30 years before the error backpropagation (or in short: backprop) algorithm popularized a way to train hidden units, leading to a new wave of neural network research and applications.

In principle, backprop provides a way to train networks with any number of hidden units arranged in any number of layers. (There are clear practical limits,

which we will discuss later.) In fact, the network does not have to be organized m layers - any pattern of connectivity that permits a partial ordering of the nodes from input to output is allowed.

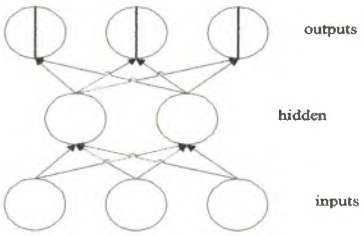


Figure 3.8 Simple structure of neural network

In other words, there must be a way to order the units such that all connections go from "earlier" (closer to the input) to "later" ones (closer to the output). This is equivalent to stating that their connection pattern must not contain any cycles. Networks that respect this constraint are called feedforward networks; their connection pattern forms a directed acyclic graph or dag.

3.7.1 The Algorithm

We want to train a multi-layer feedforward network by gradient descent to approximate an unknown function, based on some training data consisting of pairs **(x,t)**. The vector x represents a pattern of input to the network, and the vector t the corresponding **target** (desired output). As we have seen before, the overall gradient with respect to the entire training set is just the sum of the gradients for each pattern; in what follows we will therefore describe how to compute the gradient for just a single training pattern. As before, we will number the units, and denote the weight from units to unit i by w_{ij}

Definitions:

• the **error** signal for unit j:

$$\delta_j = -\partial E / \partial \text{net}_j$$

• the (negative) **gradient** for weight w,,

$$\Delta w_{ij} = -\partial E / \partial w_{ij}$$

• the set of nodes **anterior** to unit i:

$$Ai = \{j : \exists w_{ij}\}$$

• the set of nodes **posterior** to unit j:

$$Pj = \{i : \exists w_{ij}\}$$

The gradient. As we did for linear networks before, we expand the gradient into two factors by use of the chain rule:

$$\Delta w_{ij} = -\frac{\partial \mathrm{E}}{\partial net_i} \frac{\partial net_i}{\partial w_{ij}}$$

The first factor is the error of unit i. The second is

$$\frac{\partial net_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{k \in A_i} w_{ik} y_k = y_j$$

Putting the two together, we get

$$\Delta w_{ij} = \delta_i y_j$$

To compute this gradient, we thus need to know/ the activity and the error for al! relevant nodes in the network.

Forward activation. The activity of the input units is determined by the network's external input x. For all other units, the activity is propagated forward:

$$y_i = f_i(\sum_{j \in A_i} w_{ij} y_j)$$

Note that before the activity of unit i can be calculated,, the activity of all its anterior nodes (forming the set A,) must be known. Since feedforward networks do not contain cycles, there is an ordering of nodes from input to output that respects this condition.

Calculating output error. Assuming that we are using the sum-squared loss

$$E = \frac{1}{2}\sum_{0}(t_0 - y_0)^2$$

the error for output unit o is simply

$$\delta_0 = t_0 - y_0$$

Error backpropagation, For hidden units, we must propagate the error back from the output nodes (hence the name of the algorithm). Again using the chain rule, we can expand the error of a hidden unit in terms of its posterior nodes:

$$\delta_j = -\sum_{i \in P_i} \frac{\partial E}{\partial net_i} \frac{\partial net_i}{\partial y_j} \frac{\partial y_j}{\partial net_j}$$

Of the three factors inside the sum, the first is just the error of node i. The second is

$$\frac{\partial y_j}{\partial net_j} = \frac{\partial f_i(net_j)}{\partial net_j} = f_j(net_j)$$

while the third is the derivative of node j's activation function:

For hidden units h that use the tanh activation function, we can make use of the special identity' tanh(u)' = 1 - tanh(u)"", giving us

$$f'_h(net_h) = 1 - y_h^2$$

Putting all the pieces together we gel

$$\delta_j = f_j'(net_j)\sum_{i \in P_j} \delta_i w_{ij}$$

Note that in order to calculate the error for unit j, we must first know the error of all its posterior nodes (forming the set P,). Again, as long as there are no cycles in the network, there is an ordering of nodes from the output back to the input that respects this condition. For example, we can simply use the reverse of the order in which activity was propagated forward.

3.7.2 Matrix Form

For layered feed forward networks that are **fully connected** - that is, each node in a given layer connects to *every* node in the next layer - it is often more convenient to write the backprop algorithm in matrix notation rather than using more general graph form given Above. In this notation, the biases weights, net inputs, activations, and error signals for all units in a layer are combined into vectors, while all the non-bias weights from one layer to the next form a matrix W. Layers are numbered from 0 (the input layer) to I- (the output layer). The backprop algorithm then looks as follows:

Initialize the input layer:

$$\vec{y}_0 = \vec{x}$$

Where bi is the vector of bias weights. Calculate the error in
the output layer:

$$\vec{y}_i = f_i(W_i \vec{y}_{i-1} + \vec{b}_i)$$

Backpropagation the error: for I = L-l, L-2, 1

$$\vec{\delta}_L = \vec{i} - \vec{y}_L$$

where T is the matrix transposition operator. Update the weights and biases:

$$\vec{\delta}_i = (W_{i+1}^T \vec{\delta}_{i+1}).f_i'(net\vec{}_i)$$

We can see that this notation is significantly more compact than the graph form, even though it describes exactly the same sequence of operations.

$$\Delta W_l = \vec{\delta}_l \vec{y}_{l-1}^T \qquad \Delta \vec{b}_l = \vec{\delta}_l$$

CHAPTER FOUR

GENETIC ALGORITHM

4.1 Basic Description

Genetic algorithms are inspired by Darwin's theory about evolution. Solution to a problem solved by genetic algorithms is evolved.

Algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness ,the more suitable they are the more chances they have to reproduce.

This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied.

4.2 Outline of the Basic Genetic Algorithm

1. [Start] Generate random population of n chromosomes (suitable solutions for the problem)
2. [Fitness] Evaluate the fitness $f(x)$ of each chromosome x in the population
3. [New population] Create a new population by repeating following steps until the new population is complete

    a. [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)

    b. [Crossover] With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.

c. [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
d. [Accepting] Place new offspring in a new population

4. [Replace] Use new generated population for a further run of algorithm
5. [Test] If the end condition is satisfied, stop, and return the best solution in current population
6. [Loop] Go to step 2

Some Comments:

As you can see, the outline of Basic GA is very general. There are many things that can be implemented differently in various problems.

First question is how to create chromosomes, what type of encoding choose. With this is connected crossover and mutation; the two basic operators of GA. Encoding, crossover and mutation are introduced in next chapter.

Next questions are how to select parents for crossover. This can be done in many ways, but the main idea is to select the better parents (in hope that the better parents will produce better offspring). Also you may think, that making new population only by new offspring can cause lost of the best chromosome from the last population. This is true, so so called elitism is often used. This means, that at least one best solution is copied without changes to a new population, so the best solution found can survive to end of run.

Maybe you are wandering, why genetic algorithms do work. It can be partially explained by Schema Theorem (Holland), however, this theorem has been criticized in recent time. If you want to know more, check other resources.

4.3 Operators of GA

As you can see from the genetic algorithm, the crossover and mutation are the most important part of the genetic algorithm. The performance is influenced mainly by

these two operators. Before we can explain more about crossover and mutation, some information about chromosomes will be given.

4.3.1 Encoding of a Chromosome

The chromosome should in some way contain information about solution that it represents. The most used way of encoding is a binary string. The chromosome then could look like this:

Chromosome 1	1101100100110110
Chromosome 2	1101111000011110

Each chromosome has one binary string. Each bit in this string can represent some characteristic of the solution. Or the whole string can represent a number - this has been used in the basic GA.

Of course, there are many other ways of encoding. This depends mainly on the solved problem. For example, one can encode directly integer or real numbers, sometimes it is useful to encode some permutations and so on.

4.3.2 Crossover

After we have decided what encoding we will use, we can make a step to crossover. Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way how to do this is to choose randomly some crossover point and everything before this point copy from a first parent and then everything after a crossover point copy from the second parent.

Crossover can then look like this (| is the crossover point):

Chromosome 1	11011 | 00100110110
Chromosome 2	11011 | 11000011110
Offspring 1	11011 | 11000011110

Offspring 2	11011	00100110110

There are other ways to make crossover, for example we can choose more crossover points. Crossover can be rather complicated and very depends on encoding of the encoding of chromosome. Specific crossover made for a specific problem can improve performance of the genetic algorithm.

4.3.3 Mutation

After a crossover is performed, mutation takes place. This is to prevent falling all solutions in population into a local optimum of solved problem. Mutation changes randomly the new offspring. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. Mutation can then be following:

Original offspring 1	1101111000011110
Original offspring 2	1101100100110110
Mutated offspring 1	1100111000011110
Mutated offspring 2	1101101100110110

The mutation depends on the encoding as well as the crossover. For example when we are encoding permutations, mutation could be exchanging two genes.

4.4 Parameters of Genetic Algorithms

4.4.1 Crossover and Mutation Probability

There are two basic parameters of GA - crossover probability and mutation probability.

Crossover probability says how often will be crossover performed. If there is no crossover, offspring is exact copy of parents. If there is a crossover, offspring is made from parts of parents' chromosome. If crossover probability is 100%, then all offspring is made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!).Crossover is made in hope that new chromosomes will have good parts of old

chromosomes and maybe the new chromosomes will be better. However it is good to leave some part of population survive to next generation.

Mutation probability says how often will be parts of chromosome mutated. If there is no mutation, offspring is taken after crossover (or copy) without any change. If mutation is performed, part of chromosome is changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed. Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because then GA will in fact change to random search.

4.4.2 Other Parameters

There are also some other parameters of GA. One also important parameter is population size.

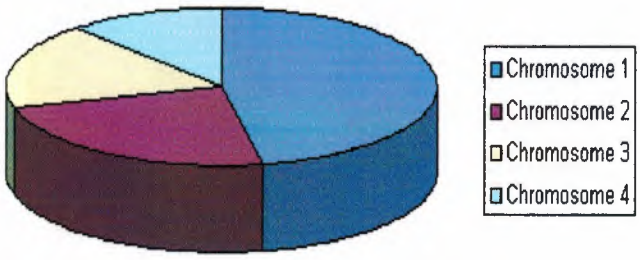
Population size says how many chromosomes are in population (in one generation). If there are too few chromosomes, GA have a few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to increase population size, because it does not make solving the problem faster.

4.5 Selection

As you already know from the Genetic algorithm outline, chromosomes are selected from the population to be parents to crossover. The problem is how to select these chromosomes. According to Darwin's evolution theory the best ones should survive and create new offspring. There are many methods how to select the best chromosomes, for example roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others.

4.5.1. Roulette Wheel Selection

Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Imagine a roulette wheel where are placed all chromosomes in the population, every has its place big accordingly to its fitness function, like on the following picture.



Then a marble is thrown there and selects the chromosome. Chromosome with biggest fitness will be selected more times.

Following algorithm can simulate this.

- [Sum] Calculate sum of all chromosome fitness's in population - sum *S*.
- [Select] Generate random number from interval *(0,S)* - *r*.
- [Loop] Go through the population and sum fitness's from *0* - sum *s*. When the sum *s* is greater then *r*, stop and return the chromosome where you are.

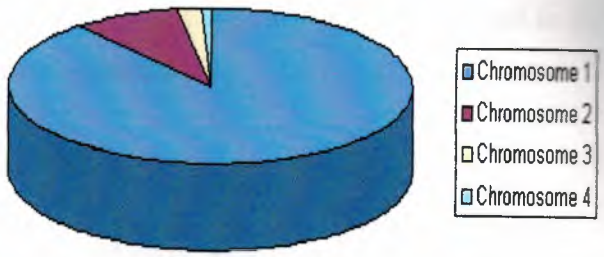
Of course, step 1 is performed only once for each population.

4.5.2 Rank Selection

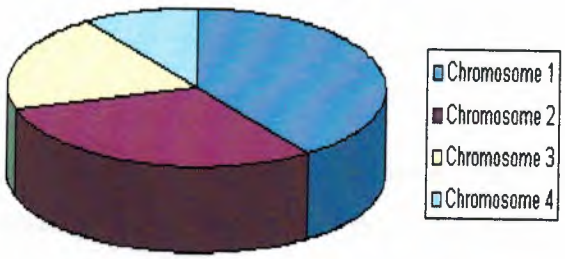
The previous selection will have problems when the fitness's differs very much. For example, if the best chromosome fitness is 90% of the entire roulette wheel then the other chromosomes will have very few chances to be selected.

Rank selection first ranks the population and then every chromosome receives fitness from this ranking. The worst will have fitness *1*, second worst *2* etc. and the best will have fitness *N* (number of chromosomes in population).

You can see in following picture, how the situation changes after changing fitness to order number.



Situation before ranking (graph of fitness's)



Situation after ranking (graph of order numbers)

After this all the chromosomes have a chance to be selected. But this method can lead to slower convergence, because the best chromosomes do not differ so much from other ones.

4.5.3 Steady-State Selection

This is not particular method of selecting parents. Main idea of this selection is that big part of chromosomes should survive to next generation.

GA then works in a following way. In every generation is selected a few (good - with high fitness) chromosomes for creating a new offspring. Then some (bad - with low fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to new generation.

4.5.4 Elitism

Idea of elitism has been already introduced. When creating new population by crossover and mutation, we have a big chance, that we will loose the best chromosome.

Elitism is name of method, which first copies the best chromosome (or a few best chromosomes) to new population. The rest is done in classical way. Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution.

4.6 Encoding

Encoding of chromosomes is one of the problems, when you are starting to solve problem with GA. Encoding very depends on the problem. In this section will be introduced some encodings, which have been already used with some success.

4.6.1 Binary Encoding

Binary encoding is the most common, mainly because first works about GA used this type of encoding.

In binary encoding every chromosome is a string of bits, 0 or 1.

Chromosome A	10110010110010101110010 1

sometimes corrections must be made after crossover and/or mutation.

Example of Problem: Knapsack problem
The problem: There are things with given value and size. The knapsack has given capacity. Select things to maximize the value of things in knapsack, but do not extend knapsack capacity.
Encoding: Each bit says, if the corresponding thing is in knapsack.

4.6.2 Permutation Encoding

Permutation encoding can be used in ordering problems, such as travelling salesman problem or task ordering problem.

In permutation encoding, every chromosome is a string of numbers, which represents number in a sequence.

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Example of chromosomes with permutation encoding

Permutation encoding is only useful for ordering problems. Even for this problems for some types of crossover and mutation corrections must be made to leave the chromosome consistent (i.e. have real sequence in it).

Example of Problem: Travelling salesman problem (TSP)
The problem: There are cities and given distances between them. Travelling salesman has to visit all of them, but he does not to travel very much. Find a sequence of cities to minimize travelled distance.
Encoding: Chromosome says order of cities, in which salesman will visit them.

4.6.3 Value Encoding

Direct value encoding can be used in problems, where some complicated values, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult.

In value encoding, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or chars to some complicated objects.

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

Example of chromosomes with value encoding

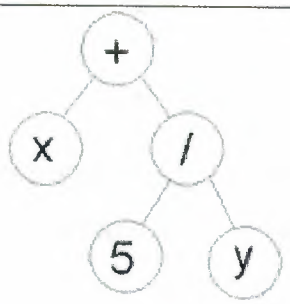
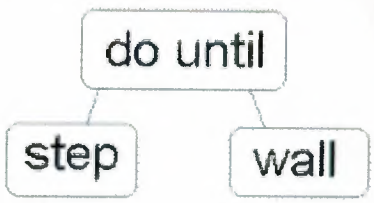
Value encoding is very good for some special problems. On the other hand, for this encoding is often necessary to develop some new crossover and mutation specific for the problem.

Example of Problem: Finding weights for neural network
the problem: There is some neural network with given architecture. Find weights for inputs of neurons to train the network for wanted output.
Encoding: Real values in chromosomes represent corresponding weights for inputs.

4.6.4 Tree Encoding

Tree encoding is used mainly for evolving programs or expressions, for genetic programming.

In tree encoding every chromosome is a tree of some objects, such as functions or commands in programming language.

Chromosome A	Chromosome B
	
(+ x (/ 5 y))	(do until step wall)

Example of chromosomes with tree encoding.

Tree encoding is good for evolving programs. Programing language LISP is often used to this, because programs in it are represented in this form and can be easily parsed as a tree, so the crossover and mutation can be done relatively easily.

Example of Problem: Finding a function from given values *the problem*: Some input and output values are given. Task is to find a function, which will give the best (closest to wanted) output to all inputs.

Encoding: Chromosome is a function represented in a tree.

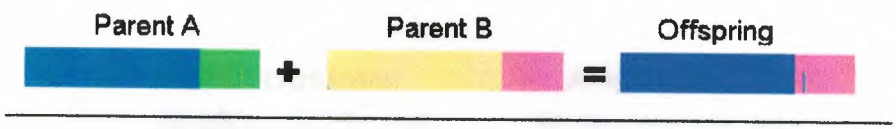
4.7 Crossover and Mutation

Crossover and mutation are two basic operators of GA. Performance of GA very depends on them. Type and implementation of operators depends on encoding and also on a problem. There are many ways how to do crossover and mutation. There are only some examples and suggestions how to do it for several encoding.

4.7.1 Binary Encoding

4.7.1.1 Crossover

Single point crossover - one crossover point is selected, binary string from beginning of chromosome to the crossover point is copied from one parent, the rest is copied from the second parent .



11001011+11011**111** = **11001111**

Two point crossover - two crossover point are selected, binary string from beginning of chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent

11001011 + 1**1011111** = **11011111**

Uniform crossover - bits are randomly copied from the first or from the second parent

11001011 + 11011101 = 11011111

Arithmetic crossover - some arithmetic operation is performed to make a new offspring

11001011 + 11011111 = 11001001 (AND)

4.7.1.2 Mutation

Bit inversion - selected bits are inverted

11001001 => 10001001

4.7.2 Permutation Encoding

4.7.2.1 Crossover

Single point crossover - one crossover point is selected, till this point the permutation is copied from the first parent, then the second parent is scanned and if the number is not yet in the offspring it is added
Note: there are more ways how to produce the rest after crossover point

(1 2 3 4 5 6 7 8 9) + (4 5 3 6 8 9 7 2 1) = (1 2 3 4 5 6 8 9 7)

4.7.2.2 Mutation

Order changing - two numbers are selected and exchanged

(1 **2** 3 4 5 6 **8** 9 7) => (1 **8** 3 4 5 6 **2** 9 7)

4.7.3 Value Encoding

4.7.3.1 Crossover

All crossovers from binary encoding can be used

4.7.3.2 Mutation

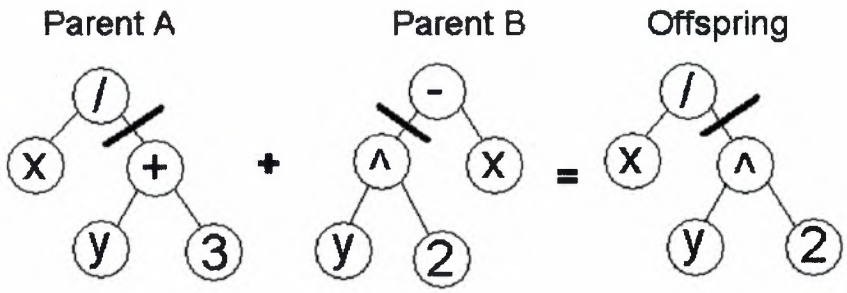
Adding a small number (for real value encoding) - to selected values is added (or subtracted) a small number

(1.29 5.68 **2.86** **4.11** 5.55) => (1.29 5.68 **2.73** **4.22** 5.55)

4.7.4 Tree Encoding

4.7.4.1 Crossover

Tree crossover - in both parent one crossover point is selected, parents are divided in that point and exchange part below crossover point to produce new offspring



CHAPTER FIVE
HYBRID SYSTEMS

5.1 Introduction

Ordinary hybrid systems are defined in many different ways. In a simple way, hybrid system are those composed by more than one intelligent system. Hybrid systems are expected to be more powerful due to the combining advantages of different intelligent techniques.

Two or more intelligent systems can be combined to create a unique hybrid system. The most popular hybrid systems are:

5.1.1 Sequential Hybrid System

This model represent the weakest degree of integration and it is composed of two intelligent systems connected in serial (Fig.5.1.a). One example of this type of system may be a pre-processor Fuzzy System activating a Neural Net.

5.1.2 Auxiliary Hybrid System

This model is composed of a sub-system added by another intelligent sub-system. The integration degree is greater than in the previous case (Fig.5.1.b). An example of this kind of systems is a Genetic Algorithm used to determine the weights of a Neural Net.

5.1.3 Incorporated Hybrid System

Incorporated hybrid systems represent the greatest degree of integration. There is no possible differentiation between the different intelligent systems.We can say that the first system contains the second one or vice-versa. An example is a Neuro-Fuzzy system, where a Fuzzy inference system is implemented using a Neural Net structure (Fig. 5.1.c).

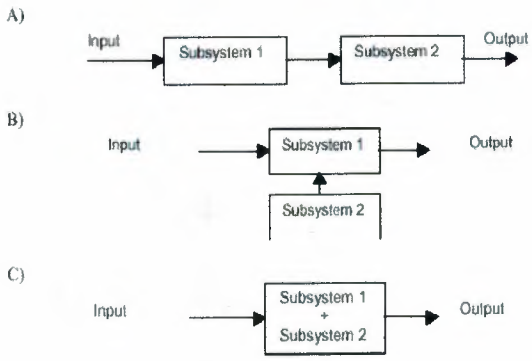


Figure 5.1. Hybrid Systems

Among the most popular hybrid models are the Neuro-Fuzzy systems, Neuro-Genetic systems and Fuzzy-Genetic systems.

5.2 Neuro Fuzzy Systems

In these, the most widely researched of all the hybrid systems; fuzzy logic provides a structure within which the learning ability of neural networks is employed. In this field there are a number of possible uses. Firstly, neural networks can be used to generate the membership functions for a fuzzy system and to tune them; a schematic for this is shown in figure5.2.

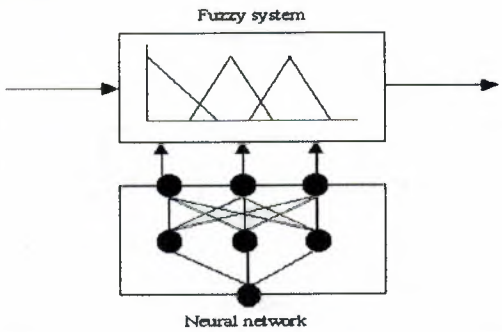


Figure 5.2 Neuro-fuzzy system for tuning a membership function

Fuzzy systems and neural networks may also be combined in series; the neural network performs a pre-processing or post-processing role in cases where, respectively, a sensor output is not suitable as a direct input to the fuzzy system, or the fuzzy system's output is not suitable for direct connection to external devices. In the latter case the neural network performs a mapping that would not easily be carried out

with analytical techniques. Figure 2 shows the pre-processing system; a post-processing system would simply have the fuzzy system and neural network reversed.

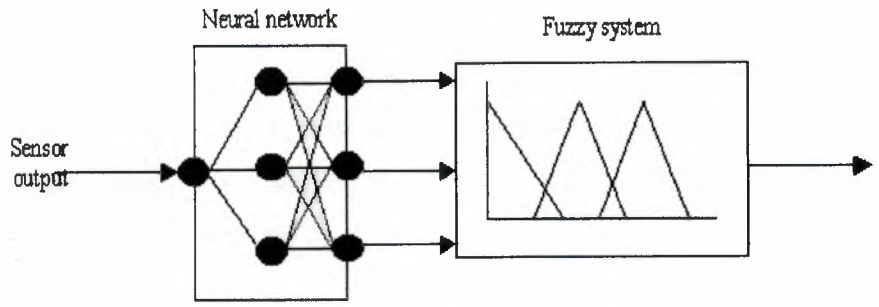


Figure 5.3 Neuro fuzzy system with neural network in pre-processing role

Parallel systems also exist, and an example of this is a system in which a neural network fine-tunes the output of a fuzzy system, according to what it has learnt from the fine adjustments that users have previously made. The schematic for such a system is shown in figure 3.

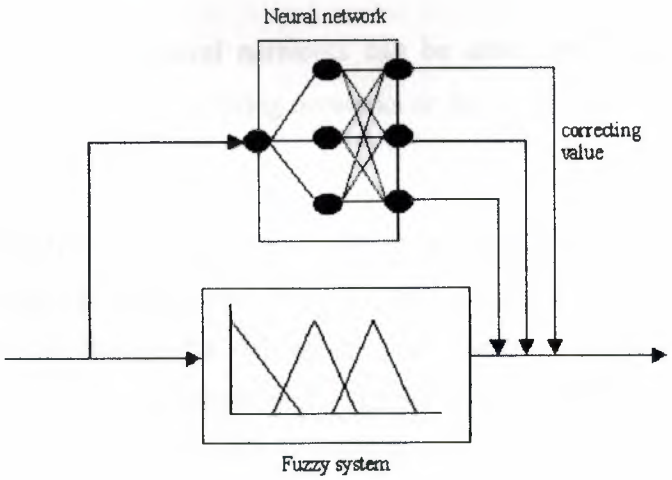


Figure 5.4. Parallel neural network/fuzzy system combination for fine-tuning an output

A system called ANFIS (Adaptive Network-based Fuzzy Inference System), in which neural networks are used to implement a fuzzy inference system. A fuzzy inference system consists of three components. Firstly, a rule base contains a selection of fuzzy rules. Secondly, a database defines the membership functions used in the rules and, finally, a reasoning mechanism carries out the inference procedure on the

rules and given facts. The concept of fuzzy reasoning is straightforward. The truth of a proposition A infers the truth of a proposition B by the implication.

$$A \Rightarrow B$$

For example, if A represents "the banana is yellow" and B represents "the banana is ripe", then if "the banana is yellow" it is inferred that "the banana is ripe". Fuzzy reasoning then allows the inference that if "the banana is more or less yellow" then "the banana is more or less ripe".

Neuro-fuzzy systems have become popular in several fields. Control is a notable example - particularly space and aviation applications, where auto-pilots aim to mimic human ability to make reasoned judgments. In another example, Lee et al describe a system for face recognition in which various features are extracted from the face and fuzzified to make them less sensitive to variation of features of the same person. The fuzzified features are then applied to a neural network for the recognition process.

5.3 Neuro Gentic Systems

The performance of neural networks can be enhanced by the use of genetic algorithms. Possibilities are evolving networks or the use of the algorithms to change network parameters.

A suggested use of a neuro-genetic system is in attitude control of a satellite. The *attitude* of a satellite is its orientation in space and is important in particular in avoiding solar and atmospheric damage, to point antennae and to orient rockets for manoeuvres. A detailed implementation is given, but the essence of the system is that what is termed a *Local Prediction Network* learns to predict the future system state given the previous states and current control inputs. Having been trained, the network can output thousands of predictions of the effect of a hypothetical control input. The best possible input is then selected using a genetic algorithm. The genetic controller could itself use a neural network to carry out the time-consuming task of fitness evaluation.

5.4 Fuzzy Neural Networks

Here, separate neural network layers perform the operations of fuzzification and defuzzification on crisp input and output data, and implement the fuzzy rules. The structure is shown in figure 5.5.

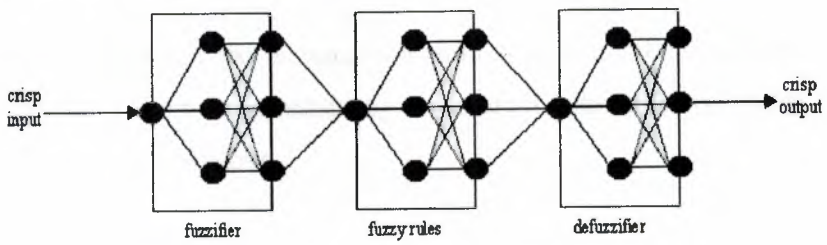


Figure 5.5. Structure of a fuzzy neural network

Buckley and Hayashi describe applications of such networks to fuzzy regression (discovering functional relationships between fuzzy data), control, solving fuzzy matrices (which are used in economics) and in fuzzy classification (described in the next section).

5.5 Implementation of Neuro-Fuzzy Systems Through Interval Mathematics

Neural network performance is dependent on the quality and quantity of training samples presented to the network. Sometimes, when the training data set is small, or perhaps not fully representative of the possibility space, utilization of fuzzy techniques improves performance. One way to carry out this improvement is to represent imprecise data with fuzzy numbers. The neuro-fuzzy system presented is a neural network that processes fuzzy numbers.

Processing fuzzy numbers can be accomplished in a variety of ways. One of the most elegant, because of its simplicity, is by using interval methods.

5.5.1 Interval Mathematics For Fuzzy Numbers

Consider a situation where the value of a given input, $x \in \Re$, is uncertain or vague. In this case, it might be logical to express the input as an interval, thereby indicating that the input is known to exist between two real numbers a_1 and a_2, as shown in Figure 5.5. The uncertain value, x, belongs to a closed bounded interval $[a_1,$

$a_2]$. We can then define an interval number, A, as the set of real numbers x such that $a_1 \leq x \leq a_2$, or

$$A = [a_1, a_2] = \{ x \mid a_1 \leq x \leq a_2, x \in \Re \}. \tag{5.1}$$

Given that we can express an uncertain input as an interval number, the operations on this input value are then governed by the interval arithmetic operations. The basic operations are outlined below:

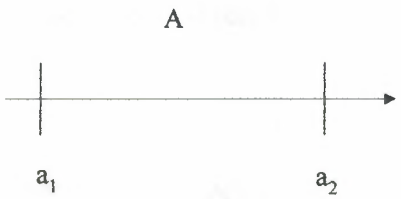


Figure 5.6. An interval number, A.

Addition of intervals

$$\begin{aligned} A + B &= [a_1, a_2] + [b_1, b_2] \\ &= [a_1 + b_1, a_2 + b_2] \end{aligned} \tag{5.2}$$

Subtraction of intervals

$$\begin{aligned} A - B &= [a_1, a_2] - [b_1, b_2] \\ &= [a_1 - b_2, a_2 - b_1] \end{aligned} \tag{5.3}$$

Multiplication of intervals

$$\begin{aligned} A \bullet B &= [a_1, a_2] \bullet [b_1, b_2] \\ &= [\min(a_1b_1, a_1b_2, a_2b_1, a_2b_2), \\ &\quad \max(a_1b_1, a_1b_2, a_2b_1, a_2b_2)] \end{aligned} \tag{5.4}$$

Division of intervals

$$\begin{aligned} A \div B &= [a_1, a_2] \div [b_1, b_2] \\ &= [a_1, a_2] \bullet [\frac{1}{b_2}, \frac{1}{b_1}], \qquad 0 \notin [b_1, b_2] \end{aligned} \tag{5.5}$$

Fuzzy numbers are a generalization of interval numbers [4]. We can interpret the exact value of x (expressed as an interval number, A) as being any number in the given interval, with all values equally possible. The generalization to a fuzzy number, $\tilde{A}$, would be that not all values in the interval are equally possible. The degree to which they are possible can then be interpreted as the membership function, i.e. the degree to which they are members of the interval. The membership function,

$$\mu_{\tilde{A}}: \mathbf{X} \rightarrow [0,1], \tag{5.6}$$

maps numbers in the interval to the interval of real numbers from 0 to 1, inclusive.

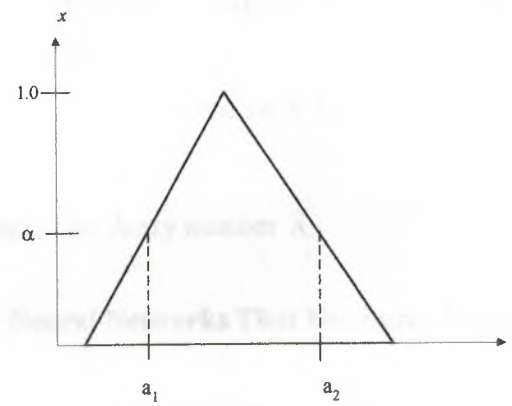


Figure 5.7. A α-cut of a fuzzy number.

There are many variations in describing fuzzy numbers. We shall confine our discussion to triangular fuzzy numbers. A triangular fuzzy number, $\tilde{A}$, is depicted in Figure 2. It is defined by the membership function

$$\mu_{\tilde{A}}(x) = \begin{cases} \dfrac{x-a_1}{a_M - a_1}, & \text{for } a_1 \le x \le a_M, \text{ and} \\ \dfrac{x-a_2}{a_M - a_2}, & \text{for } a_M \le x \le a_2, \end{cases} \tag{5.7}$$

where $[a_1, a_2]$ is the supporting interval and the point $(a_M, 1)$ is the peak.

An α-cut of a fuzzy number $\tilde{A}$ is an interval number A_α that contains all the values of real numbers that have a membership grade in $\tilde{A}$ greater than or equal to the specified value of α. This can be written as

$$A_\alpha = [a_1, a_2]$$
$$= \{x \in \tilde{A} \mid \mu_{\tilde{A}}(x) \geq \alpha\}. \qquad (5.8)$$

Thus, by taking an α-cut of a fuzzy number, one can process the operations on fuzzy numbers via the interval operations described in equations 1 through 4. It is interesting to note that the set of all α-cuts of any triangular fuzzy number is a family of nested intervals.

The level set of $\tilde{A}$ is the set of all levels $\alpha \in [0,1]$ that represent distinct α-cuts of the given fuzzy number $\tilde{A}$. Formally,

$$\Lambda_{\tilde{A}} = \{\alpha \mid \mu_{\tilde{A}}(x) = \alpha \text{ for some } x \in \tilde{A}\}, \qquad (5.9)$$

where $\Lambda_{\tilde{A}}$ denotes the level set of the fuzzy number $\tilde{A}$.

5.5.2 A Learning Rule For Neural Networks That Use Fuzzy Numbers

The neural network described here is based on a standard, feed-forward network, commonly called a multi-layer perceptron. It differs from the perceptron in its use of fuzzy signals. The difference can be readily examined when the formal definition for the neuro-fuzzy system (NFS) is given:

$$NFS = (\mathbf{F}_n, \mathbf{F}_m, \tilde{\mathbf{I}}, \tilde{\mathbf{O}}, \mathbf{A}, \mathbf{L}, f) \qquad (5.10)$$

where

$\mathbf{F}_n$ is the input field, an ordered array of neurons of size $n \times 1$,

$\mathbf{F}_m$ is the output field, an ordered array of neurons of size $m \times 1$,

$\tilde{\mathbf{I}}$ is the fuzzy input vector, $(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n) \in \Re^n$, where $\tilde{x}_i$ corresponds to the input to neuron i in the input field, F_x,

$\tilde{\mathbf{O}}$ is the fuzzy output vector,

$(\tilde{y}_1, \tilde{y}_2, ..., \tilde{y}_m) \in \Re^m$, where $\tilde{y}_i$ corresponds to the output from neuron i in the output field, F_y,

$\mathbf{A}: \Re^n \rightarrow \Re^m$ is the association function,

$\mathbf{L} \subseteq \mathbf{A}$ is the set of learned associations, and

$f: \Re^k \rightarrow \Re$ is the neurons' activation function.

It was originally presented as a neural network that learned from fuzzy If-Then rules. This network configuration can be used in several ways, the key to which is taking α-cuts of the fuzzy number in question and utilizing interval mathematics.

The basic structure of the neural network relies on neurons that have weights and an activation function that are crisp. The input, target and output signals for this system, then, are described by vectors composed of fuzzy numbers and are processed by taking α-cuts. The resulting intervals are manipulated using interval mathematics to adjust the weights. Specifically, each α-cut of the fuzzy input vector is represented by the interval vector $\mathbf{X}_p = (X_{p1}, X_{p2}, ..., X_{pn})^T$ where

$$X_{pi} = [x_{pi}^L, x_{pi}^U] \tag{5.11}$$

indicate the lower and upper limits of the interval.

The summation of weighted inputs is carried out as

$$Net_{pj}^L = \sum_{\substack{i \\ w_{ji} \geq 0}} w_{ji} o_{pi}^L + \sum_{\substack{i \\ w_{ji} < 0}} w_{ji} o_{pi}^U + \theta_j \tag{5.12}$$

and

$$Net_{pj}^U = \sum_{\substack{i \\ w_{ji} \geq 0}} w_{ji} o_{pi}^U + \sum_{\substack{i \\ w_{ji} < 0}} w_{ji} o_{pi}^L + \theta_j \ . \tag{5.13}$$

These calculations are consistent with the interval multiplication operation described in equation (5.14). The equation for the output can be expressed as

$$o_{pj} = [o_{pj}^L, o_{pj}^U] = [f(Net_{pj}^L),\ f(Net_{pj}^U)] . \qquad (5.14)$$

The learning algorithm is based on the Generalized Delta rule. That is, there is an error calculated and backpropagated in order to modify the weights. Specifically, The error is computed as the difference between the target output, t_p, and the actual output, o_p:

$$E_p = \max\{\frac{1}{2}(t_{pj} - o_{pj})^2 ,\ o_{pj} \in o_p\}, \qquad (5.15)$$

where

$$(t_{pj} - o_{pj}) = \begin{cases} (t_{pj} - o_{pj}^L), \text{ if } t_p = 1, \text{ and} \\ (t_{pj} - o_{pj}^U), \text{ if } t_p = 0. \end{cases} \qquad (5.16)$$

Succinctly stated, the Generalized Delta rule, indicates that the change in any weight (in any layer) is

$$\Delta w_{ji}(t+1) = \eta(-\frac{\partial E_p}{\partial w_{ji}}) + \alpha \Delta w_{ji}(t) . \qquad (5.17)$$

For units in the output layer, calculation of $\partial E_p / \partial w_{ji}$ is straightforward, and can be thought of as four cases based on the value of target output and weight. Note that in the four equations the value of j in the subscript is fixed (to the output neuron that had the maximum error). In the first case, the $t_p = 1$, and $w_{ji} \geq 0$:

$$\begin{aligned} \frac{\partial E_p}{\partial w_{ji}} &= \frac{\partial}{\partial w_{ji}}\left[\frac{(t_{pj} - o_{pj}^L)^2}{2}\right] \\ &= \frac{\partial}{\partial o_{pj}^L}\left[\frac{(t_{pj} - o_{pj}^L)^2}{2}\right]\frac{\partial o_{pj}^L}{\partial Net_{pj}^L}\cdot\frac{\partial Net_{pj}^L}{\partial w_{ji}} \\ &= -(t_{pj} - o_{pj}^L)\cdot o_{pj}^L \cdot (1 - o_{pj}^L)\cdot o_{pi}^L \\ &= -\delta_{pj}^L \cdot o_{pi}^L . \end{aligned} \qquad (5.18)$$

The third line in the derivation assumes that the neuronal activation function, f (Net), is the binary sigmoid function, and thus substitutes the values accordingly. The second case has $t_p = 1$, and $w_{ji} < 0$:

$$\frac{\partial E_{pj}}{\partial w_{ji}} = -\delta_{pj}^{L} \cdot o_{pi}^{U}, \qquad (5.19)$$

where $\delta_{pj}^{L} = (t_{pj} - o_{pj}^{L}) \cdot o_{pj}^{L} \cdot (1 - o_{pj}^{L})$. The third case is characterized by $t_p = 0$ and $w_{ji} \geq 0$:

$$\begin{aligned} \frac{\partial E_p}{\partial w_{ji}} &= \frac{\partial}{\partial w_{ji}} \left[\frac{(t_{pj} - o_{pj}^{U})^2}{2} \right] \\ &= \frac{\partial}{\partial o_{pj}^{U}} \left[\frac{(t_{pj} - o_{pj}^{U})^2}{2} \right] \frac{\partial o_{pj}^{U}}{\partial Net_{pj}^{U}} \cdot \frac{\partial Net_{pj}^{U}}{\partial w_{ji}} \\ &= -(t_{pj} - o_{pj}^{U}) \cdot o_{pj}^{U} \cdot (1 - o_{pj}^{U}) \cdot o_{pi}^{U} \\ &= -\delta_{pj}^{U} \cdot o_{pi}^{U}. \end{aligned} \qquad (5.20)$$

where $\delta_{pj}^{U} = (t_{pj} - o_{pj}^{U}) \cdot o_{pj}^{U} \cdot (1 - o_{pj}^{U})$. The fourth and final case (when $t_p = 0$ and $w_{ji} < 0$:

$$\frac{\partial E_{pj}}{\partial w_{ji}} = -\delta_{pj}^{U} \cdot o_{pi}^{L} \ . \qquad (5.21)$$

The calculation of the partial derivative $\partial E_p / \partial w_{ji}$ for the hidden layers is based on back-propagating the error as measured at the output layer. The following discussion assumes one hidden layer, although subsequent terms could be derived in the same way for other hidden layers. Since this derivation involves a path of two neurons (output and hidden layer), there are eight cases. For the first case, $t_p = 1$, $w_{kj} \geq 0$ and $w_{ji} \geq 0$, where w_{kj} is the weight on the path from hidden layer neuron j to output layer neuron k, k fixed:

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\left[\frac{(t_{pk} - o^L_{pk})^2}{2}\right]$$

$$= \frac{\partial}{\partial o^L_{pk}}\left[\frac{(t_{pk} - o^L_{pk})^2}{2}\right]\frac{\partial o^L_{pk}}{\partial Net^L_{pk}}\cdot\frac{\partial Net^L_{pk}}{\partial o_{pj}}$$

$$\cdot\frac{\partial o^L_{pj}}{\partial Net^L_{pj}}\cdot\frac{\partial Net^L_{pj}}{\partial w_{ji}}$$

$$= -\delta^L_{pk}\cdot w_{kj}\cdot o^L_{pj}\cdot(1-o^L_{pj})\cdot o^L_{pi} \ . \qquad (5.22)$$

The second case is a variation on the first, in which $t_p = 1$, $w_{kj} \geq 0$ and $w_{ji} < 0$, so the partial derivative becomes:

$$\frac{\partial E_p}{\partial w_{ji}} = -\delta^L_{pk}\cdot w_{kj}\cdot o^L_{pj}\cdot(1-o^L_{pj})\cdot o^U_{pi} \ . \qquad (5.23)$$

The third case is characterized by $t_p = 1$, $w_{kj} < 0$ and $w_{ji} \geq 0$:

$$\frac{\partial E_p}{\partial w_{ji}} = -\delta^L_{pk}\cdot w_{kj}\cdot o^U_{pj}\cdot(1-o^U_{pj})\cdot o^U_{pi} \ . \qquad (5.24)$$

$t_p = 1$, $w_{kj} < 0$ and $w_{ji} < 0$ for the fourth case:

$$\frac{\partial E_p}{\partial w_{ji}} = -\delta^L_{pk}\cdot w_{kj}\cdot o^U_{pj}\cdot(1-o^U_{pj})\cdot o^L_{pi} \ . \qquad (5.25)$$

The fifth through eighth cases deal with a target value of 0. For the fifth case, $t_p = 0$, $w_{kj} \geq 0$ and $w_{ji} \geq 0$:

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\left[\frac{(t_{pk} - o^U_{pk})^2}{2}\right]$$

$$= \frac{\partial}{\partial o^U_{pk}}\left[\frac{(t_{pk} - o^U_{pk})^2}{2}\right]\frac{\partial o^U_{pk}}{\partial Net^U_{pk}}\cdot\frac{\partial Net^U_{pk}}{\partial o^U_{pj}}$$

$$\cdot\frac{\partial o^U_{pj}}{\partial Net^U_{pj}}\cdot\frac{\partial Net^U_{pj}}{\partial w_{ji}}$$

$$= -\delta_{pk}^{U} \cdot w_{kj} \cdot o_{pj}^{U} \cdot (1 - o_{pj}^{U}) \cdot o_{pi}^{U} \ . \qquad (5.26)$$

$t_p = 0$, $w_{kj} \geq 0$ and $w_{ji} < 0$ for the sixth case:

$$\frac{\partial E_p}{\partial w_{ji}} = -\delta_{pk}^{U} \cdot w_{kj} \cdot o_{pj}^{U} \cdot (1 - o_{pj}^{U}) \cdot o_{pi}^{L} \ . \qquad (5.27)$$

In the seventh case, $t_p = 0$, $w_{kj} \geq 0$ and $w_{ji} < 0$:

$$\frac{\partial E_p}{\partial w_{ji}} = -\delta_{pk}^{U} \cdot w_{kj} \cdot o_{pj}^{L} \cdot (1 - o_{pj}^{L}) \cdot o_{pi}^{L} \ . \qquad (5.28)$$

The eighth and final case has $t_p = 0$, $w_{kj} < 0$ and $w_{ji} < 0$:

$$\frac{\partial E_p}{\partial w_{ji}} = -\delta_{pk}^{U} \cdot w_{kj} \cdot o_{pj}^{L} \cdot (1 - o_{pj}^{L}) \cdot o_{pi}^{U} \ . \qquad (5.29)$$

Equations (5.18) through (5.29) are used to code the training function in the simulation. Simulation results are presented in section V.

5.5.3 The Neuro-fuzzy System Applied To Speaker-independent Speech Recognition

The application problem that will serve as a testbench is speaker-independent speech recognition of the eleven vowel sounds from multiple speakers. The vowel data used in this study was originally collected by Deterding, who recorded examples of the eleven steady state vowels of English spoken by fifteen speakers for a "non-connectionist" speaker normalization study. Four male and four female speakers were used to create the training data, and the other four male and three female speakers were used to create the testing data. Robinson carried out a study comparing performance of feed-forward networks with different structures using this data.

The speech signals were low pass filtered at 4.7 kHz and digitized to 12 bits with a 10 kHz sampling rate. Twelfth order linear predictive analysis was carried out on

six 512 sample Hamming windowed segments from the steady part of the vowel. The reflection coefficients were used to calculate 10 log area parameters, giving a 10 dimensional input space. Each speaker thus yielded six frames of speech from eleven vowels. These results in 528 frames from the eight speakers used for the training set, and 462 frames from the seven speakers used to create the testing set. 528 samples is relatively small training set (in standard neural network applications), and thus an excellent testbench for the neuro-fuzzy system.

5.5.4 EXPERIMENTAL RESULTS AND CONCLUSIONS

Speaker-independent speech recognition is an extremely difficult problem. The relatively small size of this particular data set make the problem difficult, too. The Vowel data set has been used in many studies, fraught with poor results. This is true to such a degree that it caused one researcher to claim that "poor results seem to be inherent to the data".

Difficulty notwithstanding, previous studies have obtained recognition rates (best case) of 51% to 59% [5, 9, 10]. The recognition rate obtained with the neuro-fuzzy system is 89%. Results of the simulation are summarized in Table 1 below.

Type of Network	Number of Hidden Neurons	Best Recognition Rate
Std. Neural Network	11	59.3
Std. Neural Network	22	58.6
Std. Neural Network	88	51.1
Neuro-Fuzzy System	11	88.7

Table 5.1. Best Recognition Rates for Standard Neural Networks and the Neuro-Fuzzy System

A recognition rate of 89% surpassed expectations, especially with a data set as diverse as the speaker-independent speech (vowel recognition) problem. The results reinforce the initial claim, that incorporation of fuzzy techniques improves the performance of neural networks. Fuzzy theory has been used successfully in many applications. This study shows that it can be used to improve neural network performance. Furthermore, the simulations presented show that interval mathematics can be used for successful implementation of such neuro-fuzzy systems.

CONCLUSION

The construction of control system on the base of traditional technology for complicated processes characterizing with non-linearity and uncertainty is not enough to satisfy such characteristics as high speed, reliability, adequacy, and accuracy of the model. In this condition one of the perspective way of construction of control system is the use of soft computing technology, such as neural networks, fuzzy logic and genetic algorithms.

For this reason in the project architecture, functioning principle of soft computing elements, neural networks, fuzzy logic and genetic algorithms are described. The combination of these technologies allows us to create more powerful intelligent hybrid systems. In the project the development of different hybrid systems techniques are presented. They have covered the use of neural network structure in Fuzzy system (implementation) functioning.

The obtained results show that Neuro fuzzy system is able to create an appropriate set of rules for difficult processes. They provide stability of the constructing system. This allows desirable and adaptively control complicated processes.

REFERENCES

[1] Rahib Abiyev, Softcomputing elements based controllers. Electrical, Electronics and Computer Engineering Symposium NEU-CEE2001 & Exhibition, Nicosia, TRNC, Turkey, 2001.

[2] M. Aiken, "Artificial Neural Systems as a Research Paradigm for the Study of Group Decision Support Systems," Group Decision and Negotiation, in press.

[3] Kelly Fish, James Barnes, M. Aiken,"Artificial Neural Networks: A New Methodology for Industrial Market Segmentation," Industrial Marketing Management, Vol. 24, No. 5, 1995

[4] M. Aiken, J. Morrison, J. Paolillo, and L. Motiwalla,"Forecasting Gross Domestic Product Using a Neural Network," Decision Sciences Conference, November 1995.

[5] Wong, F. & Tan C., "Hybrid Neural, Genetic and Fuzzy Systems," in Trading on the Edge (Deboeck G. Ed.), John Wiley & Sons, Inc., 1994.

[6] Peters E., Fractal Structure in the Capital Markets, Financial Analyst Journal, May/June 93 Issue.

[7] Peters E., Fractal Market Analysis, John Wiley & Sons, Inc., 1994. Holland, J. (1975). Adaptation in natural and artificial systems.

[8] Tan P., "Automatic selection of neural network architectures via genetic algorithm", MSc thesis in preparation.

[9] Tan P., Lim G., Chua K., Wong F, Neo S."A Comparative Study Among Neural Networks, Radial Basis Functions and Regression Models", Second International Conference on Automation, Robotics and Computer Vision (ICARCV'92), Sept., 1992, Singapore.

[10] Wong, F., "NeuroFuzzy Computing Technology," Guest Editorial, NeuroVe$t Journal, May-Jun. 1994, pp8-10.

[11] Wong, F., Wang, P., Goh T., Quek B.K., "A Fuzzy Neural System for Stock Selection," Financial Analyst Journal, published by Association for Investment Management and Research, Jan Feb,. 1992.

[12] Wong F., "Time Series Forecasting Using BackPropagation Neural Networks," Neurocomputing, 2 (1990/91) 147 159, Elsevier.

[13] Wong F., "FastProp: A Selective Training Algorithm For Fast Error Propagation," Proceedings of the IJCNN, 1991, Singapore, pp 2038 2044.

[14] Wong F., Wang P., Goh T., "Fuzzy Neural Systems For Decision Making," Proceedings of the International Joint Conference on Neural Networks (IJCNN'91), Nov 1991, Singapore.

[15] Wong F., Tan P., Zhang X., "Neural Networks, Genetic Algorithms and Fuzzy Logic for Forecasting," Proceedings of the Third International Conference on Advanced Trading Technologies AI Applications on Wall Street & Worldwide, New York, Marriott Financial Center, July 1992.

[16] Wong F., "An Integrated Neural Network For Financial Time Series Analysis", Proceedings of the 24th Hawaii International Conference on System Sciences, Jan. 1991.

[17] Wong F., "NeuroForecaster -- A Neural Network For Time Series Predictive Analysis," Technical Report, National University of Singapore, May 1990.

[18] Wong F., Lee D., "A Hybrid Neural Network For Stock Selection," Proceedings of The Second Annual International Conference on ARTIFICIAL INTELLIGENCE APPLICATIONS ON WALL STREET, April 19-22, 1993, New York, NY USA.

[19] Wong F., "Hybrid Systems of Neural Network, Fuzzy Logic and Genetic Algorithms", in Advanced Technology for Trading, Portfolio and Risk Management, Edited by Dr. Guido Deboeck, Advanced Analytical Laboratory, Investment Dept., World Bank.

[20] Wong F., "Time Series Forecasting Using Backpropagation Neural Network", Neurocomputing, Elsevier,1990, 147-159

[21] Wong F. and Wang P., "A stock selection strategy using fuzzy neural networks", Neurocomputing 2 Elsevier (1990/91) 233-242

[22] Wong F. and Wang P.Z., "A Fuzzy Neural Network Approach For Forex Investment," International Fuzzy Engineering Symposium '91, Nov. 1991, Japan.

[23] Wong, F., "NeuroGenetic Computing ," NeuroVe$t Journal, July-August 1994.

[24] Lapedes, A., R. Farber (1987). "How neural nets work" Advances in Neural Information Systems.

[25] Parker D., "Learning Logic", Report TR 47, MIT Center for Computational Research in Economics and Management Science.