

**DEPARTMENT OF ELECTRICAL ENGINEERING &
ELECTRICAL & ELECTRONICS ENGINEERING
VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY, BURLA, ODISHA, INDIA**

8th SEMESTER EE & EEE

LECTURE NOTES ON SOFT COMPUTING

SUBJECT CODE: BCS 1705

SOFT COMPUTING (3-1-0)

MODULE-I (10 HOURS)

Introduction to Neuro, Fuzzy and Soft Computing, Fuzzy Sets : Basic Definition and Terminology, Set-theoretic Operations, Member Function Formulation and Parameterization, Fuzzy Rules and Fuzzy Reasoning, Extension Principle and Fuzzy Relations, Fuzzy If-Then Rules, Fuzzy Reasoning , Fuzzy Inference Systems, Mamdani Fuzzy Models, Sugeno Fuzzy Models, Tsukamoto Fuzzy Models, Input Space Partitioning and Fuzzy Modeling.

MODULE-II (10 HOURS)

Neural networks: Single layer networks, Perceptrons: Adaline, Mutilayer Perceptrons Supervised Learning, Back-propagation, LM Method, Radial Basis Function Networks, Unsupervised Learning Neural Networks, Competitive Learning Networks, Kohonen Self-Organizing Networks, Learning Vector Quantization, Hebbian Learning. Recurrent neural networks,. Adaptive neuro-fuzzy information; systems (ANFIS), Hybrid Learning Algorithm, Applications to control and pattern recognition.

MODULE-III (10 HOURS)

Derivative-free Optimization Genetic algorithms: Basic concepts, encoding, fitness function, reproduction. Differences of GA and traditional optimization methods. Basic genetic programming concepts Applications.,

MODULE-IV (10 HOURS)

Evolutionary Computing, Simulated Annealing, Random Search, Downhill Simplex Search, Swarm optimization

BOOKS

- [1]. J.S.R.Jang, C.T.Sun and E.Mizutani, "*Neuro-Fuzzy and Soft Computing*", PHI, 2004, Pearson Education 2004.
- [2]. Timothy J.Ross, "*Fuzzy Logic with Engineering Applications*", McGraw-Hill, International Editions, Electrical Engineering Series, Singapore, 1997.
- [3]. Davis E.Goldberg, "*Genetic Algorithms: Search, Optimization and Machine Learning*", Addison Wesley, N.Y., 1989.
- [4]. R.Eberhart, P.Simpson and R.Dobbins, "*Computational Intelligence - PC Tools*", AP Professional, Boston, 1996.
- [5]. Stamatios V. Kartalopoulos "Understanding Neural Networks and Fuzzy Logic Basic concepts & Applications", IEEE Press, PHI, New Delhi, 2004.

- [6]. Vojislav Kecman, "Learning & Soft Computing Support Vector Machines, Neural Networks, and Fuzzy Logic Models", Pearson Education, New Delhi, 2006.
- [7] S. Rajasekaran & GA Vijayalakshmi Pai "Neural Networks, Fuzzy Logic, and Genetic Algorithms synthesis and application", PHI

MODULE-I (10 HOURS)

Introduction to Neuro, Fuzzy and Soft Computing, Fuzzy Sets : Basic Definition and Terminology, Set-theoretic Operations, Member Function Formulation and Parameterization, Fuzzy Rules and Fuzzy Reasoning, Extension Principle and Fuzzy Relations, Fuzzy If-Then Rules, Fuzzy Reasoning , Fuzzy Inference Systems, Mamdani Fuzzy Models, Sugeno Fuzzy Models, Tsukamoto Fuzzy Models, Input Space Partitioning and Fuzzy Modeling.

LECTURE-1

INTRODUCTION:

What is intelligence?

Real intelligence is what determines the normal thought process of a human.

Artificial intelligence is a property of machines which gives it ability to mimic the human thought process. The intelligent machines are developed based on the intelligence of a subject, of a designer, of a person, of a human being. Now two questions: can we construct a control system that hypothesizes its own control law? We encounter a plant and looking at the plant behavior, sometimes, we have to switch from one control system to another control system where the plant is operating. The plant is may be operating in a linear zone or non-linear zone; probably an operator can take a very nice intelligent decision about it, but can a machine do it? Can a machine actually hypothesize a control law, looking at the model? Can we design a method that can estimate any signal embedded in a noise without assuming any signal or noise behavior?

That is the first part; before we model a system, we need to observe. That is we collect certain data from the system and How do we actually do this? At the lowest level, we have to sense the environment, like if I want to do temperature control I must have temperature sensor. This data is polluted or corrupted by noise. How do we separate the actual data from the corrupted data? This is the second question. The first question is that can a control system be able to hypothesize its own control law? These are very important questions that we should think of actually. Similarly, also to represent knowledge in a world model, the way we manipulate the objects in this world and the advanced is a very high level of intelligence that we still do not understand; the capacity to perceive and understand.

What is AI ?

Artificial Intelligence is concerned with the design of intelligence in an artificial device.

The term was coined by McCarthy in 1956.

There are two ideas in the definition.

1. Intelligence
2. artificial device

What is intelligence?

-Is it that which characterize humans? Or is there an absolute standard of judgement?

-Accordingly there are two possibilities:

- A system with intelligence is expected to behave as intelligently as a human
- A system with intelligence is expected to behave in the best possible manner

- Secondly what type of behavior are we talking about?

- Are we looking at the thought process or reasoning ability of the system?
- Or are we only interested in the final manifestations of the system in terms of its actions?

Given this scenario different interpretations have been used by different researchers as defining the scope and view of Artificial Intelligence.

1. One view is that artificial intelligence is about designing systems that are as intelligent as humans. This view involves trying to understand human thought and an effort to build machines that emulate the human thought process. This view is the cognitive science approach to AI.
2. The second approach is best embodied by the concept of the Turing Test. Turing held that in future computers can be programmed to acquire abilities rivaling human intelligence. As part of his argument Turing put forward the idea of an 'imitation game', in which a human being and a computer would be interrogated under conditions where the interrogator would not know which was which, the communication being entirely by textual messages. Turing argued that if the interrogator could not distinguish them by questioning, then it would be unreasonable not to call the computer intelligent. Turing's 'imitation game' is now usually called 'the Turing test' for intelligence.
3. Logic and laws of thought deals with studies of ideal or rational thought process and inference. The emphasis in this case is on the inferencing mechanism, and its properties. That is how the system arrives at a conclusion, or the reasoning behind its selection of actions is very important in this point of view. The soundness and completeness of the inference mechanisms are important here.
4. The fourth view of AI is that it is the study of rational agents. This view deals with building machines that act rationally. The focus is on how the system acts and performs, and not so much on the reasoning process. A rational agent is one that acts rationally, that is, in the best possible manner.

Typical AI problems

While studying the typical range of tasks that we might expect an “intelligent entity” to perform, we need to consider both “common-place” tasks as well as expert tasks.

Examples of common-place tasks include

- *Recognizing* people, objects.
- *Communicating* (through *natural language*).
- *Navigating* around obstacles on the streets

These tasks are done matter of factly and routinely by people and some other animals.

Expert tasks include:

- Medical diagnosis.
- Mathematical problem solving
- Playing games like chess

These tasks cannot be done by all people, and can only be performed by skilled specialists.

Now, which of these tasks are easy and which ones are hard? Clearly tasks of the first type are easy for humans to perform, and almost all are able to master them. However, when we look at what computer systems have been able to achieve to date, we see that their achievements include performing sophisticated tasks like medical diagnosis, performing symbolic integration, proving theorems and playing chess.

On the other hand it has proved to be very hard to make computer systems perform many routine tasks that all humans and a lot of animals can do. Examples of such tasks include navigating our way without running into things, catching prey and avoiding predators. Humans and animals are also capable of interpreting complex sensory information. We are able to recognize objects and people from the visual image that we receive. We are also able to perform complex social functions.

Intelligent behaviour

This discussion brings us back to the question of what constitutes intelligent behaviour. Some of these tasks and applications are:

1. Perception involving image recognition and computer vision
2. Reasoning
3. Learning
4. Understanding language involving natural language processing, speech processing
5. Solving problems
6. Robotics

Practical applications of AI

AI components are embedded in numerous devices e.g. in copy machines for automatic correction of operation for copy quality improvement. AI systems are in everyday use for identifying credit card fraud, for advising doctors, for recognizing speech and in helping complex planning tasks. Then there are intelligent tutoring systems that provide students with personalized attention.

Thus AI has increased understanding of the nature of intelligence and found many applications. It has helped in the understanding of human reasoning, and of the nature of intelligence. It has also helped us understand the complexity of modeling human reasoning.

Approaches to AI

Strong AI aims to build machines that can truly reason and solve problems. These machines should be self aware and their overall intellectual ability needs to be indistinguishable from that of a human being. Excessive optimism in the 1950s and 1960s concerning strong AI has given way to an appreciation of the extreme difficulty of the problem. Strong AI maintains that suitably programmed machines are capable of cognitive mental states.

Weak AI: deals with the creation of some form of computer-based artificial intelligence that cannot truly reason and solve problems, but can act as if it were intelligent. Weak AI holds that suitably programmed machines can simulate human cognition.

Applied AI: aims to produce commercially viable "smart" systems such as, for example, a security system that is able to recognise the faces of people who are permitted to enter a particular building. Applied AI has already enjoyed considerable success.

Cognitive AI: computers are used to test theories about how the human mind works--for example, theories about how we recognise faces and other objects, or about how we solve abstract problems.

Limits of AI Today

Today's successful AI systems operate in well-defined domains and employ narrow, specialized knowledge. Common sense knowledge is needed to function in complex, open-ended worlds. Such a system also needs to understand unconstrained natural language. However these capabilities are not yet fully present in today's intelligent systems.

What can AI systems do

Today's AI systems have been able to achieve limited success in some of these tasks.

- In Computer vision, the systems are capable of face recognition
- In Robotics, we have been able to make vehicles that are mostly autonomous.

- In Natural language processing, we have systems that are capable of simple machine translation.
- Today's Expert systems can carry out medical diagnosis in a narrow domain
- Speech understanding systems are capable of recognizing several thousand words continuous speech
- Planning and scheduling systems had been employed in scheduling experiments with the Hubble Telescope.
- The Learning systems are capable of doing text categorization into about a 1000 topics
- In Games, AI systems can play at the Grand Master level in chess (world champion), checkers, etc.

What can AI systems NOT do yet?

- Understand natural language robustly (e.g., read and understand articles in a newspaper)
- Surf the web
- Interpret an arbitrary visual scene
- Learn a natural language
- Construct plans in dynamic real-time domains
- Exhibit true autonomy and intelligence

Applications:

We will now look at a few famous AI system that has been developed over the years.

1. ALVINN:

Autonomous Land Vehicle In a Neural Network

In 1989, Dean Pomerleau at CMU created ALVINN. This is a system which learns to control vehicles by watching a person drive. It contains a neural network whose input is a 30x32 unit two dimensional camera image. The output layer is a representation of the direction the vehicle should travel.

The system drove a car from the East Coast of USA to the west coast, a total of about 2850 miles. Out of this about 50 miles were driven by a human, and the rest solely by the system.

2. Deep Blue

In 1997, the Deep Blue chess program created by IBM, beat the current world chess champion, Gary Kasparov.

3. Machine translation

A system capable of translations between people speaking different languages will be a remarkable achievement of enormous economic and cultural benefit. Machine translation is one of the important fields of endeavour in AI. While some translating systems have been developed, there is a lot of scope for improvement in translation quality.

4. Autonomous agents

In space exploration, robotic space probes autonomously monitor their surroundings, make decisions and act to achieve their goals.

NASA's Mars rovers successfully completed their primary three-month missions in April, 2004. The Spirit rover had been exploring a range of Martian hills that took two months to reach. It is finding curiously eroded rocks that may be new pieces to the puzzle of the region's past. Spirit's twin, Opportunity, had been examining exposed rock layers inside a crater.

5. Internet agents

The explosive growth of the internet has also led to growing interest in internet agents to monitor users' tasks, seek needed information, and to learn which information is most useful

What is soft computing?

An approach to computing which parallels the remarkable ability of the human mind to reason and learn in an environment of uncertainty and imprecision.

It is characterized by the use of inexact solutions to computationally hard tasks such as the solution of nonparametric complex problems for which an exact solution can't be derived in polynomial of time.

Why soft computing approach?

Mathematical model & analysis can be done for relatively simple systems. More complex systems arising in biology, medicine and management systems remain intractable to conventional mathematical and analytical methods. Soft computing deals with imprecision, uncertainty, partial truth and approximation to achieve tractability, robustness and low solution cost. It extends its application to various disciplines of Engg. and science. Typically human can:

1. Take decisions
2. Inference from previous situations experienced
3. Expertise in an area
4. Adapt to changing environment
5. Learn to do better
6. Social behaviour of collective intelligence

Intelligent control strategies have emerged from the above mentioned characteristics of human/ animals. The first two characteristics have given rise to Fuzzy logic; 2nd, 3rd and 4th have led to Neural Networks; 4th, 5th and 6th have been used in evolutionary algorithms.

Characteristics of Neuro-Fuzzy & Soft Computing:

1. Human Expertise
2. Biologically inspired computing models
3. New Optimization Techniques
4. Numerical Computation
5. New Application domains
6. Model-free learning
7. Intensive computation
8. Fault tolerance
9. Goal driven characteristics
10. Real world applications

Intelligent Control Strategies (Components of Soft Computing): The popular soft computing components in designing intelligent control theory are:

1. Fuzzy Logic
2. Neural Networks
3. Evolutionary Algorithms

Fuzzy logic:

Most of the time, people are fascinated about fuzzy logic controller. At some point of time in Japan, the scientists designed fuzzy logic controller even for household appliances like a room heater or a washing machine. Its popularity is such that it has been applied to various engineering products.

Fuzzy number or fuzzy variable:

We are discussing the concept of a fuzzy number. Let us take three statements: zero, almost zero, near zero. Zero is exactly zero with truth value assigned 1. If it is almost 0, then I can

think that between minus 1 to 1, the values around 0 is 0, because this is almost 0. I am not very precise, but that is the way I use my day to day language in interpreting the real world. When I say near 0, maybe the bandwidth of the membership which represents actually the truth value. You can see that it is more, bandwidth increases near 0. This is the concept of fuzzy number. Without talking about membership now, but a notion is that I allow some small bandwidth when I say almost 0. When I say near 0 my bandwidth still further increases. In the case minus 2 to 2, when I encounter any data between minus 2 to 2, still I will consider them to be near 0. As I go away from 0 towards minus 2, the confidence level how near they are to 0 reduces; like if it is very near to 0, I am very certain. As I progressively go away from 0, the level of confidence also goes down, but still there is a tolerance limit. So when zero I am precise, I become imprecise when almost and I further become more imprecise in the third case.

When we say fuzzy logic, that is the variables that we encounter in physical devices, fuzzy numbers are used to describe these variables and using this methodology when a controller is designed, it is a fuzzy logic controller.

Neural networks :

Neural networks are basically inspired by various way of observing the biological organism. Most of the time, it is motivated from human way of learning. It is a learning theory. This is an artificial network that learns from example and because it is distributed in nature, fault tolerant, parallel processing of data and distributed structure.

The basic elements of artificial Neural Network are: input nodes, weights, activation function and output node. Inputs are associated with synaptic weights. They are all summed and passed through an activation function giving output y. In a way, output is summation of the signal multiplied with synaptic weight over many input channels.

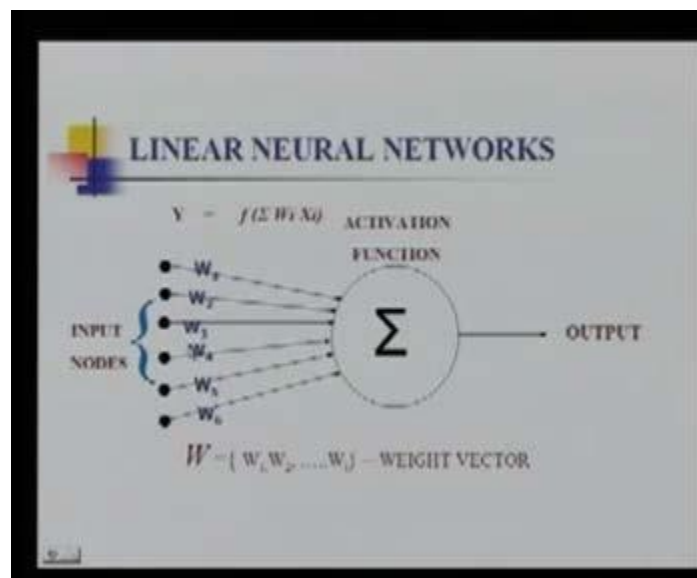


Fig. Basic elements of an artificial neuron

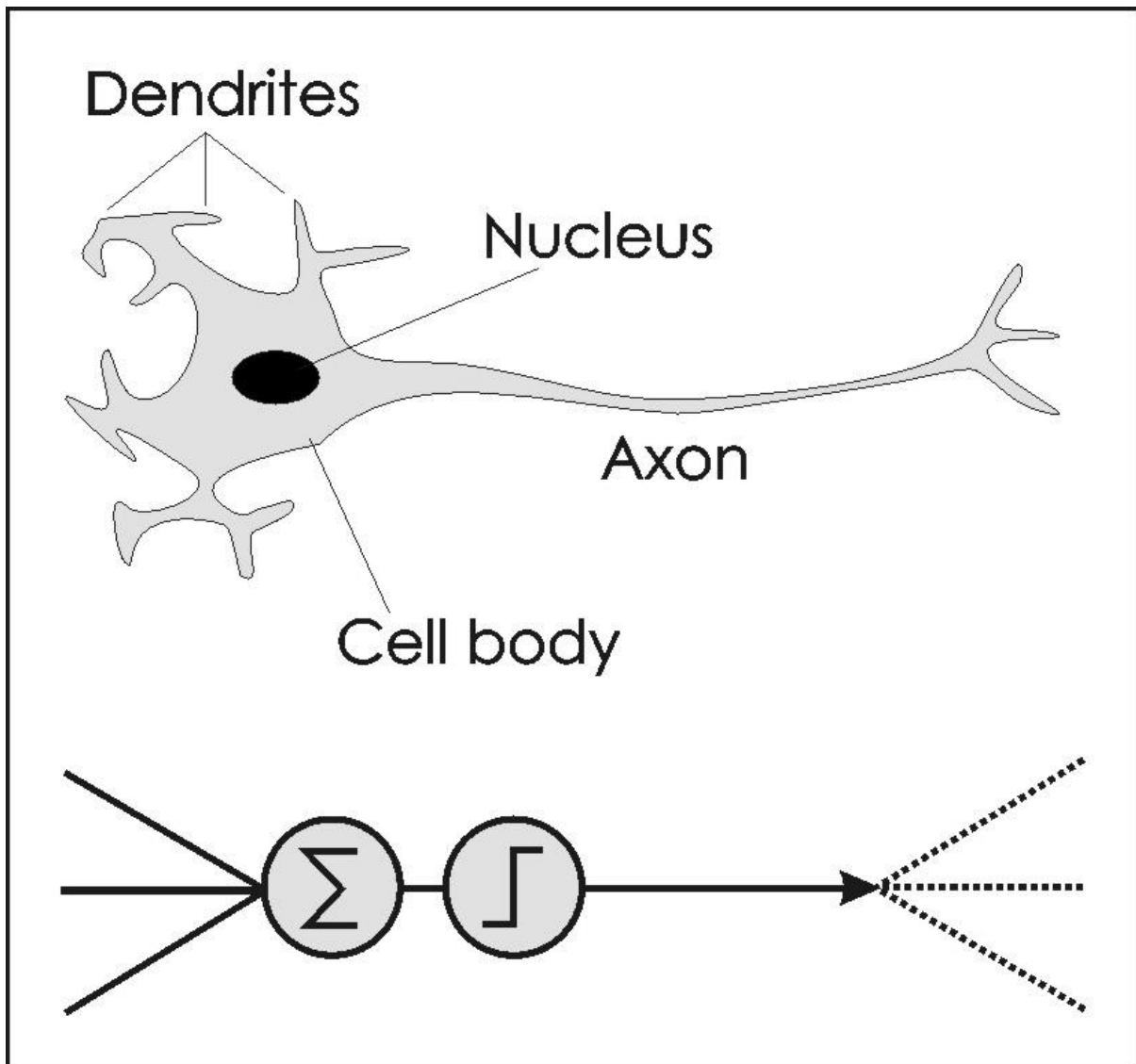


Fig. Analogy of biological neuron and artificial neuron

Above fig. Shows a biological neuron on top. Through axon this neuron actuates the signal and this signal is sent out through synapses to various neurons. Similarly shown a classical artificial neuron(bottom).This is a computational unit. There are many inputs reaching this. The input excites this neuron. Similarly, there are many inputs that excite this computational unit and the output again excites many other units like here. Like that taking certain concepts in actual neural network, we develop these artificial computing models having similar structure.

There are various locations where various functions take place in the brain.

If we look at a computer and a brain, this is the central processing unit and a brain. Let us compare the connection between our high speed computers that are available in the market today and a brain. Approximately there are 10 to the power of 14 synapses in the human brain, whereas typically you will have 10 to the power of 8 transistors inside a CPU. The element size is almost comparable, both are 10 to the power minus 6 and energy use is almost like 30 Watts and comparable actually; that is energy dissipated in a brain is almost same as in a computer. But you see the processing speed. Processing speed is only 100 hertz; our brain is very slow, whereas computers nowadays, are some Giga hertz.

When you compare this, you get an idea that although computer is very fast, it is very slow to do intelligent tasks like pattern recognition, language understanding, etc. These are certain activities which humans do much better, but with such a slow speed, 100 Hz. contrast between these two, one of the very big difference between these two is the structure; one is brain, another is central processing unit is that the brain learns, we learn. Certain mapping that is found in biological brain that we have studied in neuroscience is not there in a central processing unit and we do not know whether self awareness takes place in the brain or somewhere else, but we know that in a computer there is no self-awareness.

Neural networks are analogous to adaptive control concepts that we have in control theory and one of the most important aspects of intelligent control is to learn the control parameters, to learn the system model. Some of the learning methodologies we will be learning here is the error-back propagation algorithm, real-time learning algorithm for recurrent network, Kohonen's self organizing feature map & Hopfield network.

Features of Artificial Neural Network (ANN) models:

1. Parallel Distributed information processing
2. High degree of connectivity between basic units
3. Connections are modifiable based on experience
4. Learning is a continuous unsupervised process
5. Learns based on local information
6. Performance degrades with less units

All the methods discussed so far makes a strong assumption about the space around; that is, when we use whether a neural network or fuzzy logic or and any method that may have been adopted in intelligent control framework, they all make always very strong assumptions and normally they cannot work in a generalized condition. The question is that can they hypothesize a theory? When I design all these controllers, I always take the data; the engineer takes the data. He always builds these models that are updated. They update their own weights based on the feedback from the plant. But the structure of the controller, the model by which we assume the physical plant, all these are done by the engineer and also the structure of the intelligent controller is also decided by the engineer. We do not have a machine that can hypothesize everything; the model it should select, the controller it should select, looking at simply data. As it encounters a specific kind of data from a plant can it come up with specific controller architecture and can it come up with specific type of system model? That is the question we are asking now.

You will see that in the entire course we will be discussing various tools. They will only be dealing with these two things; behaviour. These tools are actually developed by mimicking the human behavior, but not the human way of working. An intelligent machine is one which learns, thinks and behaves in line with the thought process. That we would like but we are very far from it. At least, at the moment, we are very far from this target of achieving real intelligence.

We perceive the environment in a very unique way, in a coherent manner. This is called unity of perception and intelligence has also something to do with this unity of perception, awareness and certain things are not very clear to us until now. So an intelligent machine is one which learns, thinks & behaves in line with thought process.

Evolutionary algorithms:

These are mostly derivative free optimization algorithms that perform random search in a systematic manner to optimize the solution to a hard problem. In this course Genetic Algorithm being the first such algorithm developed in 1970's will be discussed in detail. The other algorithms are swarm based that mimic behaviour of organisms, or any systematic process.

LECTURE-2

Fuzzy Sets Basic Concepts

- Characteristic Function (Membership Function)
- Notation
- Semantics and Interpretations
- Related crisp sets
- Support, Bandwidth, Core, α -level cut
- Features, Properties, and More Definitions
- Convexity, Normality
- Cardinality, Measure of Fuzziness
- MF parametric formulation
- Fuzzy Set-theoretic Operations
- Intersection, Union, Complementation
- T-norms and T-conorms
- Numerical Examples
- Fuzzy Rules and Fuzzy Reasoning
- Extension Principle and Fuzzy Relations
- Fuzzy If-Then Rules
- Fuzzy Reasoning
- Fuzzy Inference Systems
- Mamdani Fuzzy Models
- Sugeno Fuzzy Models
- Tsukamoto Fuzzy Models
- Input Space Partitioning
- Fuzzy Modeling.

The father of fuzzy logic is Lotfi Zadeh who is still there, proposed in 1965. Fuzzy logic can manipulate those kinds of data which are imprecise.

Basic definitions & terminology:

Fuzzy Number:

A fuzzy number is fuzzy subset of the universe of a numerical number that satisfies condition of normality & convexity. It is the basic type of fuzzy set.

Why fuzzy is used? Why we will be learning about fuzzy? The word fuzzy means that, in general sense when we talk about the real world, our expression of the real world, the way we quantify the real world, the way we describe the real world, are not very precise.

When I ask what your height is, nobody would say or nobody would expect you to know a precise answer. If I ask a precise question, probably, you will give me your height as 5 feet 8 inches. But normally, when I see people, I would say this person is tall according to my own estimate, my own belief and my own experience; or if I ask, what the temperature is today, the normal answer people would give is, today it is very hot or hot or cool. Our expression about the world around us is always not precise. Not to be precise is exactly what is fuzzy.

Fuzzy logic is logic which is not very precise. Since we deal with our world with this imprecise way, naturally, the computation that involves the logic of impreciseness is much

more powerful than the computation that is being carried through a precise manner, or rather precision logic based computation is inferior; not always, but in many applications, they are very inferior in terms of technological application in our day to day benefits, the normal way.

Fuzzy logic has become very popular; in particular, the Japanese sold the fuzzy logic controller, fuzzy logic chips in all kinds of house hold appliances in early 90's. Whether it is washing machine or the automated ticket machine, anything that you have, the usual house hold appliances, the Japanese actually made use of the fuzzy logic and hence its popularity grew.

Fuzzy Sets

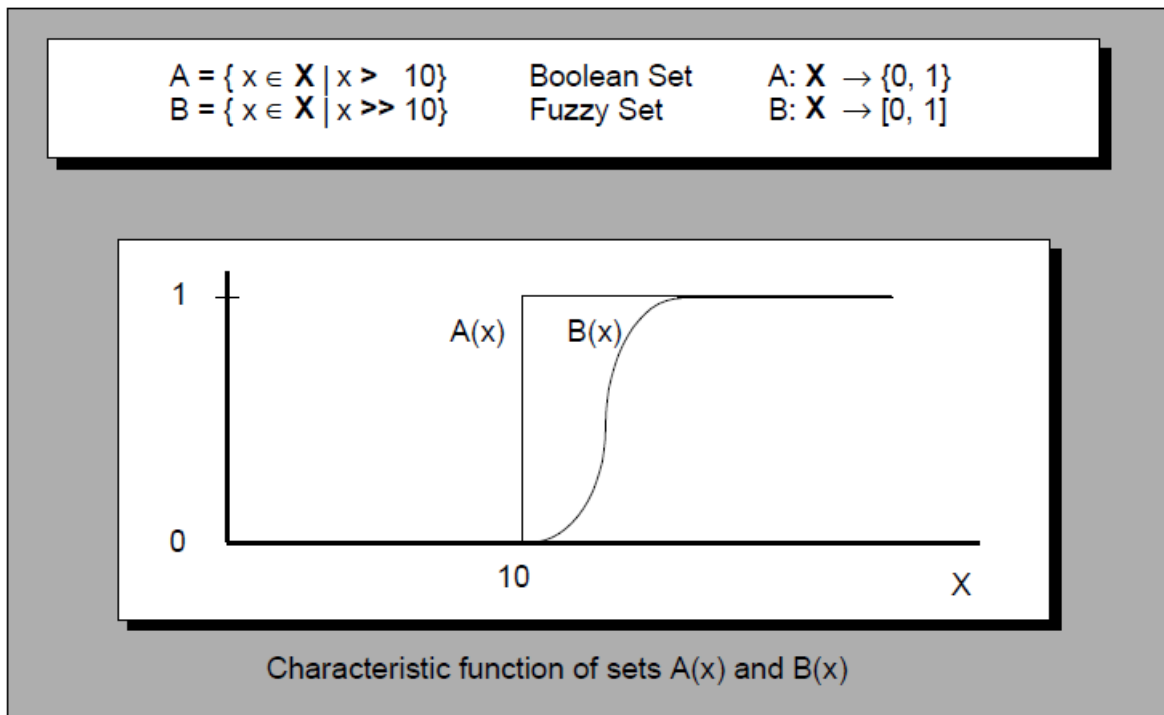


Fig. Difference in Fuzzy and crisp boundary

As fuzzy means from precision to imprecision. Here, when I say 10, I have an arrow at 10, pointing that I am exactly meaning 10 means 10.00000 very precise. When I say they are all almost 10, I do not mean only 10, rather in the peripheral 10. I can tolerate a band from minus 9 to 9, whereas if I go towards 9 or 11, I am going away from 10, the notion of 10. That is what is almost 10, that is around 10, but in a small bandwidth, I still allow certain bandwidth for 10.

This concept to be imprecise is fuzzy or to deal with the day to day data that we collect or we encounter and representing them in an imprecise manner like here almost 0, near 0, or hot, cold, or tall; if I am referring to height, tall, short medium. This kind of terminology that we normally talk or exchange among ourselves in our communication actually deals with imprecise data rather than precise data. Naturally, since our communications are imprecise, the computation resulting out of such communication language, the language which is imprecise must be associated with some logic.

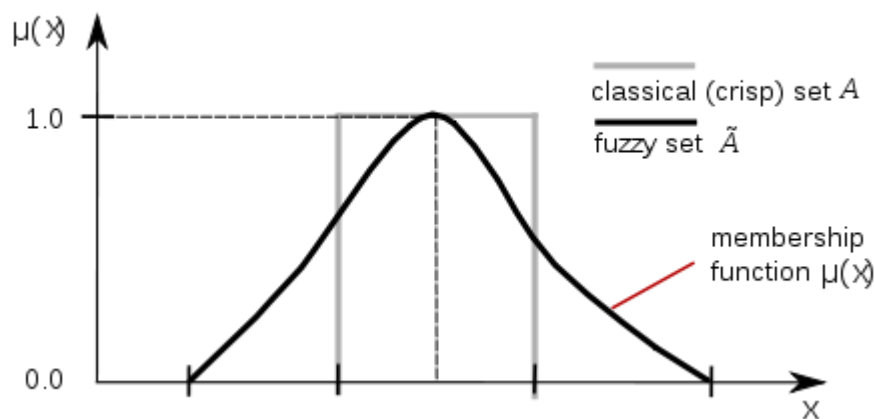


Fig. Sets: classical & fuzzy boundary

Set: A collection of objects having one or more common characteristics. For example, set of natural number, set of real numbers, members, or elements. Objects belonging to a set is represented as x belonging to A , where A is a set.

Universe of Discourse:

Defined as “a collection of objects all having the same characteristics”.

Notation: U or X , and elements in the universe of discourse are: u or x

Now, we will be talking about fuzzy sets. When I talked about classical set, we had classical set of the numbers that we know, like we talked about the set of natural numbers, set of real numbers. What is the difference between a fuzzy set and a classical set or a crisp set? The difference is that the members, they belong to a set A or a specific set A or B or X or Y , whatever it is, we define them; but the degree of belonging to the set is imprecise. If I say, a universal set in natural numbers, all the natural numbers fall in this set. If I take a subset of this natural number, like in earlier case, we put 1 to 11 in one set. When I ask, whether 12 belongs to set A , the answer is no; 13 belongs to set A ? The answer is no; because, in my natural number set, only 1 to 11 are placed. This is called classical set and their belongingness here is one. They all belong to this set.

But in a fuzzy set, I can have all the numbers in this set, but with a membership grade associated with it. When I say membership grade is 0 that means, they do not belong to the set, whereas a membership grade between 0 to 1, says how much this particular object may belong to the set.

The nomenclature/ Notation of a fuzzy set - how do we represent a fuzzy set there? One way is that let the elements of X be x_1, x_2, \dots, x_n ; then the fuzzy set A is denoted by any of the following nomenclature.

Mainly 2 types:

1. Numeric
2. Functional

Mostly, we

will be using either this or the first one, where you see the ordered pair x

$1 \mu_A x_1$; x_1 is member of A and x_1 is associated with a fuzzy index and so forth, x_2 and its fuzzy index, x_n and its fuzzy membership. The same thing, I can also write x_1 upon $\mu_A x_1$.

That means x_1 is the member and this is the membership. The other way is here, in the third pattern the membership is put first and in the bottom the member x_1 with a membership, x_2 with membership and x_n with membership.

Every member x of a fuzzy set A is assigned a fuzzy index. This is the membership grade $\mu_A x$ in the interval of 0 to 1, which is often called as the grade of membership of x in A . In a classical set, this membership grade is either 0 or 1; it either belongs to set A or does not belong. But in a fuzzy set this answer is not precise, answer is, it is possible. It is belonging to set A with a fuzzy membership 0.9 and I say it belongs to A with a fuzzy membership 0.1; that is, when I say 0.9, more likely it belongs to set A . When I say 0.1, less likely it belongs to set A . Fuzzy sets are a set of ordered pairs given by A . The ordered pair is x , where x is a member of the set. Along with that, what is its membership grade and how likely the subject belongs to set A ? That is the level we put, where x is a universal set and μ_x is the grade of membership of the object x in A . As we said, this membership μ .

$A x$ lies between 0 to 1; so, more towards 1, we say more likely it belongs to A . Like if I say membership grade is 1, certainly it belongs to A .

For an example: a set of all tall people. Tall if I define, classically I would say above 6 is tall and below 6 is not tall; that is, 5.9, 5 feet 9 inches is not tall and 6.1, 6 feet 1 inch is tall. That looks very weird; it does not look nice to say that a person who is 6 feet 1 inch is tall and 5 feet 9 inches is not tall. This ambiguity that we have in terms of defining such a thing in classical set, the difficulty that we face can be easily resolved in fuzzy set. In fuzzy set, we can easily say both 6.1, 6 feet 1 inch as well as 5.9 inches as tall, but level this difference; they are tall, but with a membership grade associated with this. This is what fuzzy set is.

Membership function - a membership function $\mu_A x$ is characterized by μ_A that maps all the members in set x to a number between 0 to 1, where x is a real number describing an object or its attribute, X is the universe of discourse and A is a subset of X .

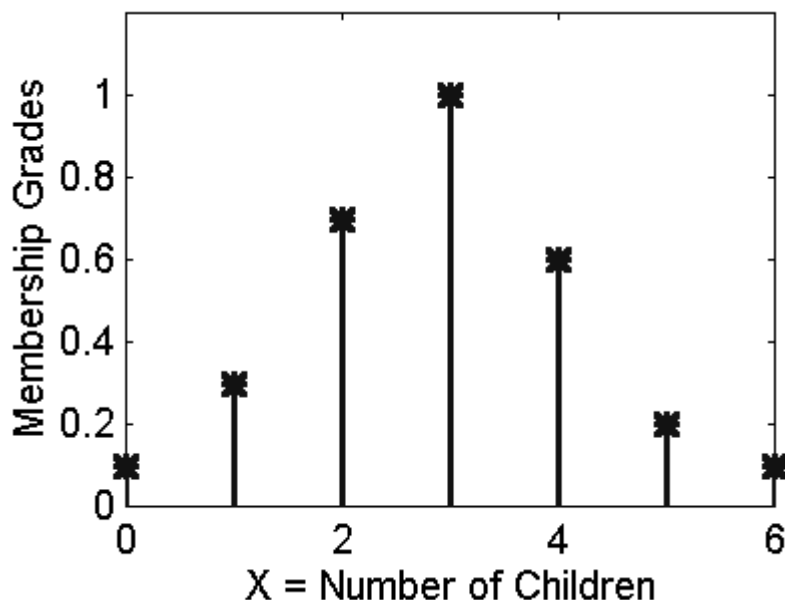


Fig. Fuzzy Sets with Discrete Universes

Fuzzy set A = "sensible number of children"

$X = \{0, 1, 2, 3, 4, 5, 6\}$ (discrete universe)

$A = \{(0, .1), (1, .3), (2, .7), (3, 1), (4, .6), (5, .2), (6, .1)\}$ --(See discrete ordered pairs)(1st expression)

or

$$A = \{(x, \mu_A(x)) \mid x \in X\},$$

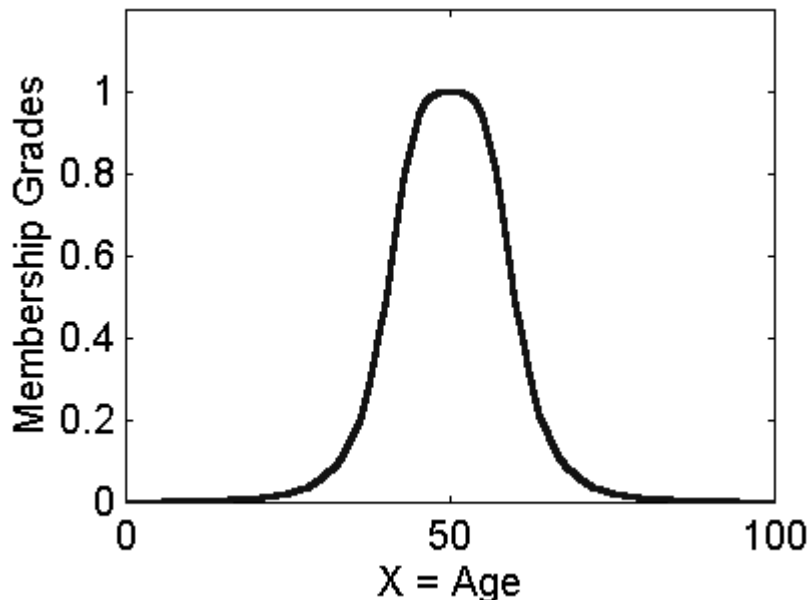


Fig. Fuzzy Set with Cont. Universe

Fuzzy set B = “about 50 years old”

X = Set of positive real numbers (continuous)

B = $\{(x, \mu_B(x)) \mid x \in X\}$

$\mu_B(x) = f(x)$

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

(2nd expression –with function that is subjective)

3rd expression of fuzzy set:

$$A = \begin{cases} \sum_{x_i \in X} \mu_A(x_i) / x_i, & \text{if } X \text{ is a collection of discrete objects.} \\ \int_X \mu_A(x) / x, & \text{if } X \text{ is a continuous space (usually the real line } R). \end{cases}$$

$$A = 0.1/0 + 0.3/1 + 0.7/2 + 1.0/3 + 0.7/4 + 0.3/5 + 0.1/6,$$

$$B = \int_{R^+} \frac{1}{1 + \left(\frac{x-50}{10}\right)^4} / x,$$

Linguistic variable and linguistic values:

Linguistic variable is a variable expressed in linguistic terms e.g. “Age” that assumes various linguistic values like :midleaged, young, old. The linguistic variables are characterized by membership functions.

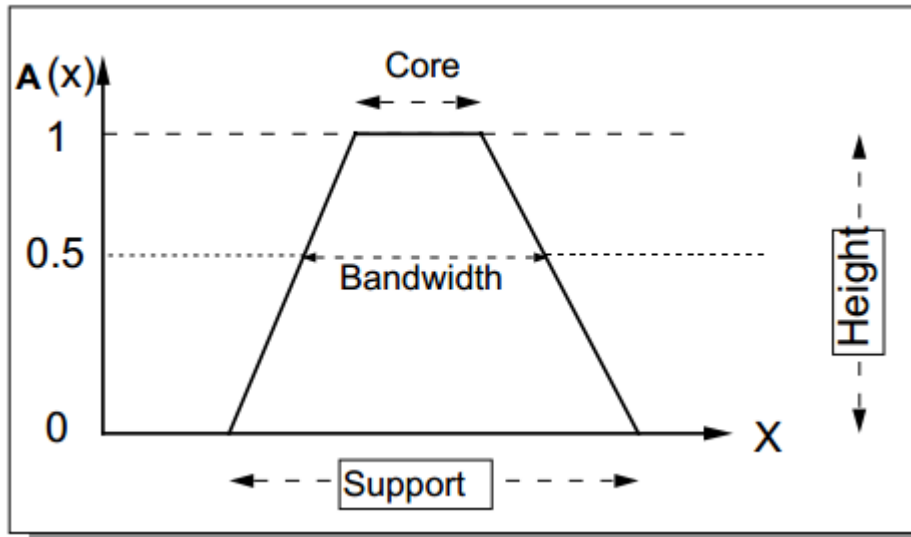


Fig. A membership function showing support, bandwidth, core, crossover points

Support:

Support of a fuzzy set A is the set of all points x in X such that $\mu_A(x) > 0$.

$$\text{Support}(A) = \{x \mid \mu_A(x) > 0\}$$

Core:

The core of a fuzzy set A is the set of all points x in X such that $\mu_A(x) = 1$

$$\text{core}(A) = \{x \mid \mu_A(x) = 1\}$$

Normality:

A fuzzy set A is normal if its core is nonempty. Always there is at least one x with $\mu_A(x) = 1$ then it is normal.

Crossover point:

A cross over point in fuzzy set A is the x with $\mu_A(x) = 0.5$

$$\text{crossover}(A) = \{x \mid \mu_A(x) = 0.5\}$$

Bandwidth:

For a normal & convex fuzzy set

$$\text{Width}(A) = |x_2 - x_1|, \text{ where } x_2 \text{ \& } x_1 \text{ are crossover points.}$$

fuzzy singleton:

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

The α -cut or α -level set of a fuzzy set A is a crisp set defined by

$$A_\alpha = \{x | \mu_A(x) \geq \alpha\}.$$

Strong α -cut or strong α -level set are defined similarly:

$$A'_\alpha = \{x | \mu_A(x) > \alpha\}.$$

For the set given in figure we can find equivalence & write

$$\text{support}(A) = A'_0,$$

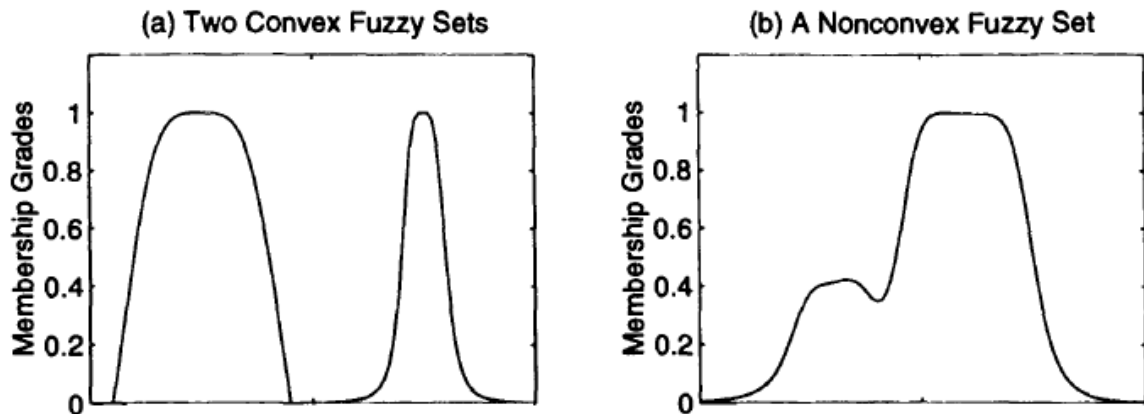
$$\text{core}(A) = A_1,$$

Convexity:

A fuzzy set A is **convex** if and only if for any $x_1, x_2 \in X$ and any $\lambda \in [0, 1]$,

$$\mu_A(\lambda x_1 + (1 - \lambda)x_2) \geq \min\{\mu_A(x_1), \mu_A(x_2)\}.$$

Alternatively, A is convex if all its α -level sets are convex.



Symmetry:

A fuzzy set is symmetric if its MF is symmetric about a certain point $x=c$ such that,

$$\mu_A(c+x) = \mu_A(c-x) \text{ for all } x \text{ in } X$$

Comparison of the classical approach and fuzzy approach:

Let us say, consider a universal set T which stands for temperature. Temperature I can say cold, normal and hot. Naturally, these are subsets of the universal set T ; the cold temperature, normal temperature and hot temperature they are all subsets of T .

The classical approach, probably, one way to define the classical set is cold. I define cold: temperature T ; temperature is a member of cold set which belongs to the universal set T such that this temperature, the member temperature is between 5 degree and 15 degree centigrade.

Similarly, the member temperature belongs to normal, if it is between 15 degree centigrade and 25 degree centigrade. Similarly, the member temperature belongs to hot set when the

temperature is between 25 degree centigrade and 35 degree centigrade. As I said earlier, one should notice that 14.9 degree centigrade is cold according to this definition while 15.1 degree centigrade is normal implying the classical sets have rigid boundaries and because of this rigidity, the expression of the world or the expression of data becomes very difficult. For me, I feel or any one of us will feel very uneasy to say that 14.9 degrees centigrade is cold and 15.1 degree centigrade is normal or for that matter, 24.9 degrees centigrade is normal and 25 degree or 25.1 degree centigrade is hot. That is a little weird or that is bizarre to have such an approach to categorize things into various sets.

In a fuzzy set, it is very easy to represent them here. If the temperature is around 10 degree centigrade, it is cold; temperature is around 20 degrees centigrade, it is normal and when temperature is around 30 degree centigrade it is hot. In that sense, they do not have a rigid boundary. If you say here, 25 degree centigrade, the 25 degree centigrade can be called simultaneously hot as well as normal, with a fuzzy membership grade 0.5. 25 degrees centigrade belongs to both normal as well as hot, but when I say 28 degree centigrade, this is more likely a temperature in the category of hot, whereas the 22 degree centigrade is a temperature that is more likely belonging to the set normal. This is a much nicer way to represent a set. This is how the imprecise data can be categorized in a much nicer way using fuzzy logic. This is the contrasting feature, why the fuzzy logic was introduced in the first place.

Fuzzy sets have soft boundaries. I can say cold from almost 0 degree centigrade to 20 degree centigrade. If 10 degree has a membership grade 1 and as I move away from 10 degree in both directions, I lose the membership grade. The membership grade reduces from 1 to 0 here, and in this direction also from 1 to 0. The temperature, As I go, my membership grade reduces; I enter into a different set simultaneously and that is normal. You can easily see, like temperature 12, 13, 14, 15 all belong to both categories cold as well as normal, but each member is associated with a membership grade; this is very important.

In a classical set, there are members in a set. Here, there are members in a set associated with a fuzzy index or membership function.

LECTURE-3

Parameterization of Membership Function:

Once we talk about each member in a fuzzy set associated with membership function, you must know how to characterize this membership function. The parameters are adjusted to fine tune a fuzzy inference system to achieve desired I/O mapping. The membership functions given here are one- dimensional. 2 dimensional MFs can be formed by cylindrical extension from these basic MFs.

$$\text{triangle}(x; a, b, c) = \begin{cases} 0, & x \leq a. \\ \frac{x-a}{b-a}, & a \leq x \leq b. \\ \frac{c-x}{c-b}, & b \leq x \leq c. \\ 0, & c \leq x. \end{cases}$$

Where $a < b < c$ & that are x coordinates of the corners of triangular MF

$$\text{trapezoid}(x; a, b, c, d) = \begin{cases} 0, & x \leq a. \\ \frac{x-a}{b-a}, & a \leq x \leq b. \\ 1, & b \leq x \leq c. \\ \frac{d-x}{d-c}, & c \leq x \leq d. \\ 0, & d \leq x. \end{cases}$$

Where $a < b < c < d$ & that are x coordinates of the corners of trapezoidal MF

$$\text{bell}(x; a, b, c) = \frac{1}{1 + \left| \frac{x-c}{a} \right|^{2b}},$$

Where c is the centre & a is adjusted to vary the width of MF, b controls slope at crossover points.

Bell membership function is also termed as Cauchy MF.

$$\text{gaussian}(x; c, \sigma) = e^{-\frac{1}{2} \left(\frac{x-c}{\sigma} \right)^2}.$$

Where c is the centre & σ is the width of MF.

Left-Right MF:

$$\text{LR}(x; c, \alpha, \beta) = \begin{cases} F_L \left(\frac{c-x}{\alpha} \right), & x \leq c. \\ F_R \left(\frac{x-c}{\beta} \right), & x \geq c, \end{cases}$$

Sigmoidal MF:

A sigmoidal MF is defined by

$$\text{sig}(x; a, c) = \frac{1}{1 + \exp[-a(x-c)]},$$

where a controls the slope at the crossover point $x = c$.

It can be open left or open right depending on sign of a.

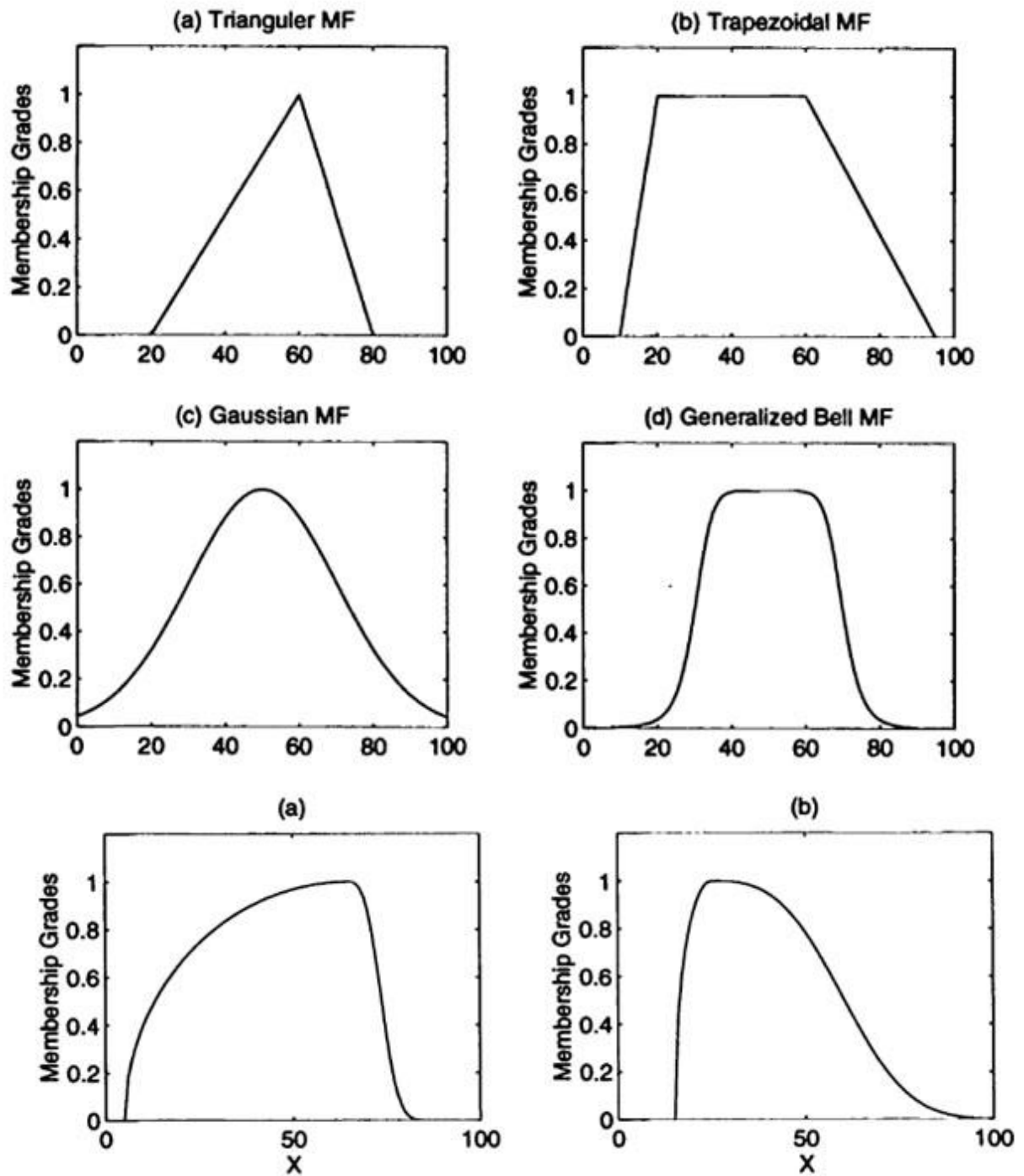


Fig. Membership functions a. Triangle b. Trapezoidal c. Gaussian d. Bell, e. Left f. Right

LECTURE-4

Fuzzy set operations:

The main features of operation on fuzzy set are that unlike conventional sets, operations on fuzzy sets are usually described with reference to membership function. When I say operation, I do not do with the member itself, but I manipulate. When I say operation, I manipulate the membership of the members in a set; members are not manipulated, rather the membership function of the member is manipulated. This is very important; that is, x and $\mu(x)$. In classical set what is manipulated is x .

If I say, x is 1 In classical set when I say x is 1 then, I would say 1 minus x is 0. In this, the manipulation concerns with the member; whereas any kind of manipulation in fuzzy set does not involve with x ; rather it involves μx .

Containment or subset:

Fuzzy set A is **contained** in fuzzy set B (or, equivalently, A is a **subset** of B , or A is smaller than or equal to B) if and only if $\mu_A(x) \leq \mu_B(x)$ for all x . In symbols,

$$A \subseteq B \iff \mu_A(x) \leq \mu_B(x)$$

Three common operations: intersection which we say is the minimum function, union, which we say is the maximum function and then fuzzy complementation

Standard fuzzy operations:

Intersection(Conjunction)or T-norm:

We can easily see that, the membership of A (green) intersection B (red) in fig. is all the members that belongs to, that is common between A and B . Their membership will follow these (blue) curves. There are two things we are doing. We have 2 sets. One is set A and the other is set B . Classically, what we see is the common members between A and B . We are not only seeing the common members, here we are also seeing, what is their membership function.

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

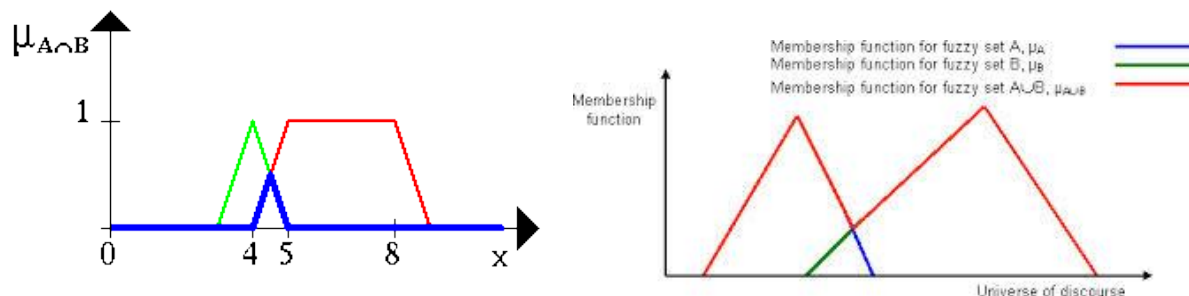


Fig. Fuzzy set operations intersection & union

The membership function is computed minimum; that is, μ_A intersection B is minimum of $\mu_A x$ and $\mu_B x$. That is the membership function. When there is a common member between A and B , the membership function wherever is minimum that is retained and the other one is thrown away. The member is retained; what is changing is the membership function.

Union(Disjunction) or T-co-norm or S-norm:

That is the meaning of these two curves that we have and then we are trying to find out what the fuzzy union is. I have to find out In this the members are both belonging to A and B. But their membership is maximum of both. if I have common members. I have set A and I have set B; A union B is my union set. If x belongs to A and x belongs to B, then x also belongs to A union B. But in fuzzy set, here this is $\mu_A(x)$ and here it is $\mu_B(x)$ and in this case, this is maximum of $\mu_A(x)$ and $\mu_B(x)$; the membership function. That is the way it is assigned.

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

This candidate, when it comes to A union B take these two values of membership, find the maximum which is 0.1 and assign here, which is 0.1. This is, $\mu_{A \cup B}$ is 0.1. This is the meaning. This is a very important operation that we do. When we have two different fuzzy sets, the operations are classical. The manipulation is among the membership functions; otherwise, the notion of the classical fuzzy operation also remains intact, except that the associated fuzzy membership gets changed.

Complement(Negation):

now it is fuzzy complementation. What is complement? This one, this particular triangular function is my set R(red); fuzzy set R. The complement is like this; just inverse (blue). What is 1 minus $\mu_A(x)$; meaning 1 minus $\mu_A(x)$.

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x).$$

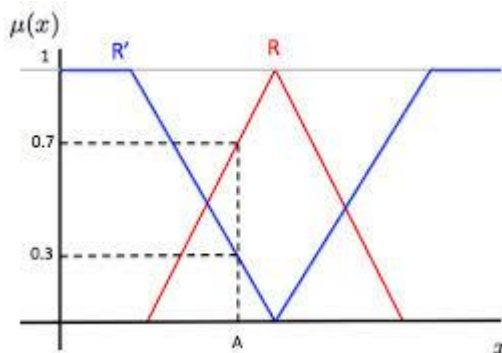


Fig. Complement of fuzzy set

What is seen that the members remain intact in the set A, whereas the associated membership functions got changed.

The other operations that we know for classical sets like De Morgan's law, the difference also can be used for the sets like De Morgan's law.

Properties/ identities of fuzzy sets:

They are commutative. A union B is B union A; A intersection B is B intersection A. It is like classical sets; fuzzy sets equally hold.

Associativity; A union B union C is A union B union C. Similarly, A union bracket B union C is A intersection B intersection C is A intersection B combined with intersection C.

Distributivity: you can easily see that $A \cup (B \cap C)$ is $(A \cup B) \cap (A \cup C)$ which is here. Similarly, here $A \cap (B \cup C)$. So, this is distributivity.

Idempotency which is $A \cup A$ is A and $A \cap A$ is A .

Identity: $A \cup \emptyset$ is A , $A \cap X$ is A , $\emptyset \cap A$ is \emptyset and $A \cup X$ is X ; here, X represents universal set.

The next step in establishing a complete system of fuzzy logic is to define the operations of EMPTY, EQUAL, COMPLEMENT (NOT), CONTAINMENT, UNION (OR), and INTERSECTION (AND). Before we can do this rigorously, we must state some formal definitions:

Definition 1: Let X be some set of objects, with elements noted as x . Thus, $X = \{x\}$.

Definition 2: A fuzzy set A in X is characterized by a membership function $\mu_A(x)$ which maps each point in X onto the real interval $[0.0, 1.0]$. As $\mu_A(x)$ approaches 1.0, the "grade of membership" of x in A increases.

Definition 3: A is EMPTY iff for all x , $\mu_A(x) = 0.0$.

Definition 4: $A = B$ iff for all x : $\mu_A(x) = \mu_B(x)$ [or, $\mu_A = \mu_B$].

Definition 5: $\mu_{A'} = 1 - \mu_A$.

Definition 6: A is CONTAINED in B iff $\mu_A \leq \mu_B$.

Definition 7: $C = A \cup B$, where: $\mu_C(x) = \text{MAX}(\mu_A(x), \mu_B(x))$.

Definition 8: $C = A \cap B$ where: $\mu_C(x) = \text{MIN}(\mu_A(x), \mu_B(x))$.

Difference probability & fuzzy operations:

It is important to note the last two operations, UNION (OR) and INTERSECTION (AND), which represent the clearest point of departure from a probabilistic theory for sets to fuzzy sets. Operationally, the differences are as follows:

For independent events, the probabilistic operation for AND is multiplication, which (it can be argued) is counterintuitive for fuzzy systems. For example, let us presume that $x = \text{Bob}$, S is the fuzzy set of smart people, and T is the fuzzy set of tall people. Then, if $\mu_S(x) = 0.90$ and $\mu_T(x) = 0.90$, the probabilistic result would be:

$$\mu_S(x) * \mu_T(x) = 0.81$$

whereas the fuzzy result would be:

$$\text{MIN}(\mu_S(x), \mu_T(x)) = 0.90$$

The probabilistic calculation yields a result that is lower than either of the two initial values, which when viewed as "the chance of knowing" makes good sense. However, in fuzzy terms the two membership functions would read something like "Bob is very smart" and "Bob is very tall." If we presume for the sake of argument that "very" is a stronger term than "quite," and that we would correlate "quite" with the value 0.81, then the semantic difference becomes obvious. The probabilistic calculation would yield the statement If Bob is very smart, and Bob is very tall, then Bob is a quite tall, smart person. The fuzzy calculation, however, would yield If Bob is very smart, and Bob is very tall, then Bob is a very tall, smart person.

Another problem arises as we incorporate more factors into our equations (such as the fuzzy set of heavy people, etc.). We find that the ultimate result of a series of AND's approaches 0.0, even if all factors are initially high. Fuzzy theorists argue that this is wrong: that five factors of the value 0.90 (let us say, "very") AND'ed together, should yield a value of 0.90 (again, "very"), not 0.59 (perhaps equivalent to "somewhat").

Similarly, the probabilistic version of A OR B is $(A+B - A*B)$, which approaches 1.0 as additional factors are considered. Fuzzy theorists argue that a string of low membership grades should not produce a high membership grade instead, the limit of the resulting membership grade should be the strongest membership value in the collection.

The skeptical observer will note that the assignment of values to linguistic meanings (such as 0.90 to "very") and vice versa, is a most imprecise operation. Fuzzy systems, it should be noted, lay no claim to establishing a formal procedure for assignments at this level; in fact, the only argument for a particular assignment is its intuitive strength. What fuzzy logic does propose is to establish a formal method of operating on these values, once the primitives have been established.

Hedges :

Another important feature of fuzzy systems is the ability to define "hedges," or modifier of fuzzy values. These operations are provided in an effort to maintain close ties to natural language, and to allow for the generation of fuzzy statements through mathematical calculations. As such, the initial definition of hedges and operations upon them will be quite a subjective process and may vary from one project to another. Nonetheless, the system ultimately derived operates with the same formality as classic logic. The simplest example is in which one transforms the statement "Jane is old" to "Jane is very old." The hedge "very" is usually defined as follows:

$$\mu_{\text{very}}A(x) = \mu A(x)^2$$

Thus, if $m_{\text{OLD}}(\text{Jane}) = 0.8$, then $m_{\text{VERYOLD}}(\text{Jane}) = 0.64$.

Other common hedges are "more or less" [typically $\text{SQRT}(\mu A(x))$], "somewhat," "rather," "sort of," and so on. Again, their definition is entirely subjective, but their operation is consistent: they serve to transform membership/truth values in a systematic manner according to standard mathematical functions.

$$\text{CON}(A) = A^2,$$

$$\text{DIL}(A) = A^{0.5}.$$

Cartesian Product & Co-product:

Let A & B be fuzzy sets in \underline{X} & \underline{Y} respectively, then Cartesian product of A & B is a fuzzy set in the product space $\underline{X} \times \underline{Y}$ with the membership function

$$\mu_{A \times B}(x, y) = \min(\mu_A(x), \mu_B(y)).$$

Similarly, Cartesian co-product $A+B$ is a fuzzy set

$$\mu_{A+B}(x, y) = \max(\mu_A(x), \mu_B(y)).$$

Both Product & Co-product are characterized by 2- dimensional MFs.

LECTURE-5

Fuzzy Extension Principle:

Consider a function $y = f(x)$.

If we know x it is possible to determine y .

Is it possible to extend this mapping when the input, x , is a fuzzy value.

The extension principle developed by Zadeh (1975) and later by Yager (1986) establishes how to extend the domain of a function on a fuzzy sets.

Suppose that f is a function from X to Y and A is a fuzzy set on X defined as $A = \mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + \dots + \mu_A(x_n)/x_n$.

The extension principle states that the image of fuzzy set A under the mapping $f(\cdot)$ can be expressed as a fuzzy set B defined as

$$B = f(A) = \mu_A(x_1)/y_1 + \mu_A(x_2)/y_2 + \dots + \mu_A(x_n)/y_n$$

where $y_i = f(x_i)$

If $f(\cdot)$ is a many-to-one mapping, then, for instance, there may exist $x_1, x_2 \in X, x_1 \neq x_2$, such that $f(x_1) = f(x_2) = y_*$, $y_* \in Y$. The membership degree at $y = y_*$ is the maximum of the membership degrees at x_1 and x_2 more generally, we have $\mu_B(y_*) = \max_{y=f(x_i)} \mu_A(x_i)$

A point to point mapping from a set A to B through a function is possible. If it is many to one for two x in A then the membership function value in set B is calculated for $f(x)$ as max value of MF.

Let

$$A = 0.1/-2 + 0.4/-1 + 0.8/0 + 0.9/1 + 0.3/2$$

and

$$f(x) = x^2 - 3.$$

Upon applying the extension principle, we have

$$\begin{aligned} B &= 0.1/1 + 0.4/-2 + 0.8/-3 + 0.9/-2 + 0.3/1 \\ &= 0.8/-3 + (0.4 \vee 0.9)/-2 + (0.1 \vee 0.3)/1 \\ &= 0.8/-3 + 0.9/-2 + 0.3/1, \end{aligned}$$

Fuzzy Relation:

CRISP MAPPINGS:

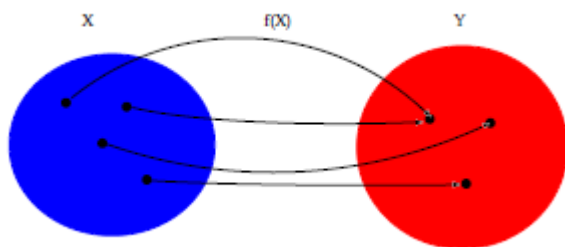


Fig. Mapping a relation

Consider the Universe $X = \{-2, -1, 0, 1, 2\}$

Consider the set $A = \{0, 1\}$

Using the Zadeh notation $A = \{ 0/-2 + 0/-1 + 1/0 + 1/1 + 0/2 \}$

Consider the mapping $y = |4x| + 2$

What is the resulting set B on the Universe $Y = \{2, 6, 10\}$

It is possible to achieve the results using a relation that express the mapping $y = |4x| + 2$.

Lets $X = \{-2, -1, 0, 1, 2\}$.

Lets $Y = \{0, 1, 2, \dots, 9, 10\}$

The relation

$$R = \begin{matrix} & & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \begin{matrix} -2 \\ -1 \\ 0 \\ 1 \\ 2 \end{matrix} & \left[\begin{array}{cccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right] \end{matrix}$$

$$B = A \circ R$$

$$A = \left\{ \frac{0}{-2} + \frac{0}{-1} + \frac{1}{0} + \frac{1}{1} + \frac{0}{2} \right\} \text{ or more conveniently } A = \{0, 0, 1, 1, 0\}$$

Using $\chi_B(y) = \bigvee_{x \in X} (\chi_A(x) \wedge \chi_R(x, y))$

we find

$$\chi_B(y) = \begin{cases} 1, & \text{for } y = 2, 6 \\ 0, & \text{otherwise} \end{cases}$$

Or

$$B = \left\{ \frac{0}{0} + \frac{0}{1} + \frac{1}{2} + \frac{0}{3} + \frac{0}{4} + \frac{0}{5} + \frac{1}{6} + \frac{0}{7} + \frac{0}{8} + \frac{0}{9} + \frac{0}{10} \right\}$$

Fuzzy Mappings:

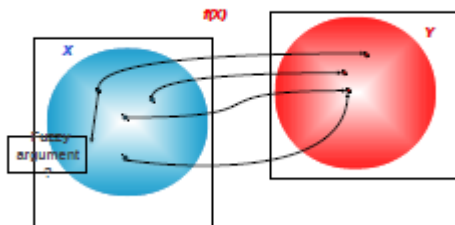


Fig. Fuzzy arguments mapping

Consider two universes of discourse X and Y and a function $y = f(x)$.

Suppose that elements in universe X form a fuzzy set A .

What is the image (defined as B) of A on Y under the mapping f ?

Similarly to the crisp definition, B is obtained as

$$\mu_B(y) = \mu_{f(A)}(y) = \bigvee_{y=f(x)} \mu_A(x)$$

Fuzzy vector is a convenient shorthand for calculations that use matrix relations.

Fuzzy vector is a vector containing only the fuzzy membership values.

Consider the fuzzy set:

$$B = \left\{ \frac{0}{0} + \frac{0.2}{1} + \frac{0.3}{2} + \frac{0.5}{3} + \frac{0.7}{4} + \frac{0.9}{5} + \frac{1}{6} + \frac{0}{7} + \frac{0}{8} + \frac{0}{9} + \frac{0}{10} \right\}$$

The fuzzy set B may be represented by the fuzzy vector b :

$$b = \{0, 0.2, 0.3, 0.5, 0.7, 0.9, 1, 0, 0, 0, 0\}$$

Now, we will be talking about fuzzy relation. If x and y are two universal sets, the fuzzy sets, the fuzzy relation R x y is given. As this is all ordered pair, $\mu_{R(x,y)}$ up on x y for all x y , belonging to the Cartesian space x , you associate $\mu_{R(x,y)}$ with each ordered pair.

What is the difference between fuzzy and crisp relation? In fuzzy this is missing, where $\mu_{R(x,y)}$ is a number in 0 and 1. $\mu_{R(x,y)}$ is a number between 0 and 1. This is the difference between crisp relation and fuzzy relation. In crisp relation, it was either 0 or 1. It is either completely connected or not connected, but in case of fuzzy, connection is a degree; that is, it is from 0 to 1.

Let X and Y be two universes of discourse. Then

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

The example is, let x equal to 1 2 3. Then x has three members, y has two members 1 and 2. If the membership function associated with each ordered pair is given by this e to the power minus x minus y whole squared. I is seen that this is the kind of membership function that is used to know, how close is the members of y are from members of x . Because, if I relate from 1 to 1 using this, then you can see 1 minus 1 is 0 that is 1 and 1 very close to each other; whereas, 2 and 1 is little far and 3 1 one is further far. This is a kind of relationship we are looking between these two sets.

Let us derive fuzzy relation. If this is the membership function, fuzzy relation is of course all the ordered pairs. We have to find out 11 2 2 1 2 2 3 1 and 3 2. These are all the sets of ordered pairs and associated membership functions. You just compute e to the power minus x minus y whole square. Here, 1 1 1 minus 1 whole square, 1 2 1 minus 2 whole square, 2 1 2 minus 1 whole square, 2 2 2 minus 2 whole square, 3 1 3 minus 1 whole square, 3 2 3 minus 2 whole square and if you compute them, you find 1 0.4 3 0.4 3 1 0.1 6 0.4 3. This is your membership function. This is one way to find relation.

Normally, I know, it is easier to express the relation in terms of a matrix instead of this continuum fashion, where each ordered pair is associated with membership function. It is easier to appreciate the relation by simply representing them in terms of matrix. How do we do that? This is my x 1 2 3 y is 1 21 the membership function associated was 1 1 2 membership is 0.4 3 2 1 0.4 3 2 2 1 3 1 0.1 6 and 3 2 is 0.4 3 that you can easily verify here 1 3 0.4 3 0.1 6 and 1.

The membership function describes the closeness between set x and y . It is obvious that higher value implies stronger relations. What is the stronger relation? It is between 1 and 1, and they are very close to each other, and 2 and 2; they are very close to each other. Closeness between 2 and

2, between 1 and 1 is actually 1 and 1. They are very close to each other; similarly, 2 and 2. If I simply say numerical closeness, then 2 and 2 are the closest, and 1 and 1 are the closest. That is how these are the closest. Higher value implies stronger relations.

This is a formal definition of fuzzy relation; it is a fuzzy set defined in the Cartesian product of crisp sets; crisp sets x_1 x_2 until x_n . A fuzzy relation R is defined as μ_R upon x_1 to x_n , where x_1 to x_n belongs to the Cartesian product space of x_1 until x_n ; whereas, this μ_R the fuzzy membership associated is a number between 0 and 1.

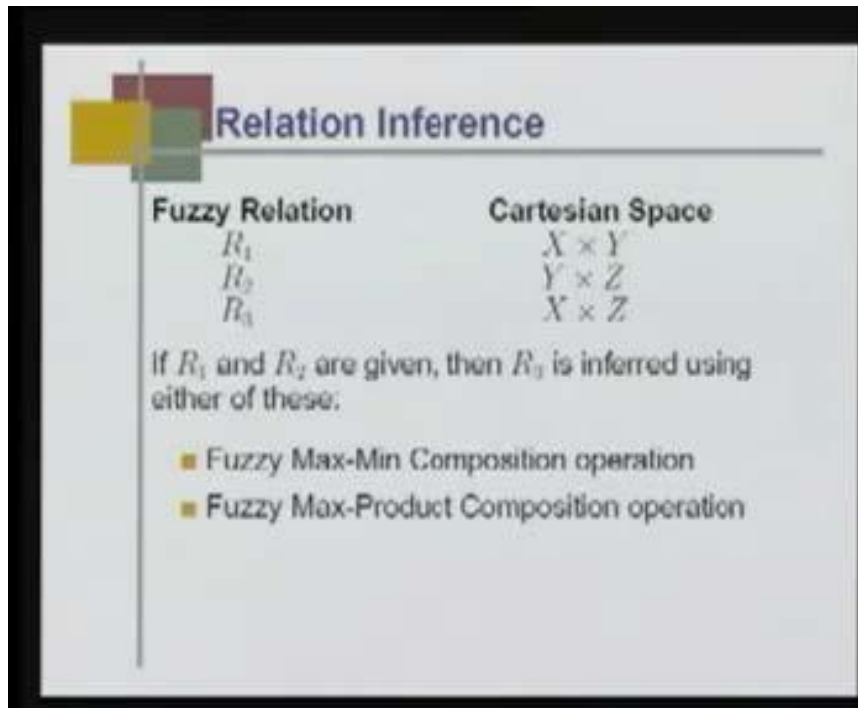


Fig. Inferring Fuzzy relation

Max-min composition or Max-min product:

Let \mathcal{R}_1 and \mathcal{R}_2 be two fuzzy relations defined on $X \times Y$ and $Y \times Z$, respectively. The **max-min composition** of \mathcal{R}_1 and \mathcal{R}_2 is a fuzzy set defined by

$$\mathcal{R}_1 \circ \mathcal{R}_2 = \{[(x, z), \max_y \min(\mu_{\mathcal{R}_1}(x, y), \mu_{\mathcal{R}_2}(y, z))] | x \in X, y \in Y, z \in Z\},$$

or, equivalently,

$$\begin{aligned} \mu_{\mathcal{R}_1 \circ \mathcal{R}_2}(x, z) &= \max_y \min[\mu_{\mathcal{R}_1}(x, y), \mu_{\mathcal{R}_2}(y, z)] \\ &= \bigvee_y [\mu_{\mathcal{R}_1}(x, y) \wedge \mu_{\mathcal{R}_2}(y, z)], \end{aligned}$$

It is a sort of matrix multiplication but memberships are not multiplied.

We will now explain, max min composition operation using an example that makes things much more clear. This is my matrix, relational matrix R_1 relating x and y and R_2 relating y and z . I have to find out the relational matrix from x to z using fuzzy rule of composition. We normally write R_3 is R_1 composition R_2 . Using max min composition, how do we compute R_3 ?



Fuzzy Max-Min Composition operation

Let us consider two fuzzy relations R_1 and R_2 defined on cartesian spaces $X \times Y$ and $Y \times Z$ respectively. The max-min composition of R_1 and R_2 is a fuzzy set defined on cartesian space $X \times Z$ as

$$R_3 = R_1 \circ R_2 = \left\{ \frac{\mu_{R_3}(x, z)}{(x, z)} \right\}$$

where

$$\mu_{R_3}(x, z) = \max_y \{ \min(\mu_{R_1}(x, y), \mu_{R_2}(y, z)) \mid x \in X, y \in Y, z \in Z \}$$



Max-min composition operation

$R_1 =$

| | | | |
|-------|-------|-------|--|
| | x_1 | x_2 | |
| y_1 | 0.1 | 0.2 | |
| y_2 | 0.4 | 0.5 | |
| y_3 | 0.7 | 0.8 | |

$R_2 =$

| | | | |
|-------|-------|-------|--|
| | z_1 | z_2 | |
| y_1 | 0.9 | 0.8 | |
| y_2 | 0.7 | 0.6 | |

Then

$R_3 =$

| | | | |
|-------|-------|-------|--|
| | z_1 | z_2 | |
| x_1 | 0.1 | 0.2 | |
| x_2 | 0.4 | 0.5 | |
| x_3 | 0.7 | 0.7 | |

$$\begin{array}{r} 0.1 \quad 0.9 \quad 0.1 \\ \hline 0.2 \quad 0.7 \quad 0.2 \\ \hline 0.2 \end{array}$$

$$\max\{\min(0.1, 0.9), \min(0.2, 0.7)\} = 0.2$$

$$\begin{array}{r} 0.4 \quad 0.9 \quad 0.4 \\ \hline 0.5 \quad 0.7 \quad 0.5 \\ \hline \max \quad 0.5 \end{array}$$

$$R_3 = R_1 \circ R_2$$

Fig. Example Max-Min composition or Max-min product

I want to now build a relationship between R1 & R2. Membership associated with x1 is 0.1 and z1 is 0.9. Let me put it very precise, x1 x2 x3 z1 and z2; if you look at what we will be doing here, This is my x1 row and this is my z1 column. What I do, x1 row and z1 column; I put them parallel and find out what is minimum. Here, minimum is 0.1 and here minimum is 0.2. After that, I find out what is the maximum, which is 0.2. This is what maximum of minimum 0.1. 0.9 is minimum 0.2. 0.7 is 0.2. This is how we found out. The easiest way if I want to find out is this one; this x1 x2 and z1. x2 means this row which is 0.4 and 0.5 and x2 and z1. z1 is again 0.9 and 0.7. I will find out. Minimum here is 0.4, minimum here is 0.5 and maximum here is 0.5. You get this 0.5. Similarly, we can compute all the elements in R3 using a max min composition operation. As usual, any max min composition can follow certain properties, associative and distributive over union. That is P fuzzy composition Q union R is P composition Q union P composition R.

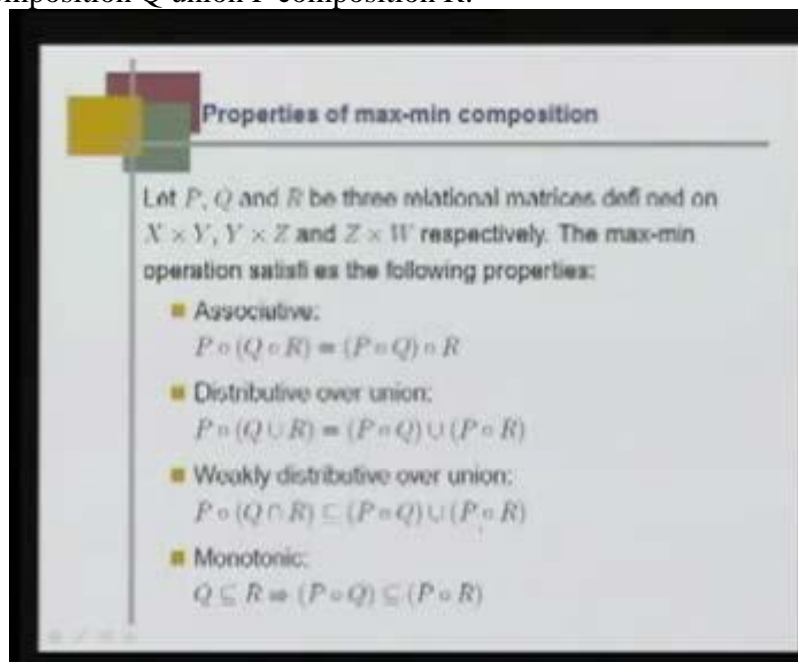


Fig. Properties of max-min composition

Similarly, weakly distributed over union is P composition, Q intersection, R is a subset of P composition. Q union P composition R monotonic Q is a subset of R implies that, P composition Q is a subset of P composition R.

Max-product composition:

$$\mu_{R_1 \circ R_2}(x, z) = \max_y [\mu_{R_1}(x, y) \mu_{R_2}(y, z)].$$

Now, again, the same example we have taken R

1, R2 and R3. Now, I want to find out from R1 and R2, what R3 using max product composition is.

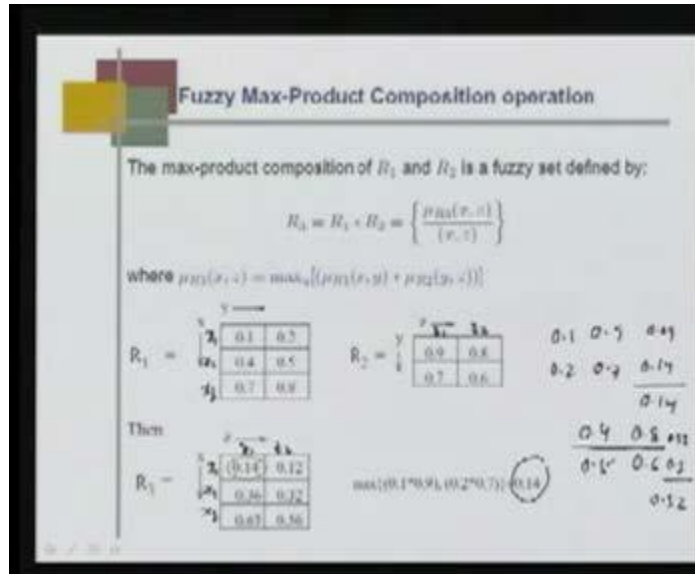


Fig. Example Max-product composition

Let us say, this is $x_1 \ x_2 \ x_3 \ z_1 \ z_2 \ z_1 \ z_2$ and this is $x_1 \ x_2 \ x_3$ for x_1 . I take this row which is 0.1 0.2 and finding the relation the fuzzy membership associate x_1 and z_1 . I take the column from z_1 which is 0.9 0.7 and I multiply them here 0.1 0.9 is point 0.9 0.2 0.7 is 0.1 4 and find out what is the maximum. This is the maximum 0.1 4.

I take another example. Let us find out the relationship between x_2 and z_2 ; for x_2 the row is 0.4 0.5 and z_2 the column is 0.8 0.6. Corresponding to this, if I multiply I get 0.4 0.8 is 0.3 2 0.5 0.6 is 0.3. Maximum is 0.3 2. This is 0.4 3 0.3 2. This is where it is 0.1. The answer is here, the R_3 and if I go back, if I look, R_3 here is different.

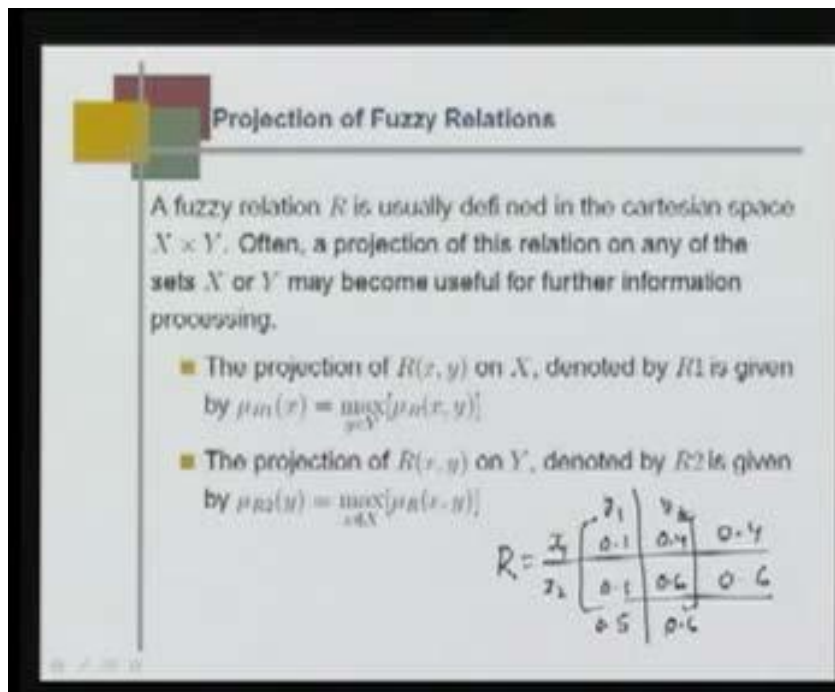


Fig. Projection of fuzzy relation

Projection of fuzzy relation:

A fuzzy relation R is usually defined in the Cartesian space x and x and y . Often a projection of this relation on any of the sets x or y , may become useful for further information processing.

The projection of $R \times y$ on x denoted by $R \downarrow$ is given by $\mu_{R \downarrow}(x) = \max_y \mu_R(x, y)$. So, y belongs to $y \in \mu_R(x, y)$. The meaning is that if I have R , this is x_1 and x_2 and this is y_1 and y_2 , and this is 0.1 0.4 and this is 0.5 0.6 . If these are the membership functions associated with x_1 y_1 x_2 y_2 is 0.4 x_2 y_1 is 0.5 x_2 y_2 is 0.6 . projection, which means for x projection, I find out what the maximum is. Overall, y in this case maximum is 0.4 and for x_2 the max maximum projection is if I took it here, 0.6 . Similarly, if I make projection of R , x , y over x , what is the maximum? This is 0.5 and this is 0.6 . This is called x projection and y projection of a relation matrix R .

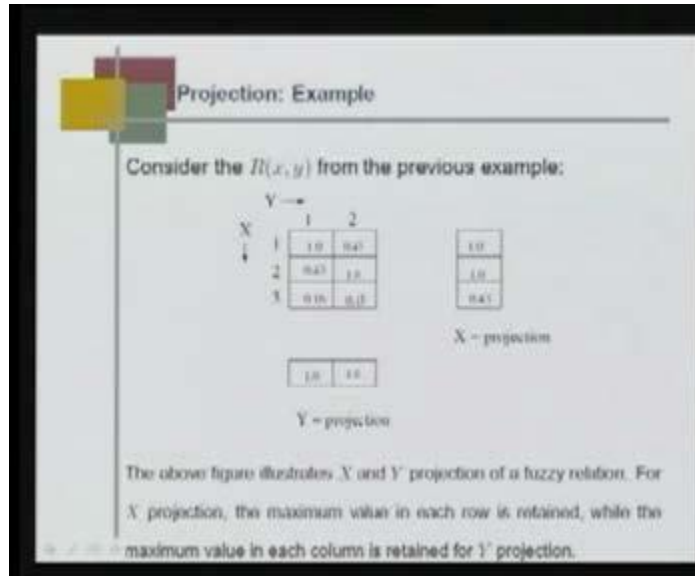


Fig. Example of projection

We repeat another example. We have x as 3 components 1 2 3, y has 2 components 1 and 2. This is the previous example that we had 1 0.4 3 0.4 3 1 0.1 6 0.4 3 . x projection would be 1 0.4 3 maximum 1 0.4 3 1 maximum 1 0.1 6 0.4 3 maximum 0.4 3 . Above figure illustrates x and y projection of fuzzy relation. For x projection, the maximum value in each row is retained. What is the maximum value in each row? Here, x projection maximum value in each row is retained, while the maximum value in each column is retained for y projection.

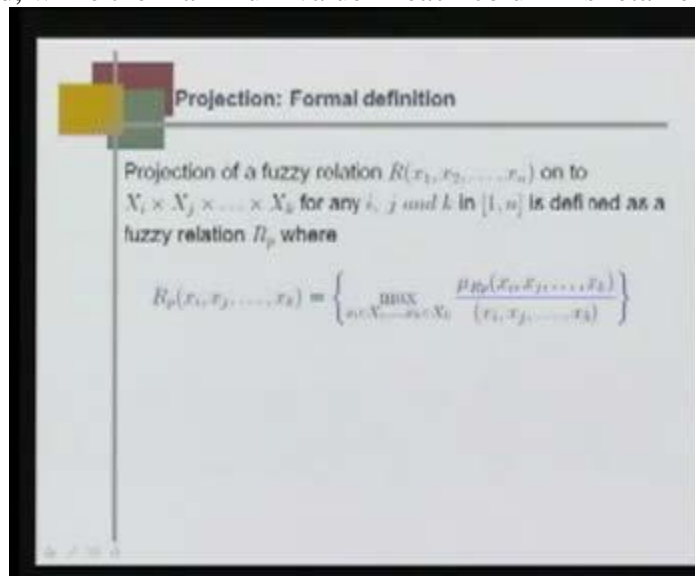


Fig. Definition of projection

This is our formal definition of a fuzzy relation, projection of a fuzzy relation R on to any of its set in the Cartesian product space; that is in the Cartesian product space. This is our Cartesian product space and for that, we can map this one to any of these i or j or k ; whatever

it is, for any value, then is defined as a fuzzy relation R_p , where R_p is defined as maximum over X_i until X_k , where this is our $X_i X_j X_k$ and this is μ_{R_p} .

First, we talked about fuzzy relation projection of fuzzy relation. Once we have projection of fuzzy relation, we can extend the projection to again infer what should be the relation. This kind of technique may be useful in coding the information, where we have a huge number of information and we want to transfer such a kind of projection and from projection to extension would be beneficial for coding operation.

The crisp relation and fuzzy relation:

the difference is that in crisp relation; the index is either 0 or 1 that is, either complete relation or no relation. But in fuzzy the membership grade is either 0 or 1; Whereas, in fuzzy the relation has a grade from 0 to 1. Fuzzy composition rule; max min composition max product composition unlike in crisp relation, where both max min and max product gives you the same answer; whereas in fuzzy composition, max min and max product will give two different answers .

LECTURE-7

Fuzzy If-then rules:

A fuzzy if-then rule (also known as fuzzy rule, fuzzy implication, or fuzzy conditional statement) assumes the form

If x is A then y is B

“x is A” is antecedent or premise which tells the fact

“y is B” is consequence or conclusion

The whole statement is the rule.

Eg. If tomato is red then it is ripe.

These if then rules are the base of fuzzy reasoning.

If then rules are of different types:

1. Single rule with single antecedent
2. Single rule with multiple antecedent
3. Multiple with multiple antecedent

Steps of Fuzzy reasoning:

Shown in fig. For 2 rules what will be the consequent MF after aggregation

1. Degree of compatibility
2. Firing strength
3. Qualified consequent MF
4. Aggregate all qualified consequent MFs to obtain an overall MF

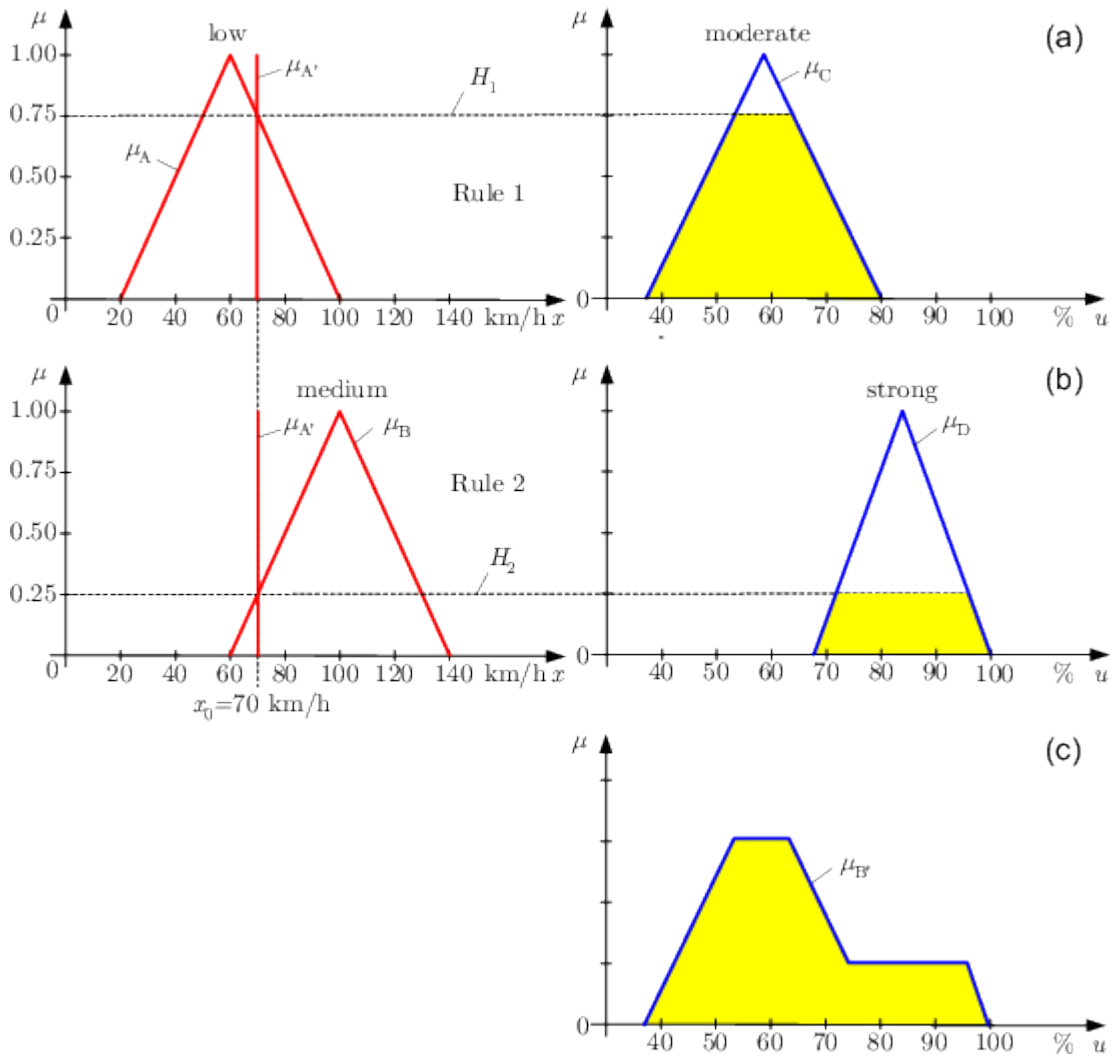


Fig. Fuzzy reasoning, deriving output

FUZZY MODELLING:

Fuzzy Inferencing

The process of fuzzy reasoning is incorporated into what is called a Fuzzy Inferencing System. It is comprised of three steps that process the system inputs to the appropriate system outputs. These steps are 1) Fuzzification, 2) Rule Evaluation, and 3) Defuzzification. The system is illustrated in the following figure.

Each step of fuzzy inferencing is described in the following sections.

Fuzzification

Fuzzification is the first step in the fuzzy inferencing process. This involves a domain transformation where crisp inputs are transformed into fuzzy inputs. Crisp inputs are exact inputs measured by sensors and passed into the control system for processing, such as temperature, pressure, rpm's, etc.. Each crisp input that is to be processed by the FIU has its own group of membership functions or sets to which they are transformed. This group of membership functions exists within a universe of discourse that holds all relevant values that the crisp input can possess. The following shows the structure of membership functions within a universe of discourse for a crisp input.

where:

degree of membership: degree to which a crisp value is compatible to a membership function, value from 0 to 1, also known as truth value or fuzzy input.

membership function, MF: defines a fuzzy set by mapping crisp values from its domain to the sets associated degree of membership.

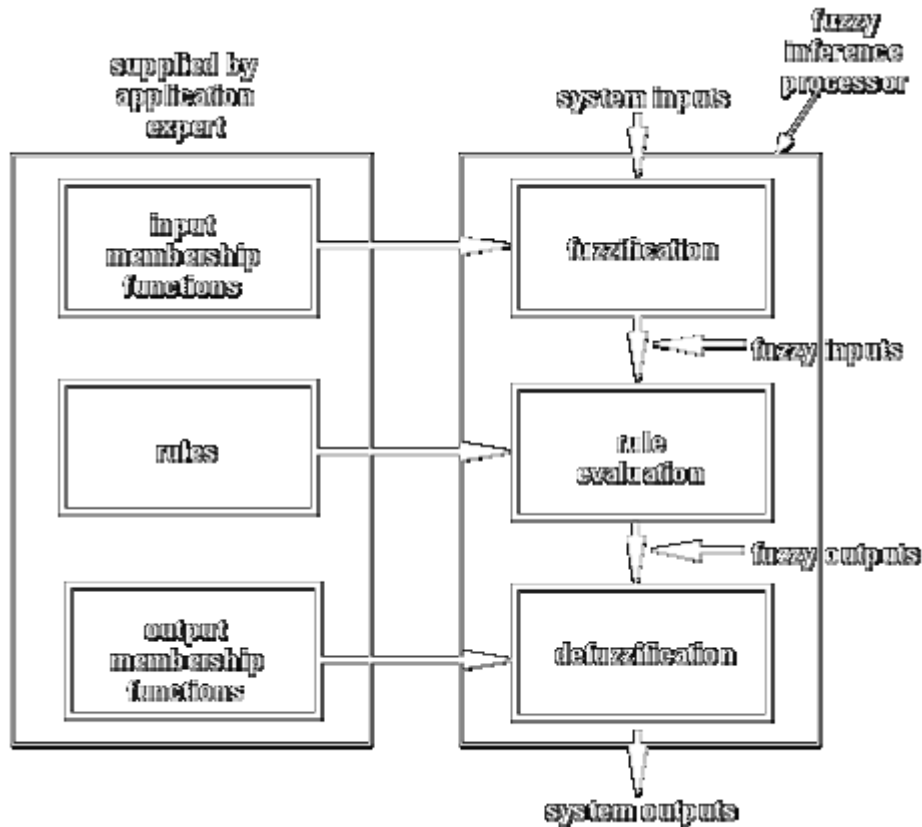


Fig. Fuzzy inferencing system

crisp inputs: distinct or exact inputs to a certain system variable, usually measured parameters external from the control system, e.g. 6 Volts.

label: descriptive name used to identify a membership function.

scope: or domain, the width of the membership function, the range of concepts, usually numbers, over which a membership function is mapped.

universe of discourse: range of all possible values, or concepts, applicable to a system variable.

When designing the number of membership functions for an input variable, labels must initially be determined for the membership functions. The number of labels correspond to the number of regions that the universe should be divided, such that each label describes a region of behavior. A scope must be assigned to each membership function that numerically identifies the range of input values that correspond to a label.

The shape of the membership function should be representative of the variable. However this shape is also restricted by the computing resources available. Complicated shapes require more complex descriptive equations or large lookup tables. The next figure shows examples of possible shapes for membership functions.

When considering the number of membership functions to exist within the universe of discourse, one must consider that:

i) too few membership functions for a given application will cause the response of the system to be too slow and fail to provide sufficient output control in time to recover from a small input change. This may also cause oscillation in the system.

ii) too many membership functions may cause rapid firing of different rule consequents for small changes in input, resulting in large output changes, which may cause instability in the system.

These membership functions should also be overlapped. No overlap reduces a system based on Boolean logic. Every input point on the universe of discourse should belong to the scope of at least one but no more than two membership functions. No two membership functions should have the same point of maximum truth, (1). When two membership functions overlap, the sum of truths or grades for any point within the overlap should be less than or equal to 1. Overlap should not cross the point of maximal truth of either membership function.

The fuzzification process maps each crisp input on the universe of discourse, and its intersection with each membership function is transposed onto the μ axis as illustrated in the previous figure. These μ values are the degrees of truth for each crisp input and are associated with each label as fuzzy inputs. These fuzzy inputs are then passed on to the next step, Rule Evaluation.

Fuzzy If then Rules :

We briefly comment on so-called *fuzzy IF-THEN rules* introduced by Zadeh. They may be understood as partial imprecise knowledge on some crisp function and have (in the simplest case) the form IF x is A_i THEN y is B_i . They should **not** be immediately understood as implications; think of a *table* relating values of a (dependent) variable y to values of an (independent variable) x :

| | | | |
|-----|-------|-----|-------|
| x | A_1 | ... | A_n |
| y | B_1 | ... | B_n |

A_i, B_i may be crisp (concrete numbers) or fuzzy (small, medium, ...) It may be understood in two, in general non-equivalent ways: (1) as a listing of n possibilities, called Mamdani's formula:

$$MAMD(x,y) \equiv \bigvee_{i=1}^n (A_i(x) \& B_i(y))$$

(where x is A_1 and y is B_1 or x is A_2 and y is B_2 or ...). (2) as a conjunction of implications:

$$RULES(x,y) \equiv \bigwedge_{i=1}^n (A_i(x) \rightarrow B_i(y))$$

Both *MAMD* and *RULES* define a binary fuzzy relation (given the interpretation of A_i 's, B_i 's and truth functions of connectives). Now given a *fuzzy input* $A^*(x)$ one can consider the image B^* of $A^*(x)$ under this relation, i.e.,

$$B^*(y) \equiv \exists x(A(x) \& R(x,y))$$

where $R(x,y)$ is *MAMD*(x,y) (most frequent case) or *RULES*(x,y). Thus one gets an operator assigning to each fuzzy input set A^* a corresponding fuzzy output B^* . Usually this is combined with some *fuzzifications* converting a crisp input x_0 to some fuzzy $A^*(x)$ (saying something as "x is similar to x_0 ") and a *defuzzification* converting the fuzzy image B^* to a crisp output y_0 . Thus one gets a crisp function; its relation to the set of rules may be analyzed.

Rule Evaluation

Rule evaluation consists of a series of IF-Zadeh Operator-THEN rules. A decision structure to determine the rules require familiarity with the system and its desired operation. This knowledge often requires the assistance of interviewing operators and experts. For this thesis this involved getting information on tremor from medical practitioners in the field of rehabilitation medicine.

There is a strict syntax to these rules. This syntax is structured as:

IF antecedent 1 ZADEH OPERATOR antecedent 2 THEN consequent 1 ZADEH OPERATOR consequent 2.....

The antecedent consists of: input variable IS label, and is equal to its associated fuzzy input or truth value $\mu(x)$.

The consequent consists of: output variable IS label, its value depends on the Zadeh Operator which determines the type of inferencing used. There are three Zadeh Operators, AND, OR, and NOT. The label of the consequent is associated with its output membership function. The Zadeh Operator is limited to operating on two membership functions, as discussed in the fuzzification process. Zadeh Operators are similar to Boolean Operators such that: AND represents the intersection or *minimum* between the two sets, expressed as:

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

OR represents the union or *maximum* between the two sets, expressed as:

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

NOT represents the opposite of the set, expressed as:

$$\overline{\mu_A} = [1 - \mu_A(x)]$$

The process for determining the result or rule strength of the rule may be done by taking the minimum fuzzy input of antecedent 1 AND antecedent 2, min. inferencing. This minimum result is equal to the consequent rule strength. If there are any consequents that are the same then the maximum rule strength between similar consequents is taken, referred to as maximum or max. inferencing, hence min./max. inferencing. This infers that the rule that is most true is taken. These rule strength values are referred to as fuzzy outputs.

Defuzzification

Defuzzification involves the process of transposing the fuzzy outputs to crisp outputs. There are a variety of methods to achieve this, however this discussion is limited to the process used in this thesis design.

A method of averaging is utilized here, and is known as the Center of Gravity method or COG, it is a method of calculating centroids of sets. The output membership functions to which the fuzzy outputs are transposed are restricted to being singletons. This is so to limit the degree of calculation intensity in the microcontroller. The fuzzy outputs are transposed to their membership functions similarly as in fuzzification. With COG the singleton values of outputs are calculated using a weighted average, illustrated in the next figure. The crisp output is the result and is passed out of the fuzzy inferencing system for processing elsewhere.

Fuzzy Rule base and Approximate Reasoning an example:

What is fuzzy linguistic variable? Algebraic variables take numbers as values, while linguistic variables take words or sentences as values.

For example, let x be a linguistic variable with a label 'temperature'. The universe of discourse is temperature. In that universe, I am looking at a fuzzy variable x when I describe the temperature. The fuzzy set temperature denoted as T can be written as $T = \text{very cold, cold, normal, hot or very hot}$.

For each linguistic value, we get a specific membership function.

These are necessary because in the traditional sense, when we express worldly knowledge, we express them in natural language. So here it is. From computational perspective, such worldly knowledge can be expressed in terms of rule base systems.

Rule based systems:

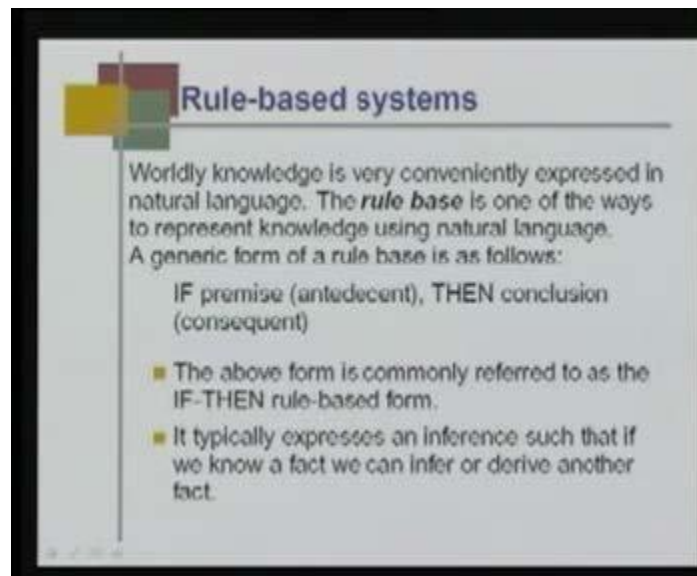


Fig. Basics of rule based system

The above form is commonly referred to as the IF-THEN rule-based form. It typically expresses an inference such that if we know a fact, we can infer or derive another fact. Given a rule, I can derive another rule or given a rule, if I know a rule and the associated relation, then given another rule, I can predict what should be the consequence.

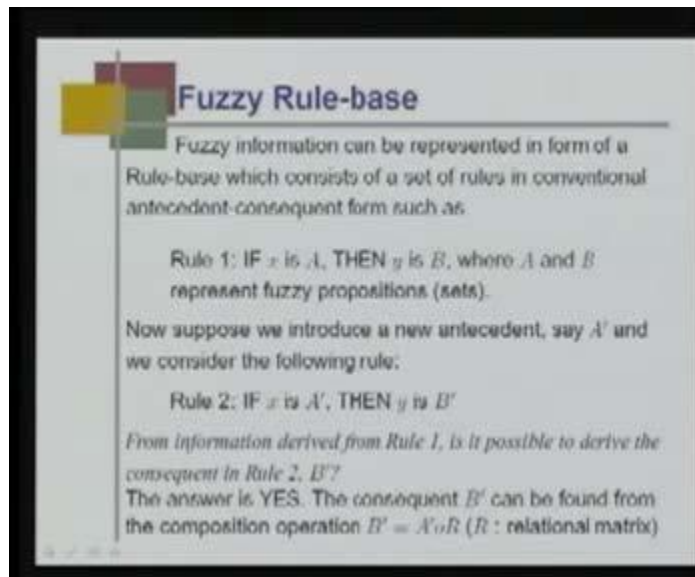


Fig.Fuzzy rules

This is a fuzzy rule base. Any worldly knowledge can be expressed in form in the form of a rule base. Now, when I talk about fuzzy rule base, fuzzy information can be represented in the form of a rule base, which consists of a set of rules in conventional antecedent and consequent form such as if x is A , then y is B , where A and B represent fuzzy propositions (sets). Suppose we introduce a new antecedent say A' and we consider the following rule if x is A' , then y is B' , from the information derived from rule 1, is it possible to derive the consequent in rule 2, which is B' ?

The consequent B' in rule 2 can be found from composition operation B' equal to A' . This is called the compositional rule of inference, the compositional operator with R .

Fuzzy implication Relation:

A fuzzy implication relation is another category, which will call Zadeh implication. This is if p implies q may imply either p and q are true or p is false. What we are saying is that just like a local Mamdani rule, we say p and q are true imply either p and q are true or p is false. Thus, p implies q means... p and q are simultaneously true, which is Mamdani local rule or if p is false, then p implies q has no meaning or p is false. This has taken an extra logic that is p and q or not p .

Thus, the relational matrix can be computed as follows. If I look at this, what is p and q ? p and q means minimum of $\mu_A(x)$ and $\mu_B(y)$. What is not p ? $1 - \mu_A(x)$. This entire thing has to be maximum of minimum of these and this, which is this statement. μ , the relational matrix elements are computed using this particular expression. Given a set of rules, we just learnt various schemes by which we can construct a relational matrix between the antecedent and the consequent. The next step would be to utilize this relational matrix for inference. This method is commonly known as compositional rule of inference, that is, associated with each rule we have a relational matrix. So, given a rule means given a relational matrix and given another antecedent, we compute a consequent.

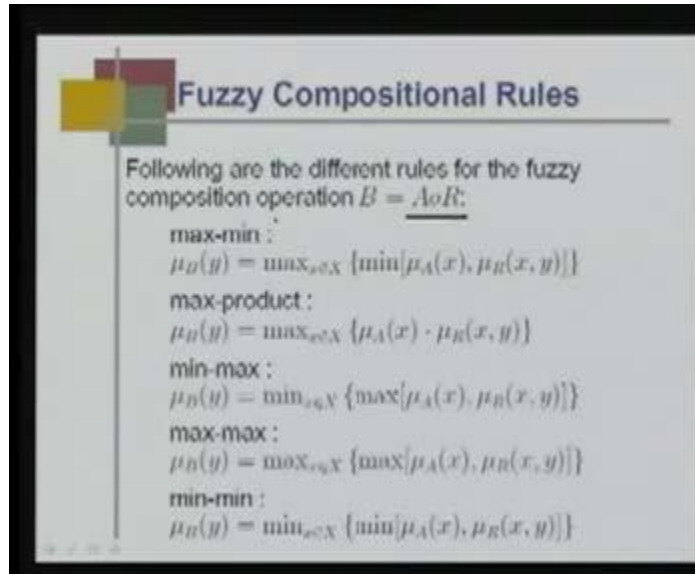


Fig. Compositional rules

This is derived using fuzzy compositional rules. The following are the different rules for fuzzy composition operation, that is, B equal to A composition R. R is the relational matrix associated with a specific rule, A is a new antecedent that is known, R is known, B is the new consequent for the new antecedent A. I have to find out what is B for this new A, given R. That is computed by A composition R and we have already discussed in the relation class that there are various methods and max-min is very popular.

First, we compute min and then max. Similarly, max-product: instead of min, we take the product and compute what is the maximum value. Similarly, min-max: instead of max-min, it is min-max. First, max and then min. Next, max-max and min-min. One can employ these looking at the behavior of a specific data.

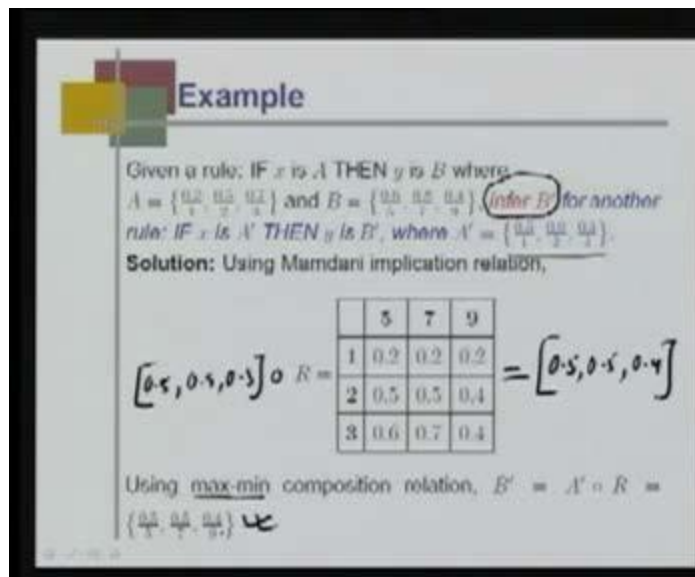


Fig. Example of Compositional rules

Now, we will take an example.

We are given a rule if x is A , then y is B , where A is this fuzzy set: 0.2 for 1, 0.5 for 2, and 0.7 for 3. This is a discrete fuzzy set. B is another fuzzy set that defines fuzzy membership

0.6 for 5, 0.8 for 7, and 0.4 for 9. The question is infer B dash for another rule if x is A dash, then y is B dash, where A dash is known. A is known, B is known, and A dash is known. What we have to find out is what B dash is. Infer B dash is the question that is being asked Using Mamdani implication relation, first we will find out between A... the first rule, that is, if x = A, then y is B. The relational matrix associated with this rule is... For R, how do we compute? A elements are 1, 2, and 3 and B elements are 5, 7, and 9. We have to find out now for 0.2. Here, we compare with all the elements in point B and with each element, we found what the minimum is. The minimum is always 0.2. Hence, the maximum of that is always 0.2. I have to find out the relational matrix between A and B.

The Mamdani principle means minimum, so between 1 and 5, 1 is associated with 0.2, and 5 is associated with 0.6, so the minimum is 0.2. Similarly, 1 is associated with 0.2, 7 is associated with 0.8, so for 1 and 7, the minimum is 0.2. Similarly, 1 is associated with 0.2, 9 is associated with 0.4, so from 1 to 9, the minimum membership is 0.2. Similarly, you can see that from 2 to all the elements 5, 7, 9, the minimum are 0.5, 0.5, and 0.4. Similarly, from 3 to 5, 7, and 9, we have 0.6, 0.7, and 0.4. These are the minimum fuzzy memberships between an element in A to element in B. That is how we compute the relational matrix.

Once we compute the relational matrix, then we use max-min composition relation to find out what is B dash, which is A dash (which is 0.5, 0.9, and 0.3) composition R and you can compute. This is my R. I have to find out my matrix. This is 0.5, 0.9, and 0.3. So this composition R is... you can easily see I take this row vector, put along the column matrix and I see what is the minimum for each case. You can easily see 0.2 will be minimum here, 0.5 will be minimum here, 0.3 and maximum is 0.5.

The first element is 0.5. Again, I take this place in parallel with this column and then, I find first minimum here is 0.2, here 0.5, here 0.3 and then maximum is again 0.5. Again, I take the same row vector, put along this column vector and then, I find here the minimum is 0.2, here minimum is 0.4, here minimum is 0.3 and the maximum is 0.4. This is the relation, this is the answer. This is our B dash. Given A, this is my B dash using fuzzy compositional principle or relation.

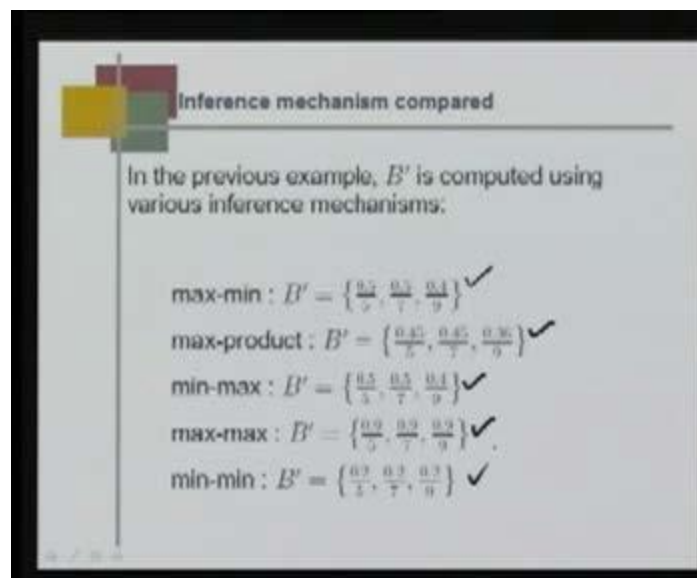


Fig. Comparison of compositional rules

There are other mechanisms also that we discussed. For the same example, if you use max-min, you get B dash; for max-product, you get another B dash; for min-max, you get another. Min-max and max are same for this example. Then, for max-max, you see that all the fuzzy membership are the maximum values and for min-min, they are the minimum values here.

Approximate reasoning:

means given any logical system, we do not have, it is very difficult to make an exact result. That is why from engineering perspective, we are more liberal. We do not want to be so precise. As long as our system works, we are happy; if our control system works, we are happy.

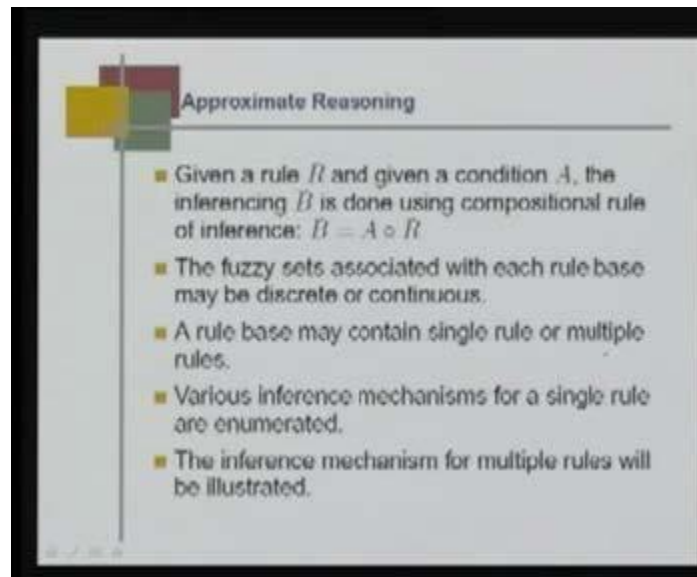


Fig. Approximate reasoning

Approximate reasoning. We have set up rules so we use a specific compositional rule of inference and then we infer the knowledge or the consequence. Given a rule R (R is the relational matrix associated with a specific rule) and given a condition A , the inferencing B is done using compositional rule of inference B equal to A composition R . The fuzzy sets associated with each rule base may be discrete or continuous, that is, A may be discrete or A and B may be discrete or continuous.

A rule base may contain a single rule or multiple rules. If it is continuous, I cannot define what the R relational matrix is. It is very difficult because it will have infinite values. R is not defined. That is why for continuous, we apply compositional rule of inference but the method to compute is different. A rule base may contain single rule or multiple rules. Various inference mechanisms for a single rule are enumerated. Various mechanism means we talked about min-max, max-min, max-max, min-min and so on. The inference mechanism for multiple rules.

Single rule:

Now, we will take the examples one by one. Single rule with discrete fuzzy set. We talked about a fuzzy set that may consist of a single rule or multiple rules. It can be discrete fuzzy set or a continuous fuzzy set. We will try to understand how to make approximate reasoning for such a rule base using the methods that we just enumerated. For each rule, we compute what is the relational matrix if it is discrete fuzzy set and then we use compositional rule of inference to compute the consequence given an antecedent. That is for discrete fuzzy set. We have already talked about this but again, for your understanding, I am presenting another example for single rule with discrete fuzzy set.

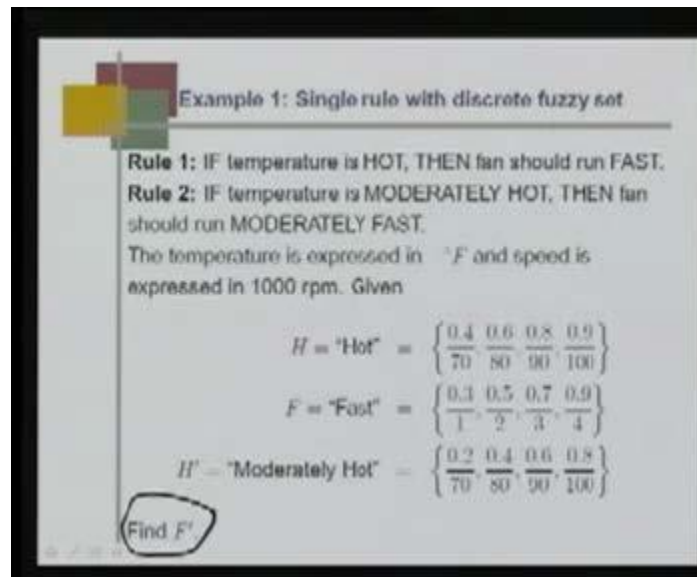


Fig. Single rule

Rule 1: If temperature is hot, then the fan should run fast. If temperature is moderately hot, then the fan should run moderately fast. In this example, we are given the temperature is in degree Fahrenheit and the speed is expressed as 1000 rpm. The fuzzy set for hot H is for 70 degree Fahrenheit, 80 degree Fahrenheit, 90 degree Fahrenheit, and 100 degree Fahrenheit, the membership values are 0.4, 0.6, 0.8, and 0.9. Similarly, for the fuzzy set F, for which the fan should run fast, the fuzzy set is for 1000 rpm, the membership is 0.3, for 2000 rpm, the membership is 0.5, for 3000 rpm, the membership 0.7, and for 4000 rpm, the membership is 0.9.

Given H dash, which is moderately hot, to be for 70... moderately hot means it is a little more hot. So, same temperature obviously and their corresponding membership values will reduce, because if I am describing moderately hot, they will have the same temperature but the membership values will be less. You can easily see here that for 70, instead of 0.4, now it is 0.2; for 80, instead of 0.6, it is 0.4; for 90, instead of 0.8, it is 0.6; for 100, instead of 0.9, it is 0.8. This is moderately hot. Now, the question is find F dash.

I hope you are clear with this question. The question is very simple. We are given rule 1, we have defined what is the fuzzy set hot and fuzzy set fast by these two statements and in the second rule for moderately hot, we know the fuzzy set. We do not know what the fuzzy set is corresponding to moderately hot, that is, moderately fast. We do not know moderately fast. Find out F dash. If H, then F. If H dash, then F dash. Find out F dash. First, what do we do?

Corresponding to rule 1, we found out what is R. This is for rule 1. We knew that the membership functions for H were 0.4, 0.6, 0.8, and 0.9, and for fast, the membership functions were 0.3, 0.5, 0.7, and 0.9. If you look at this, these are my H values, the crisp values: 70 degree Fahrenheit, 80 degree Fahrenheit, 90 degree Fahrenheit, and 100 degree Fahrenheit. This is my speed: 1000 rpm, 2000 rpm, 3000 rpm, and 4000 rpm.

Between 70 and 1000 rpm, the entry would be minimum of these two (Refer Slide Time: 41:57), which is 0.3. Similarly, between 0.4 and 0.5, the minimum would be again 0.4 and then between 0.4 and 0.7, it will be 0.4, and for 0.4 and 0.9, it is 0.4.

Similarly, we go to the next one, which is 0.6. For 0.6, 0.3 minimum 0.3, for 0.6 and 0.5, the minimum is 0.5, for 0.6 and 0.7, minimum is 0.6, for 0.6 and 0.9, it is 0.6. Similarly, you can fill all other cells here with their values: 0.3, 0.5, 0.7, 0.8, 0.3, 0.5, 0.7, and 0.9. This is my relation matrix associated with rule 1: if H, then F. Now, what I have to do is I have to find

out F dash given H dash, using the fuzzy compositional rule of inference, which is represented like this.

Example 1: Solution

The relational matrix R that connects fuzzy sets H and F is computed using Mamdani implication:

| | | | | |
|-----|-----|-----|-----|-----|
| | 1 | 2 | 3 | 4 |
| 70 | 0.3 | 0.4 | 0.4 | 0.4 |
| 80 | 0.3 | 0.5 | 0.6 | 0.6 |
| 90 | 0.3 | 0.5 | 0.7 | 0.8 |
| 100 | 0.3 | 0.5 | 0.7 | 0.9 |

Handwritten notes on the right side of the matrix:

$H = (0.4, 0.6)$
 $0.8, 0.9$
 $F = (0.3, 0.5, 0.7, 0.9)$

Handwritten notes on the left side:

Rule 1
 $H' = (0.2, 0.4, 0.6, 0.8)$

Equation for the composition:

$$F' = H' \circ R = \left\{ \frac{0.3}{1}, \frac{0.5}{2}, \frac{0.7}{3}, \frac{0.8}{4} \right\}$$

Max - Min

Fig. Relational matrix

F dash is H dash compositional rule of inference with R. This is max-min composition operation. First, we take the min and then compute. H dash is given as 0.2, 0.4, 0.6, and 0.8.

Example 2: Multiple rules with discrete fuzzy sets

Rule 1: IF height is TALL, THEN speed is HIGH
 Rule 2: IF height is MEDIUM, THEN speed is MODERATE

The fuzzy sets for height (in feet) and speed (in m/s) are as follows:

$$H_1 = \text{"Tall"} = \left\{ \frac{0.5}{5}, \frac{0.9}{6}, \frac{1}{7} \right\} \quad R_1 = \text{"High"} = \left\{ \frac{0.1}{3}, \frac{0.7}{7}, \frac{0.9}{9} \right\}$$

$$H_2 = \text{"Medium"} = \left\{ \frac{0.6}{5}, \frac{0.7}{6}, \frac{0.6}{7} \right\} \quad R_2 = \text{"Moderate"} = \left\{ \frac{0.6}{5}, \frac{0.8}{7}, \frac{0.7}{9} \right\}$$

Given $H' = \text{"ABOVE AVERAGE"} = \left\{ \frac{0.5}{5}, \frac{0.9}{6}, \frac{0.2}{7} \right\}$, find $S' = \text{"ABOVE NORMAL"}$.

Fig. Multiple rules

This is my H dash (moderately hot) and I have to do compositional inference between H dash and R. Again, I am repeating so that you understand how to compute it. You put this row

vector along this column vector first . For each element, you find out what is the minimum. You see that here it is 0.2, 0.3, 0.3, and 0.3 and the maximum of that is 0.3.

Similarly, you take again these values and put them here vertically. Here, the minimum is 0.2, here 0.4, here 0.5, here 0.5, and maximum is 0.5. I am sure you will see here it is 0.7, but in this case, you find that if you take this here, it is 0.2, here 0.4, here 0.6, here 0.8, and maximum is 0.8. F dash is 0.3, 0.5, 0.7, and 0.8. That is how we infer or we do approximate reasoning for a rule base. This is a very simple case.

Multiple rule:

There are two rules now. Rule 1 is if height is tall, then speed is high. Rule 2: if height is medium, then speed is moderate. This is describing a rule for a person as to how fast he can walk. Normally, those who are tall can walk very fast and those who are short, naturally their speed will be less. This is one fuzzy rule that expresses the speed of a person while walking. If height is tall, then speed is high and if height is medium, then speed is moderate. For this, the fuzzy memberships are defined as tall, high, medium, and moderate.

Tall is 0.5, 0.8, and 1 for various feet like 5, 6, and 7. For speed is high, for 5 meter per second, 7 meter per second, and 9 meter per second, the corresponding membership values are 0.4, 0.7, and 0.9. For H2, which is medium height, the corresponding fuzzy membership... you can easily see that when I say medium in this fuzzy set, 5 has 0.6, 6 has 0.7, and 7 has 0.6. The moderate speed is 0.6 for 5 meter per second, 0.8 for 7 meter per second, and 0.7 for 9 meter per second. If this is the fuzzy set given, now the question is given H dash, which is above average, and the corresponding fuzzy set is 0.5, 0.9, 0.8 for three different heights, find S dash, the speed above normal. I hope the question is very clear to you.

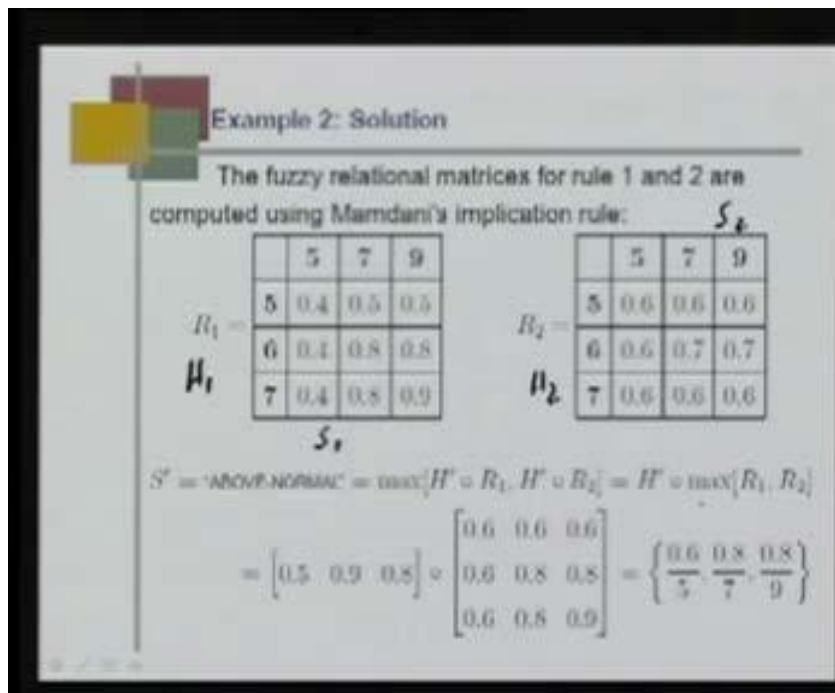


Fig. Relational matrix for 2 rules

We have two rules. If height is tall, then speed is high; tall is defined and high is defined. If height is medium, then speed is moderate. I have already defined the fuzzy sets for both medium as well as moderate. They are all discrete fuzzy sets. Now, you are presented with new data and what is that new data? You are presented with a data called above average, which is 0.5, 0.9, and 0.8 for three different heights for 5, 6, and 7. Then, find S dash equal to above normal, that is, if height is above average, then the speed should be above normal.

This is the solution of this example. We have two rules. Naturally, we will have two relational matrices: R1 for rule 1 and R2 for rule 2. I will not go in detail of how we compute. You simply you go the antecedent and consequent, look at the membership function, find the minimum for each entry. Here, these are the heights and these are the speeds; 5, 6, 7 feet is the height and 5, 7, and 9 meter per second are the speeds of the individuals.

Now, you check the fuzzy sets and corresponding to each fuzzy set, find out what is the minimum membership function. For 5, 5, you will find the membership function is 0.4, minimum 0.5, 0.5, 0.4, 0.8, 0.8, 0.4, 0.8, 0.9. You can verify this. Similarly, R2 can be found out. Taking the minimum membership entry between these two fuzzy sets, that is, if I say this is H1 and S1 and this is H2 and S2. Look at these two fuzzy sets, find out what the minimum entries are for each relation and then, how do we compute S dash above normal? We have now two relational matrices. It is very simple. We do two composition operations: H dash composition with R1 (this one) and again, H dash composition R2 and then, we take the maximum of that, maximum of these two.

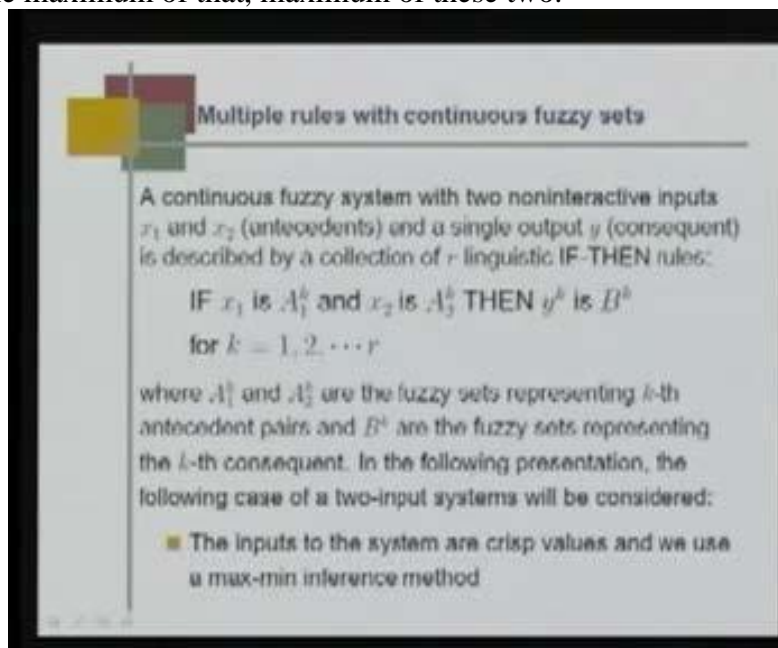


Fig. Multiple rule with continuous fuzzy sets

You can easily see that the maximum of H dash composition R1, H dash composition R2. You can easily see that because H dash is common, this particular expression is the same as H dash composition max of R1 and R2. This is R1 and R2. We look at all those entries wherever it is the maximum: for 0.4 and 0.6, the maximum is 0.6; for 0.5 and 0.6, the maximum is 0.6; for 0.5 and 0.6, the maximum is 0.6. You see the last element here 0.9 here and 0.6, so this is 0.9. Like that, for all entries of R1 and R2, whatever the maximum values, you put these values here (that is called maximum R1 and R2) and take a composition with H dash. So H dash composition max of R1 and R2. H dash is already given as 0.5, 0.9, and 0.8. If you do this composition, you get 0.6, 0.8, and 0.8. I hope this clears your concept of how

we compute or we do approximate reasoning in a rule base. Similarly, if there are multiple rules, we have no problem and we can go ahead with the same principle.

The last section is the multiple rules with continuous fuzzy sets. We talked about discrete fuzzy set, but if it is continuous fuzzy sets, how do we deal with that? Normally, a continuous fuzzy system with two non-interactive inputs x_1 and x_2 , which are antecedents, and a single output y , the consequent, is described by a collection of r linguistic IF-THEN rules. Where the rule looks like this: If x_1 is A_1^k and x_2 is A_2^k , then y is B^k , where k is 1, 2 up to r . This is the k th rule. Similarly, we can have rule 1, rule 2, rule 3, up to rule r . In this particular rule, A_1^k and A_2^k are the fuzzy sets representing the k th antecedent pairs and B^k are the fuzzy sets representing the k th consequent. In the following presentation, what we will do now is we will take a two-input system and two-rule system just to illustrate how we infer from a rule base where the fuzzy sets are continuous. The inputs to the system are crisp values and we use a max-min inference method.

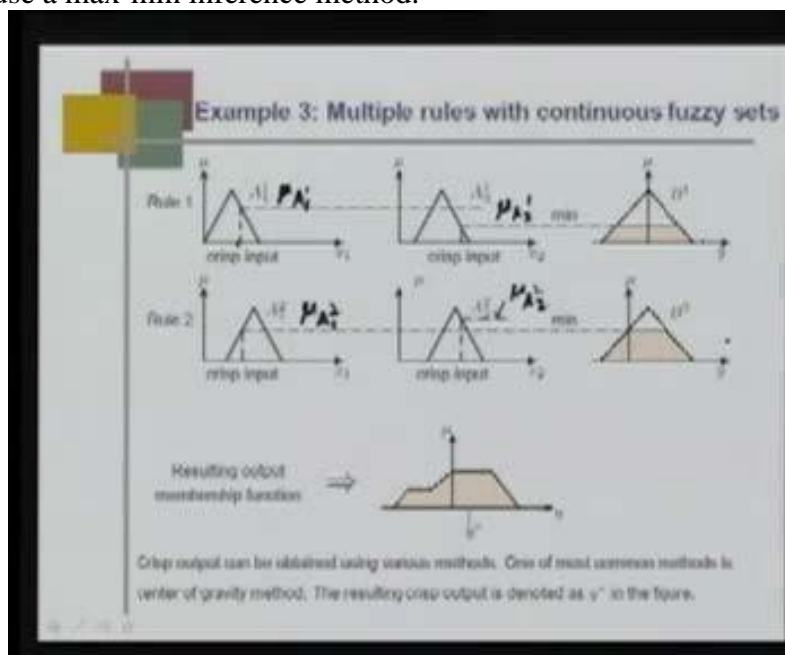


Fig. Viewing multiple rules

We have two rules here represented graphically. You can see there are two variables x_1 and x_2 . There are two fuzzy variables and for each rule, we have a consequent y . The first rule says that if x_1 is A_1^1 and x_2 is A_2^1 , then y is B^1 .

Similarly, if x_1 is A_1^2 , x_2 is A_2^2 , then y is B^2 . Now, how do we infer? Given a crisp input, a new input is given, crisp input in the domain of x_1 and another crisp input in the domain of x_2 . There can be a system whose two variables can be temperature as well as pressure. You can easily think x_1 to be the temperature and x_2 to be the pressure. For example, for a particular given system, you found out the temperature to be 50 degrees centigrade and pressure to be some value. Given these two quantities, crisp quantities, how do we infer what should be y ?

The crisp input is given – temperature. Now, you find out corresponding membership values here. Corresponding to this crisp input, we get the membership value in rule 1 as $\mu_{A_1^1}$ and for the same crisp input, this rule 2 will provide you $\mu_{A_1^2}$. Now, in the second fuzzy variable, given crisp input, rule 1 will compute $\mu_{A_2^1}$ and for the second one, the second rule, the same crisp input would give this one, which is $\mu_{A_2^2}$. Once we find out these

membership values, what do we do? We graphically see which is minimum between $\mu_{A1 1}$ and $\mu_{A2 1}$. The minimum is $\mu_{A2 1}$. We take that and we shade these areas in consequence. Now, we take the second rule. We find between $\mu_{A1 2}$ and $\mu_{A2 2}$, the minimum is $\mu_{A1 2}$. We take that minimum and shade the area and consequent part of this rule 2. Now graphically, we add these two taking the maximum. First, min and then max. You can easily see that when I overlap this figure over this figure, I get this particular figure. You overlap this second figure on the first figure or first figure on the second figure and take the resultant shaded area. After taking this resultant shaded area.... Once you find this shaded area, the next part is to see what is y given a crisp value. There are many methods, but we will focus in this class or in this course on only one method, that is, center of gravity method(COG). Obviously, if I take this figure and find out what is the center of gravity, it is this value y^* . The crisp output can be obtained using various methods. One of the most common method is the center of gravity method. The resulting crisp output is denoted as y^* in the figure. This is y^* . What we learnt in this is given a crisp input 1 and crisp input 2 and given two fuzzy rules, how do we infer correspondingly a crisp output? Our data is crisp, but we are doing fuzzy computation. Hence, rules are fuzzy. We take this data to the fuzzy rule base and then fuzzify them through fuzzification process. Graphically, we find what is the net shaded area using the max principle. We found out the shaded area for each rule in consequent taking the min principle. Taking the max principle, we found out the resultant area and then, y^* is the center of gravity of these areas.

LECTURE-8

Fuzzy Control system or Fuzzy Inference System:



Categories:

1. Mamdani type and
2. Takagi–Sugeno type (T-S or TSK for short form T. Takagi, M. Sugeno, and K. T. Kang).

Mamdani type fuzzy systems:

These employ fuzzy sets in the consequent part of the rules. This is a Mamdani type fuzzy logic controller. What they do is that the consequent part itself takes the control action; the incremental control action is described in the consequent part of each rule.

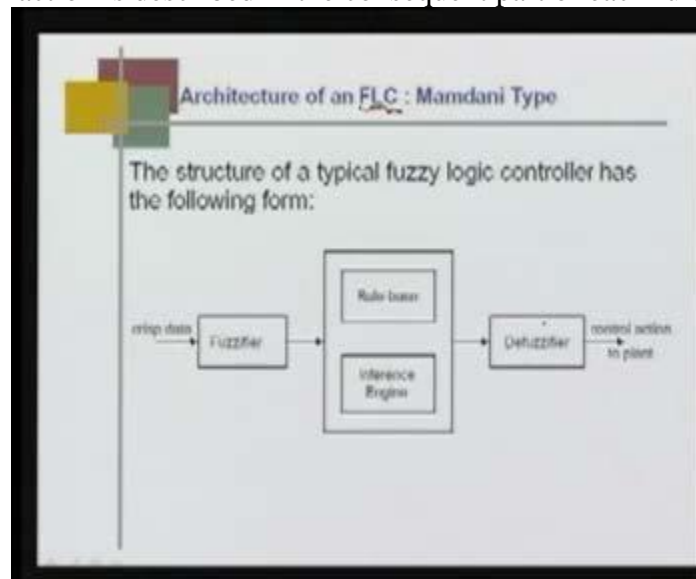


Fig. Architecture of FLC

The actual data that the controller is receiving is crisp data or classical data that has a definite value. That crisp data goes to the fuzzy logic controller and it has these four components that you can see: fuzzifier, rule base, inference engine and defuzzifier.

Fuzzifier. In a fuzzy logic controller, the computation is through linguistic values, not through exact computation. Naturally, the fuzzifier would fuzzify the crisp data. In case of temperature, I can say it is hot, medium-hot, cold, medium-cold, very hot and normal. These are the fuzzifier. That means given a crisp data or the value of temperature say 40 degrees, then I have to now convert to various linguistic values and each linguistic value will be associated with a specific membership function. That is fuzzifier.

Once the data has been fuzzified, then it goes to the rule base and using an inference mechanism... The inference is taking place in fuzzy term, not in classical term and after a fuzzy inference takes place about the decision or about the control action, we place a defuzzifier. What this defuzzifier does is it converts the fuzzy control action to a crisp control action.

In general, what we can say is the principal design parameters of a fuzzy logic controller are the following: fuzzification strategies and interpretation of a fuzzification operator. How do we fuzzify a crisp data? In the database, the discretization or normalization of universe of discourse is done, because we must know the range of data one will encounter in an actual plant. Accordingly, the normalization must be done so that we are taking into account all possible values of data that one may encounter in a physical plant.

Fuzzy partition of the input and output spaces:

If I know the dynamic range of an input to the controller and the input to the plant (input to

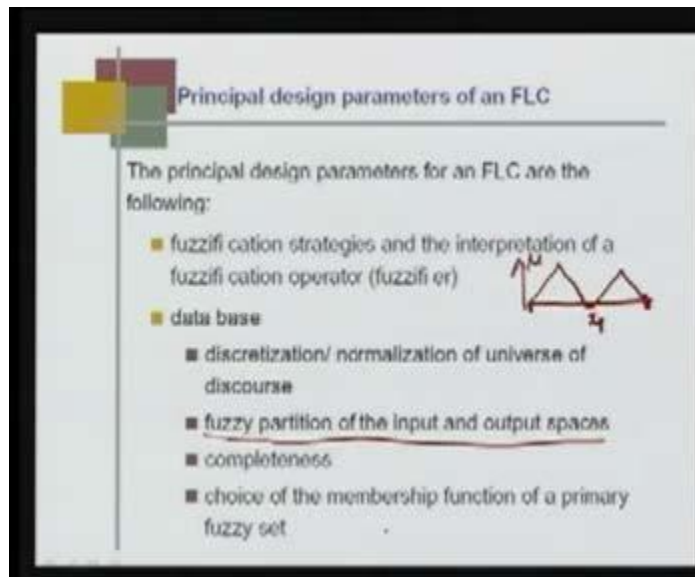


Fig. Parameters to be designed in FLC

the plant is actually output to the controller)... if I know the dynamic range, then in that dynamic range, I must learn how to do fuzzy partition of the input and output space and this fuzzification suits such the process should be complete in the sense.... You see that I am drawing a universe of discourse here. This is the real value for a specific variable x_1 . If I have defined a fuzzy set like this and like this, you can easily see that this part of the data is not associated with any fuzzy membership. This is μ and this is x_1 and unfortunately, this part is not associated with any membership.

This fuzzification process is not complete. That means the entire universe of discourse in a specific domain, wherever there are control systems.... There are various kinds of control systems: process control, robot control, and aircraft control. Every control system is associated with some input data and some output data. All possible input data and all possible output data should be associated with a specific linguistic value as well as a membership function.

Rule base:

Once fuzzification is done, how do we create a rule base? As I said, typically, in the rule base, the two variables that are most important are error and change in error and we also showed why it is so. Rule base. Choice of process state input variables and control variables. You know that if I am implementing a fuzzy state feedback controller, then, a fuzzy state feedback controller u would be minus $K x$. So, x is the states of the system, whereas if I am implementing a fuzzy PID controller, then it will be $u_{old} + K \Delta u_k$. Here, this Δu_k is a function of error and change in error, whereas, in a state feedback controller, this is a common signal r and so, the control action is dependent on state $x_1, x_2,$ and x_n .

Source and derivation of fuzzy control rules.

How do I derive these rules? What is the basis? Types of fuzzy control rules. A type of fuzzy control rule means whether it is a PID controller, fuzzy PID controller or it is a fuzzy state feedback controller. Similarly, completeness of fuzzy control rules means given any crisp data in the domain of input space as well as output space, do I have in my rule base a specific rule associated with this data? If I do not have any rule for this data, then the FLC will fail. That is meaning of completeness of fuzzy control rules.

Fuzzy inference mechanism:

We have already talked about what is fuzzy inference mechanism. Given multiple rules, how do we infer the consequence part? Defuzzification strategies and the interpretation of fuzzification operator. Once the fuzzy inference is done, from the fuzzy inference, how do I get a crisp value or a crisp control action? This is called defuzzification.

This is how we fuzzify a crisp data to fuzzy data or we make them fuzzy, that is, the crisp input for variable x_1 and x_2, \dots . Actually, this is not x_1 and x_2 , but e and Δe are converted to fuzzy sets using triangular membership functions.¹ It is ²not always triangular, it can be anything, but normally in control literature, most of these membership functions are triangular functions.

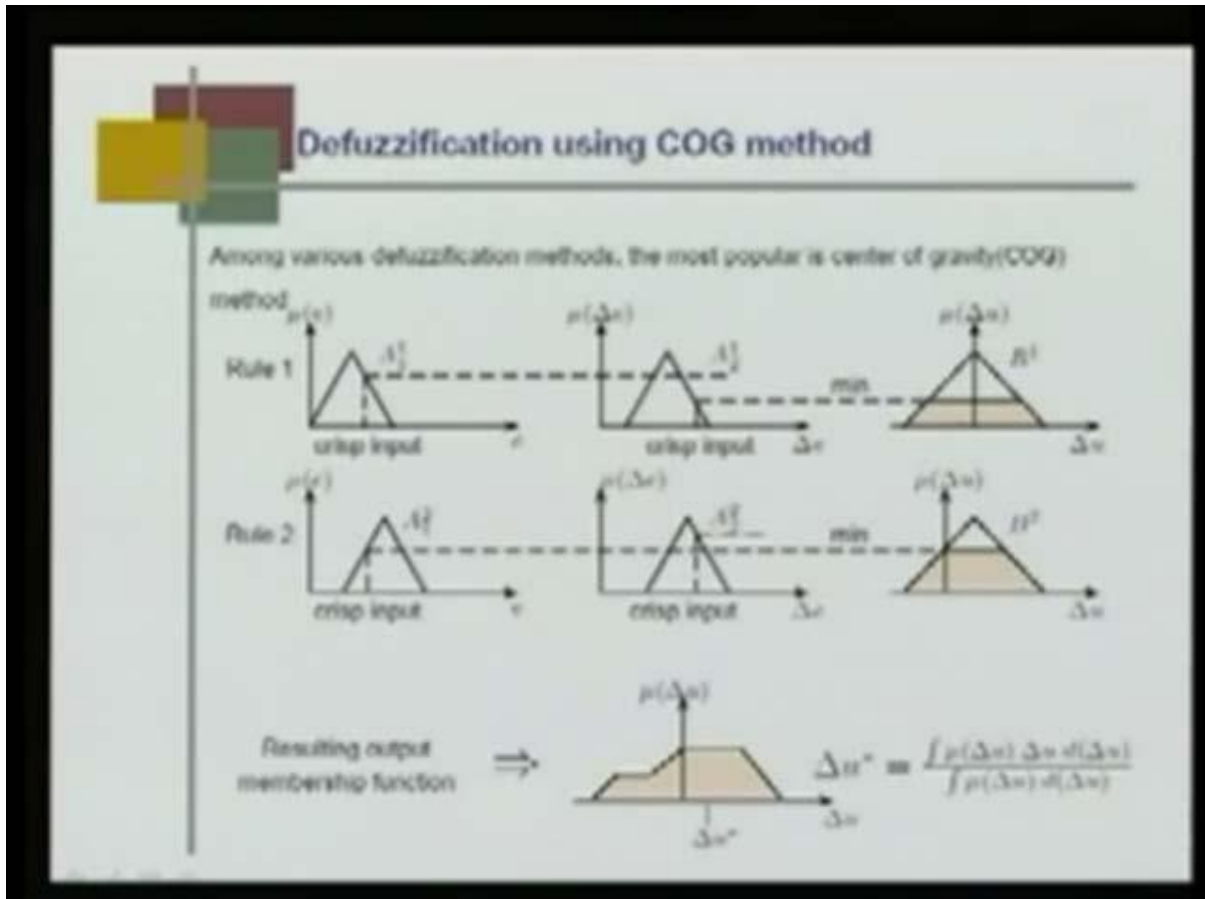


Fig. Defuzzification

Defuzzification. Once I know how to do the fuzzification, defuzzification is explained in the following diagram. You see that among various defuzzification methods, the most popular is center of gravity method. How do I do it? Crisp input is given at any situation, any k th sampling instant and the fuzzy logic controller gets the command signal, gets the actual output of the plant, computes the error, computes the change in error and then, those crisp values are fed into the fuzzification layer. Then, you have the membership function. You pass on those fuzzy data to the rule base and then, for a specific rule base... You see that in rule 1, you see that if you compare the membership μA_1 and μA_2 , μA_2 is the minimum and correspondingly, you shade the zone of action. This is Δu . How much should be the incremental control action? This is my shaded portion or shaded portion of my control action.

Now I take a second one, second rule and there again, I evaluate the fuzzy membership function A_1 and A_2 . You see that the membership function in A_1 is less. Corresponding to that, we shade the incremental control action. Now, you see that if I take the maximum of these two shaded zones, I get this (Refer Slide Time: 38:03), maximum of this. After I get, this is the fuzzy decision, this is the fuzzy incremental control action, but how do I convert this fuzzy incremental control action to a crisp action? That is by the center of gravity method. In the center of gravity method, I integrate $\mu \Delta u$ upon integration of μ . If I integrate this function, I get somewhere here to be the center of gravity. Δu^* is this value, which is graphically shown here. We discussed about Mamdani type fuzzy logic controller.

LECTURE-9

Takagi–Sugeno fuzzy systems:

The number of rules required by the Mamdani model are reduced here. They employ function of the input fuzzy linguistic variable as the consequent of the rules. What happens here is

that a fuzzy dynamic model is expressed in the form of local rules. Local rule means if there are two variables $x_1 = a$ and $x_2 = b$, then the plant dynamics can be represented either as a linear dynamical system or a nonlinear dynamical system, as a known dynamical system.

TSK Fuzzy Rule

- If x is A and y is B then $z = f(x,y)$
- Where A and B are fuzzy sets in the antecedent, and
- $Z = f(x,y)$ is a crisp function in the consequence.
- Usually $f(x,y)$ is a polynomial in the input variables x and y , but it can be any function describe the output of the model within the fuzzy region specified by the antecedence of the rule.

First order TSK Fuzzy Model

- $f(x,y)$ is a first order polynomial

Example: a two-input one-output TSK

IF x is A_j and y is B_k then $z_i = px + qy + r$

The degree the input matches i th rule is typically computed using min

operator: $w_i = \min(\mu_{A_j}(x), \mu_{B_k}(y))$

- Each rule has a crisp output
- Overall output is obtained via weighted average (reduce computation time of defuzzification required in a Mamdani model)

$$z = \frac{\sum_i w_i z_i}{\sum_i w_i}$$

To further reduce computation, weighted sum may be used, I.e.

$$z = \sum_i w_i z_i$$

Example #1: Single-input

- A single-input TSK fuzzy model can be expressed as
- If X is small then $Y = 0.1 X + 6.4$.
- If X is medium then $Y = -0.5X + 4$.
- If X is large then $Y = X - 2$.

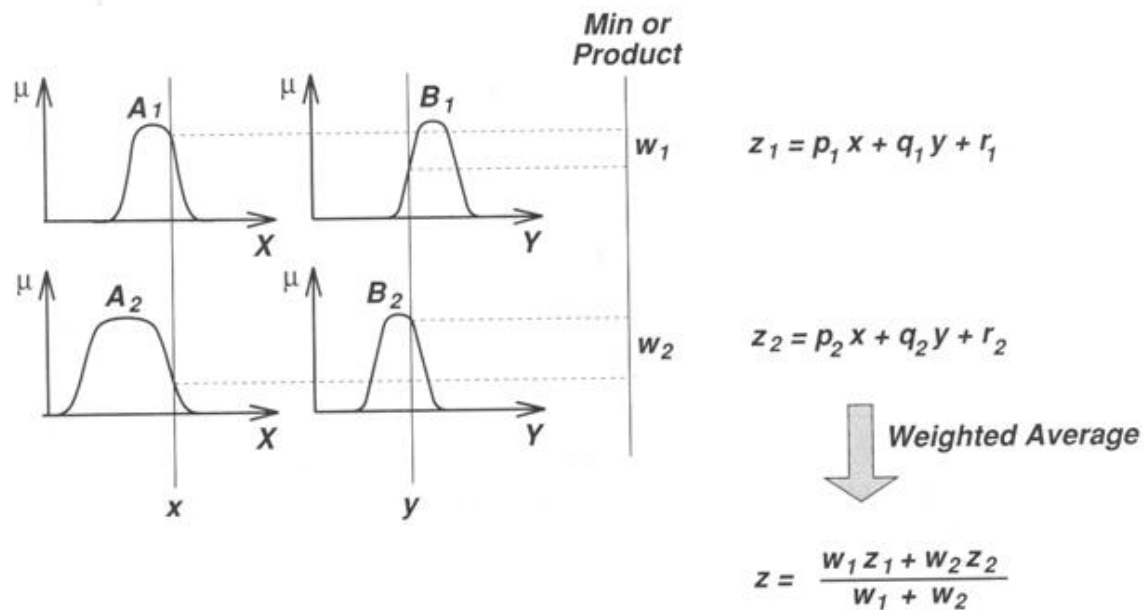


Fig. Rules in first order TSK fuzzy model

Example #2 : Two-input

- A two-input TSK fuzzy model with 4 rules can be expressed as
 - If X is small and Y is small then $Z = -X + Y + 1$.
 - If X is small and Y is large then $Z = -Y + 3$.
 - If X is large and Y is small then $Z = -X + 3$.
 - If X is large and Y is large then $Z = X + Y + 2$.

Zero-order TSK Fuzzy Model

- When f is constant, we have a zero-order TSK fuzzy model (a special case of the Mamdani fuzzy inference system which each rule's consequent is specified by a fuzzy singleton or a pre defuzzified consequent)
- Minimum computation time Overall output via either weighted average or weighted sum is always crisp
- Without the time-consuming defuzzification operation, the TSK (Sugeno) fuzzy model is by far the most popular candidate for sample data-based fuzzy modeling.

A general Takagi–Sugeno model of N rules for any physical plant, a general T–S model of N rules is given by Rule _{i} . This is the i th rule. If x_1 is a specific fuzzy set M_{1i} and x_2 is another specific fuzzy set M_{2i} and so on until x_n is another fuzzy set M_{ni} , then the system dynamics locally is described as x_k plus 1 is A_i x_k plus B_i u_k , where i equal to 1, 2 until N , because there are N rules.

Advantages over Mamdani model:

1. Less computation
2. Less time consuming
3. Simple
4. Mostly used for sample data based fuzzy modelling

Tsukamoto Fuzzy Models:

- The consequent of each fuzzy if-then rule is represented by a fuzzy set with **monotonical MF**
- As a result, the inferred output of each rule is defined as a crisp value induced by the rules' firing strength.
- The overall output is taken as the weighted average of each rule's output.

Example: Single-input Tsukamoto fuzzy model

- A single-input Tsukamoto fuzzy model can be expressed as
 - If X is small then Y is C1
 - If X is medium then Y is C2
 - If X is large then Y is C3

Example: Single-input Tsukamoto fuzzy model

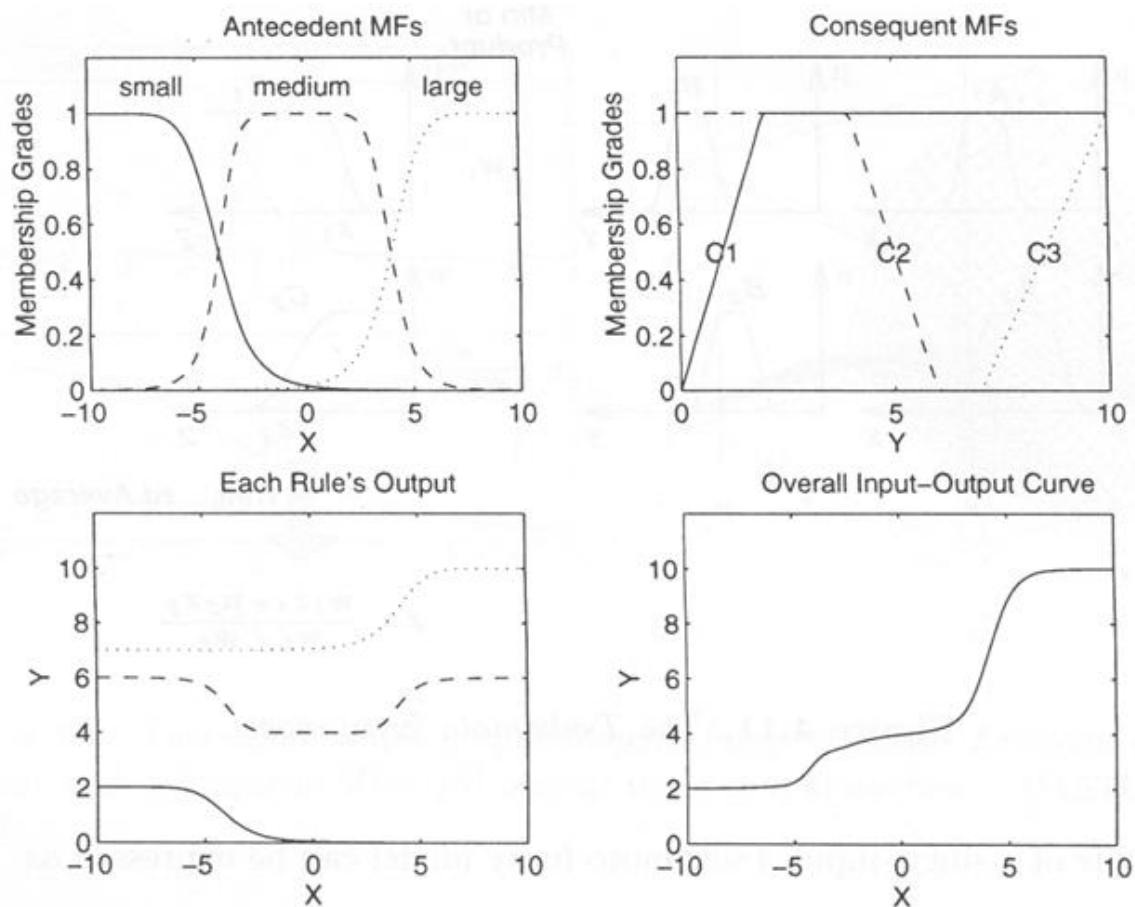


Fig. Output of Tsukamoto fuzzy model

Input Space Partitioning:

The antecedent of a fuzzy rule defines a local fuzzy region, while the consequent describes the behavior within the region via various constituents. The consequent constituents are different MF or equation or constant depending on the fuzzy model. But, antecedents of fuzzy rules can be formed by partitioning the input space.

3 types-

Grid partition: Often chosen method. Applicable to small no. of input variables and MFs i.e. curse of dimensionality .

Tree partition: Each region is specified uniquely along a corresponding decision tree. Exponential increase of no. of rules is reduced. More MFs are needed. Orthogonality holds roughly. Used in CART algorithm.

Scatter partition:

Portioning is scattered. Orthogonality doesn't hold. The portioned regions are non uniform.No. of rules is reduced, but overall mapping from consequent of each rule out put is difficult to estimate.

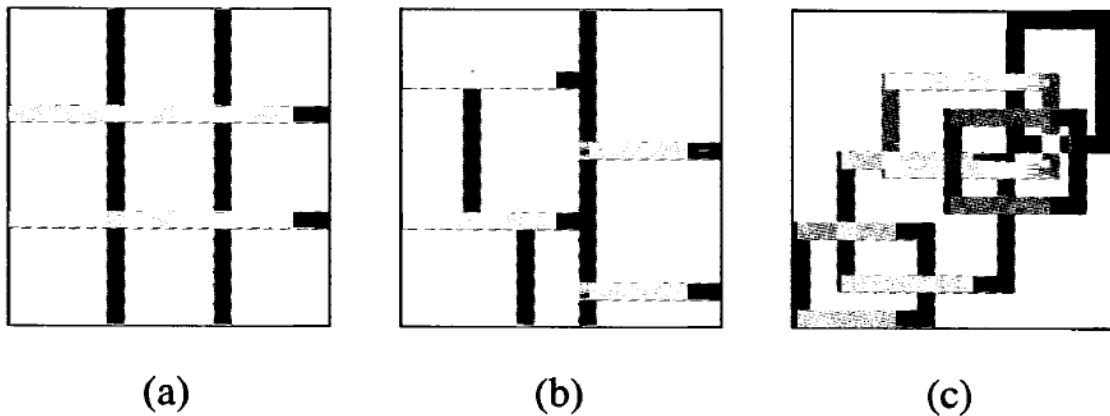


Fig. (a) Grid partition (b) Tree partition (c) Scatter partition

If certain transformation of the input is done, more flexible boundaries and partition will be obtained.

LECTURE-10

Defuzzification methods in detail:

strategy employed in Mamdani's fuzzy logic control
Defuzzification

It refers to a way a crisp value is extracted from a fuzzy set as a representative value

5 methods of defuzzification:

(i) Centroid of area Z_{COA} (most widely used)

$$Z_{COA} = \frac{\int_Z \mu_A(z) \cdot z \, dz}{\int_Z \mu_A(z) \, dz}$$

where, $\mu_A(z)$ is the aggregated o/p MF.

(ii) Bisector of area Z_{BOA}

Z_{BOA} satisfies

$$\int_a^{Z_{BOA}} \mu_A(z) \cdot dz = \int_{Z_{BOA}}^b \mu_A(z) \cdot dz$$

where $a = \min \{z \mid z \in Z\}$ $b = \max \{z \mid z \in Z\}$

i.e. Z_{BOA} line partitions the region betⁿ $z=a$ $z=b$, $y=0$ & $y=\mu_A(z)$ into 2 regions with same area.
(y coordinate not var. y)

(iii) Mean of maximum Z_{MOM}

Z_{MOM} is the average of the maximizing z at which the MF reach a max^m μ^* .

$$Z_{MOM} = \frac{\int_{Z'} z \, dz}{\int_{Z'} dz}$$

where $Z' = \{z \mid \mu_A(z) = \mu^*\}$

In particular, if $u_A(z)$ has a single \max^m at $z = z^*$, then $z_{\text{mom}} = z^*$.

Moreover if $u_A(z)$ reaches its \max^m whenever $z \in [z_{\text{left}}, z_{\text{right}}]$ the $z_{\text{mom}} = \frac{(z_{\text{left}} + z_{\text{right}})}{2}$.

(iv) Smallest of $\max^m z_{\text{mom}}$

z_{mom} is the minimum (in terms of magnitude) of the maximizing z .

(v) Largest of $\max^m z_{\text{mom}}$

z_{mom} is the maximum (in terms of magnitude) of the maximizing z .

Fuzzy Expert system

Expert systems are primarily built for the purpose of making the experience, understanding & problem solving capabilities of the expert in a particular subject area available to the non expert in this area.

The kernel of any expert system consists of

- (i) A knowledge base (also called a long term memory)
- (ii) A database (or a short term memory) or a blackboard interface
- (iii) An inference engine

(i) Knowledge base

Contains general knowledge pertaining to the problem domain.

In fuzzy expert systems, the knowledge is usually represented by a set of fuzzy production rules which connect antecedents with consequences, premises with conclusions or conditions with action:

Most common form

If A, then B, where A & B are fuzzy sets.

(ii) Data base

The purpose of the data base is to store data for each specific task of the expert system. Such as parameters of the problem or other relevant facts.

The data may be obtained from a dialog betⁿ the expert system & the user. Other data may be obtained by the inference of the expert system.

(iii) Inference Engine

The inference engine of a fuzzy expert system operates on a series of ~~data~~ production rules and makes fuzzy inferences.

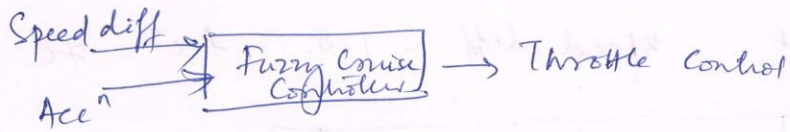
There exist 2 approaches to evaluating relevant production rules.

(i) Data driven - exemplified by the generalised modus ponens. In this case available data are supplied to the expert system, which then uses them to evaluate relevant production rules & draw all possible conclusions.

(ii) Goal driven - exemplified by the generalised modus tollens form of logical inference.

Goog root's Fuzzy Cruise Control.

To maintain a vehicle at desired speed



Fuzzy Rule base (As per Expert's experience)

Rule 1 If (speed diff. is NL) & accⁿ ZE then throttle

Rule 2 ZE & NL then PL

Rule 3 NP & ZE then PM

Rule 4 NS & PS then PS

Rule 5 PS & NS then NS

Rule 6 PL & ZE then NL

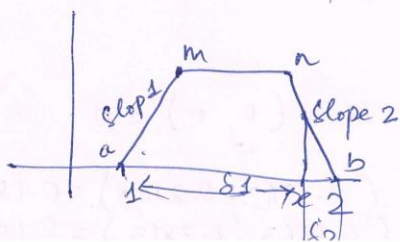
Rule 7 ZE & NS then PS

Rule 8 ZE & NM then PM

NL → Negative large S → Small
 PM → Positive medium M → medium
 NS → Neg. L

Membership → NL NM NS ZE PS PM PL

For fuzzification of I/P & to compute the membership of antecedents



$$\delta_1 = x - pt_1$$

$$\delta_2 = pt_2 - x$$

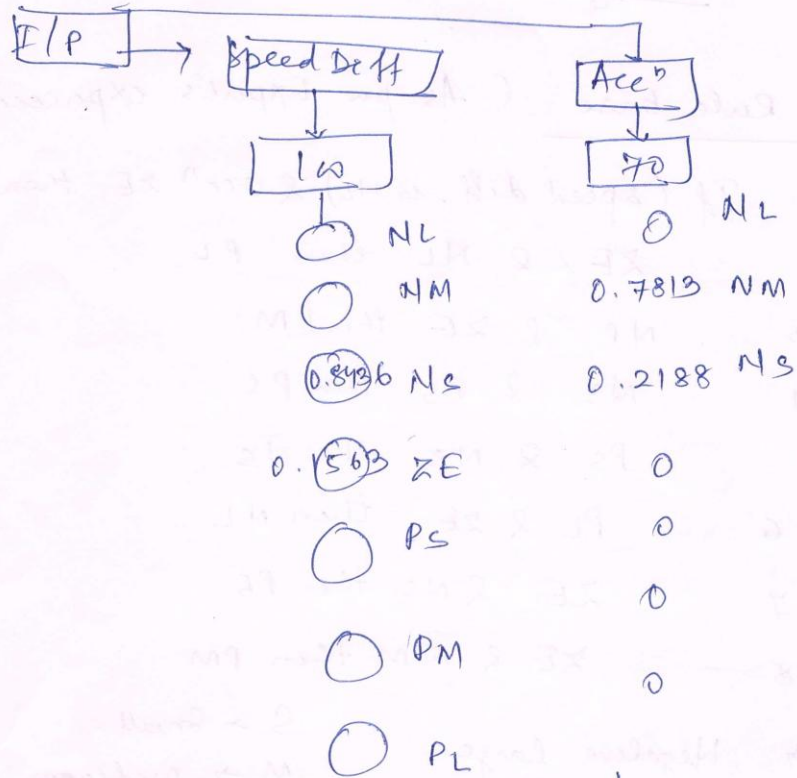
If $\delta_1 \leq 0$ or $\delta_2 \leq 0$ $\mu = 0$

Use
$$\mu = \min \begin{bmatrix} \delta_1 \cdot \text{slope 1} \\ \delta_2 \cdot \text{slope 2} \\ \text{max} \end{bmatrix}$$

Slope known = 1

membership of x is computed in all 7 members $\mu_1, \mu_2, \dots, \mu_7$ recorded in data str.

Eg Let Speed diff = 100 Accⁿ = 70



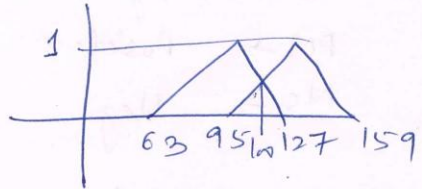
For triangular

for NS $\delta_1 = 100 - 63 = 37$
 $\delta_2 = 127 - 100 = 27$

slope 1 = $\frac{1}{32} = 0.03125$

slope 2 = $\frac{1}{32}$

$$\mu_{NS}(x) = \min \left(\begin{matrix} 37 \times 0.03125 \\ 37 \times 0.03125 \\ 1 \end{matrix} \right) = 0.8438$$



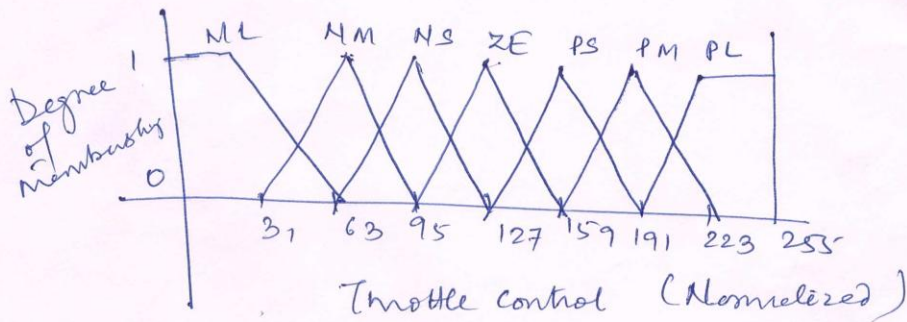
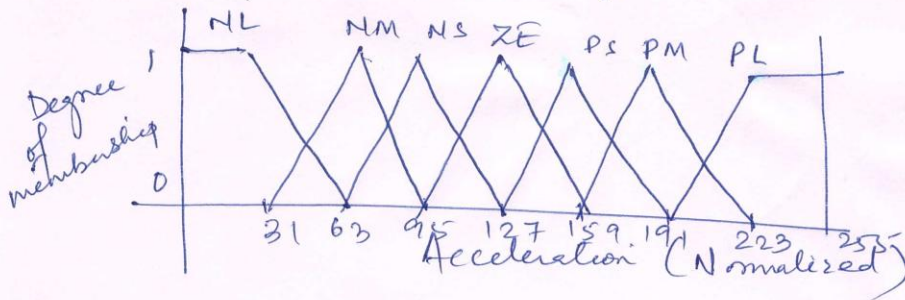
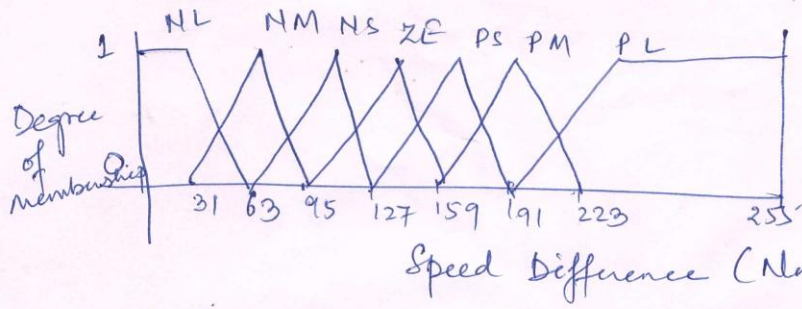
Rule strength compⁿ

Rule 1 $\min(ZE, NL) = \min(0, 0) = 0$

Rule 7 $\min(ZE, NS) = \min(0.1563, 0.2188) = 0.156$

Greg's Throttle control

Fig of membership
Continued.
page 2



Q. What are the steps of fuzzy reasoning?

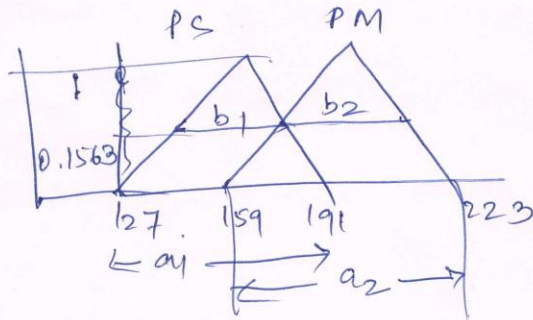
Q. What are the steps of fuzzy modelling?

Fuzzy O/P Or

As more than one rule qualify

Defuzz

O/P



$$\text{Centroid} = \frac{127 + 223}{2} =$$

for P1 Centroid = 159

for P2 = 191

$$\text{Area} = \frac{1}{2} h (a_1 + b_1)$$
$$= \frac{1}{2} (0.1563) (64 + \dots)$$
$$= 9.99$$

= 9.99

$$\text{Weighted Avg CG} = \frac{9.99 \times 159 + 9.99 \times 191}{9.99 + 9.99}$$

$$= \underline{175} \rightarrow \text{tooth like sk}$$

References:

1. Chapter 1 to 4 J.S.R.Jang, C.T.Sun and E.Mizutani, "*Neuro-Fuzzy and Soft Computing*", PHI, 2004, Pearson Education 2004.
2. Chapter 6 & 7 S. Rajasekaran & GA Vijayalakshmi Pai "Neural Networks, Fuzzy Logic, and Genetic Algorithms synthesis and application", PHI
3. Timothy J.Ross, "*Fuzzy Logic with Engineering Applications*", McGraw-Hill, International Editions, Electrical Engineering Series, Singapore, 1997.
4. Internet sources.

MODULE-II (10 HOURS)

Neural networks: Single layer networks, Perceptrons: Adaline, Mutilayer Perceptrons Supervised Learning, Back-propagation, LM Method, Radial Basis Function Networks, Unsupervised Learning Neural Networks, Competitive Learning Networks, Kohonen Self-Organizing Networks, Learning Vector Quantization, Hebbian Learning. Recurrent neural networks,. Adaptive neuro-fuzzy information; systems (ANFIS), Hybrid Learning Algorithm, Applications to control and pattern recognition.

LECTURE-1

NEURAL NETWORK INTRODUCTION:

What is a neuron? A neuron is the basic processing unit in a neural network sitting on our brain. It consists of

1. Nucleus-
2. Axon- Output node
3. Dendrites-Input node
4. Synaptic junction

The dynamics of this synaptic junction is complex. We can see the signal inputs from the action of a neuron and through synaptic junction an output is actuated which is carried over through dendrites to another neuron. Here, these are the neurotransmitters. We learned from our experience that these synaptic junctions are either reinforced or in the sense they behave in such a way that the output of synaptic junction may excite a neuron or inhibit the neuron. This reinforcement of the synaptic weight is a concept that has been taken to artificial neural model.

The objective is to create artificial machine and this artificial neural networks are motivated by certain features that are observed in human brain, like as we said earlier, parallel distributed information processing.

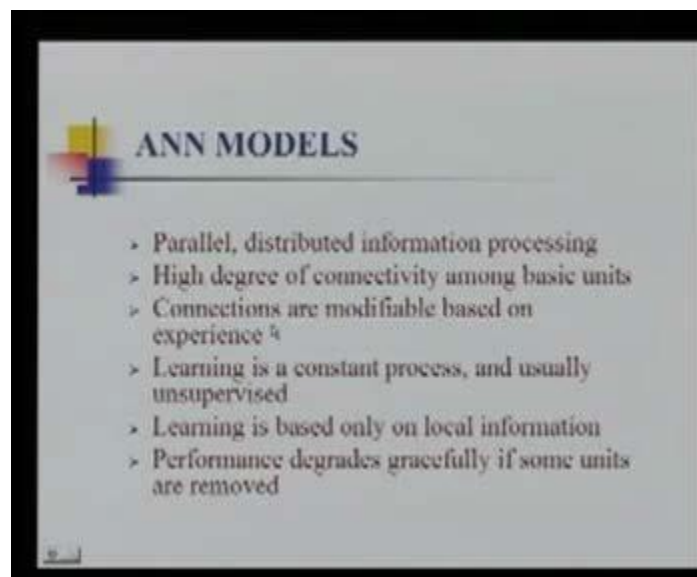


Fig. ANN model

Artificial neural networks are among the most powerful learning models. They have the versatility to approximate a wide range of complex functions representing multi-dimensional input-output maps. Neural networks also have inherent adaptability, and can perform robustly even in noisy environments.

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected simple processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well. ANNs can process information at a great speed owing to their highly massive parallelism.

A trained neural network can be thought of as an "expert" in the category of information it has been given to analyse. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.

Advantages of ANN:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

Table- Difference between the brain and a digital Computer

| Property | Computer | Brain |
|-----------------|-------------------------------|-----------------------------|
| Shape | 2d Sheets of inorganic matter | 3d volume of organic matter |
| Power | Powered by DC mains | Powered by ATP |
| Signal | Digital | pulsed |
| Clock | Centralized clock | No centralized clock |
| Clock speed | Gigahertz | 100s of Hz |
| Fault tolerance | Highly fault-sensitive | Very fault-tolerant |
| Performance | By programming | By learning |

Differences human brain & ANN:

1. Computer has such fast speed of GHz, a traditional computer, however, when it comes to certain processing like pattern recognition and language understanding, the brain is very fast.
2. Intelligence and self-awareness, are absent in an artificial machine.

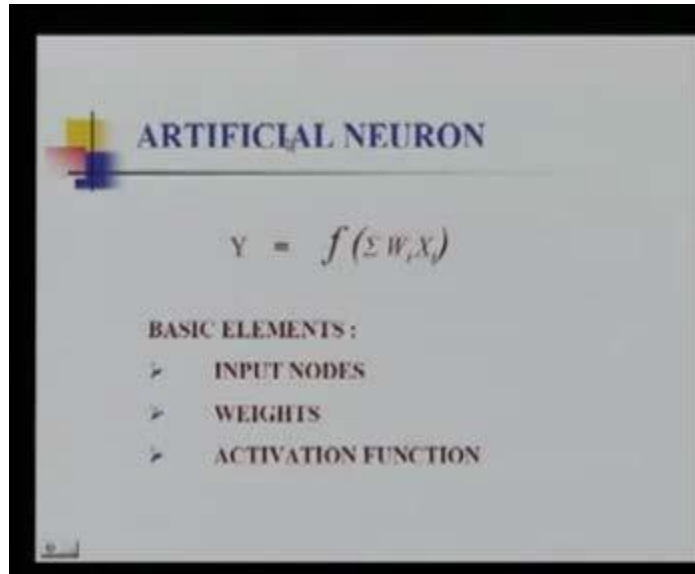


Fig. An artificial neuron

An Artificial Neuron:

Basic computational unit in an artificial neural network is neuron. Obviously, it has to be an artificial neuron.

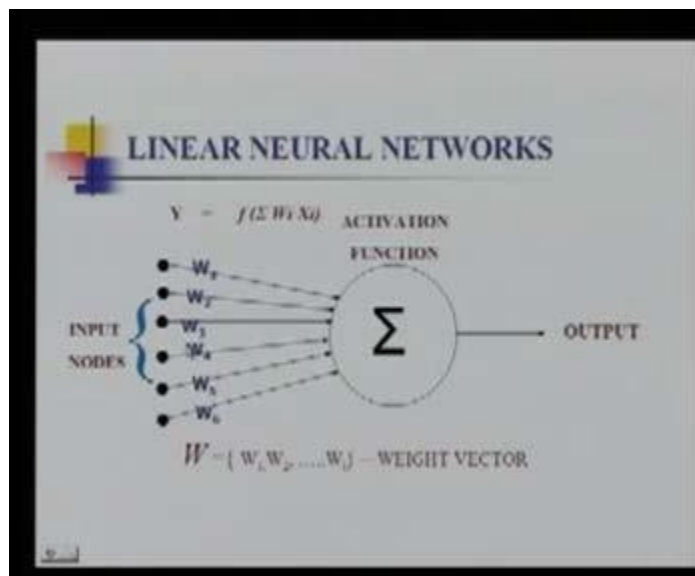


Fig. An artificial Neuron in linear NN

This artificial neuron has three basic elements:

1. Nodes,
2. Weights and
3. Activation function.

Between input nodes and output nodes, there are synaptic weights w_1, w_2, w_3, w_4, w_5 and w_6 . There can be as many weights and these weights are multiplied with the signal as they reach the output unit, where the output is simply sum of the signal multiplied with the weights and then this output goes to an activation function f .

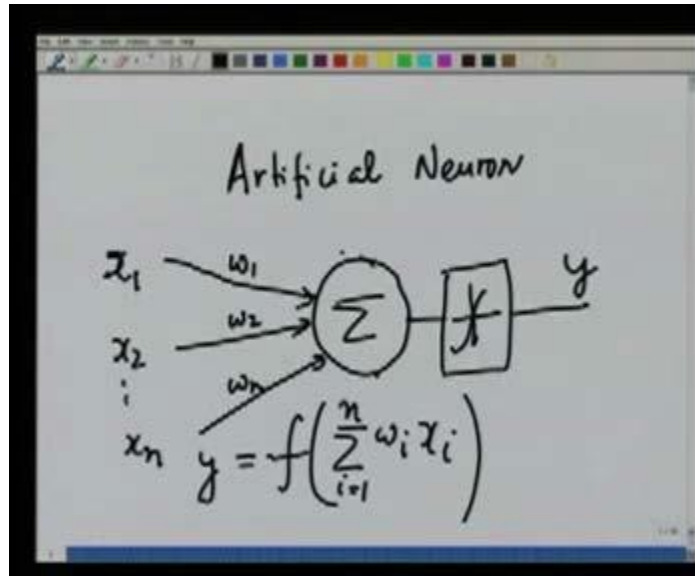


Fig. Basic processing unit- the neuron

In a simple neuron, if input signals be x_1, x_2, x_n with weights w_1, w_2 and w_n . The weighted sum will activate this total output by an activation function f . That is your output. What you are seeing is actually a nonlinear map from input vector x_1 to output y . A single neuron has single output but multiple inputs. Inputs are multiple for a single neuron and the output is unique, y and this output y and the input bear a nonlinear relationship, by f . Neural networks can be built using this single neuron. We can use the single neuron and build neural networks.

Analogy to brain:

Artificial Neural Network (ANN) is a system which performs information processing. An ANN resembles or it can be considered as a generalization of mathematical model of human brain assuming that

1. Information processing occurs at many simple elements called neurons.
2. Signals are passed between neurons over connection links.
3. Each connection link has an associated weight, which in a typical neural net multiplies the signal transmitted.

ANN is built with basic units called *neurons* which greatly resemble the neurons of human brain. A neural net consists of a large number of simple processing elements called neurons. Each neuron applies an activation function to its net input to determine its output signal. Every neuron is connected to other neurons by means of directed communication links, each with an associated weight. Each neuron has an internal state called its *activation level*, which is a function of the inputs it has received. As and when the neuron receives the signal, it gets added up and when the cumulative signal reaches the activation level the neuron sends an output. Till then it keeps receiving the input. So activation level can be considered as a threshold value for us to understand.

In general, a neural network is characterized by

1. *Pattern of connections* between the neurons called its architecture
2. Method of *determining the weights* on the connections called its training or learning algorithm
3. Its *internal state* called its Activation function.

The arrangement of neurons into layers and the connection patterns within and between layers is called the net *architecture*. A neural net in which the signals flow from the input units to the output units in a forward direction is called *feed forward nets*.

Interconnected competitive net in which there are closed loop signal paths from a unit back to it is called a *recurrent network*. In addition to architecture, the method of setting the values of the weights called *training* is an important characteristic of neural nets. Based on the training methodology used neural nets can be distinguished into *supervised* or *unsupervised* neural nets. For a neural net with supervised training, the training is accomplished by presenting a sequence of training vectors or patterns each with an associated target output vector. The weights are then adjusted according to a learning algorithm. For neural nets with unsupervised training, a sequence of input vectors is provided, but no target vectors are specified. The net modifies the weights so that the most similar input vectors are assigned to the same output unit. The neural net will produce a representative vector for each cluster formed. Unsupervised learning is also used for other tasks, in addition to clustering.

LECTURE-2

Activation functions:

Table 3.1 Typical nonlinear activation operators

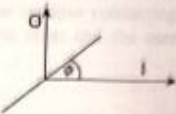
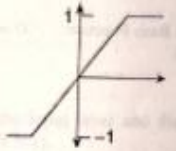
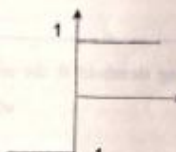
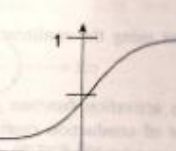
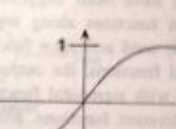
| Type | Equation | Functional form |
|--------------------|--|---|
| Linear | $O = gt$ $g = \tan \phi$ |  |
| Piecewise Linear | $O = \begin{cases} 1 & \text{if } mt > 1 \\ gt & \text{if } mt < 1 \\ -1 & \text{if } mt < -1 \end{cases}$ |  |
| Hard Limiter | $O = \text{sgn } [I]$ |  |
| Unipolar Sigmoidal | $O = \frac{1}{1 + \exp(-\lambda I)}$ |  |
| Bipolar Sigmoidal | $O = \tanh [\lambda I]$ |  |

Table 3.1 Typical nonlinear activation operators (cont.)

| Type | Equation | Functional form |
|-----------------------------|---|-----------------|
| Unipolar Multimodal | $O = \frac{1}{2} \left[1 + \frac{1}{M} \sum_{m=1}^M \tanh (g^m (I - W_0^m)) \right]$ | |
| Radial Basis Function (RBF) | $I = \frac{-\sum_{i=1}^N (W_i(t) - X_i(t))^2}{2\sigma^2}$ | |

Architecture:

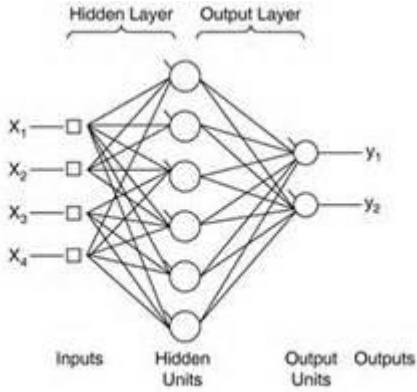


Fig. Architecture of multilayer Neural network

Artificial neural networks are represented by a set of nodes, often arranged in layers, and a set of weighted directed links connecting them. The nodes are equivalent to neurons, while the links denote synapses. The nodes are the information processing units and the links acts as communicating media.

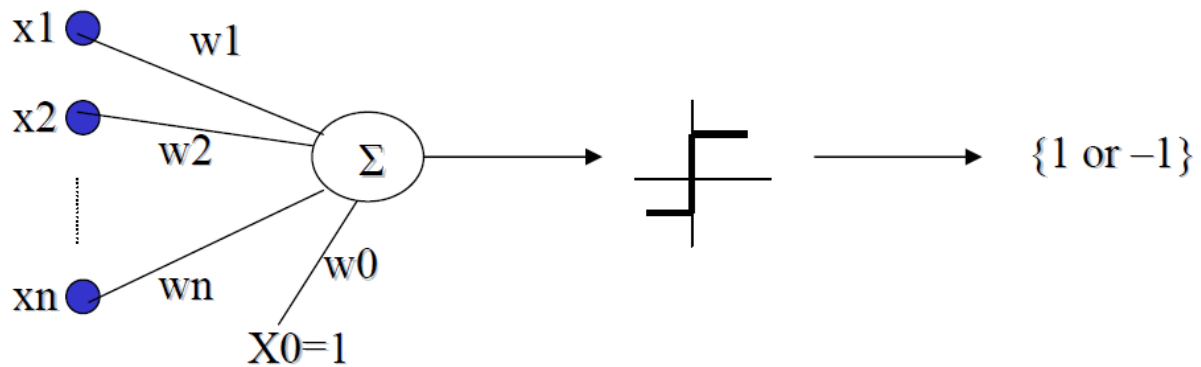
A neural network may have different layers of neurons like

1. input layer,
2. hidden layer,
3. output layer.

The input layer receives input data from the user and propagates a signal to the next layer called the hidden layer. While doing so it multiplies the weight along with the input signal. The hidden layer is a middle layer which lies between the input and the output layers. The hidden layer with non linear activation function increases the ability of the neural network to solve many problems than the case without the hidden layer. The output layer sends its calculated output to the user from which decision can be made. Neural nets can also be classified based on the above stated properties.

There are a wide variety of networks depending on the nature of information processing carried out at individual nodes, the topology of the links, and the algorithm for adaptation of link weights. Some of the popular among them include:

Perceptron: Definition: It's a step function based on a linear combination of real-valued inputs. If the combination is above a threshold it outputs a 1, otherwise it outputs a -1. This consists of a single neuron with multiple inputs and a single output. It has restricted information processing capability. The information processing is done through a transfer function which is either linear or non-linear.



$$O(x_1, x_2, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

Fig. A perceptron

A perceptron can learn only examples that are called “linearly separable”. These are examples that can be perfectly separated by a hyperplane.

Perceptrons can learn many boolean functions: AND, OR, NAND, NOR, but not XOR

However, every boolean function can be represented with a perceptron network that has two levels of depth or more.

The weights of a perceptron implementing the AND function is shown below.

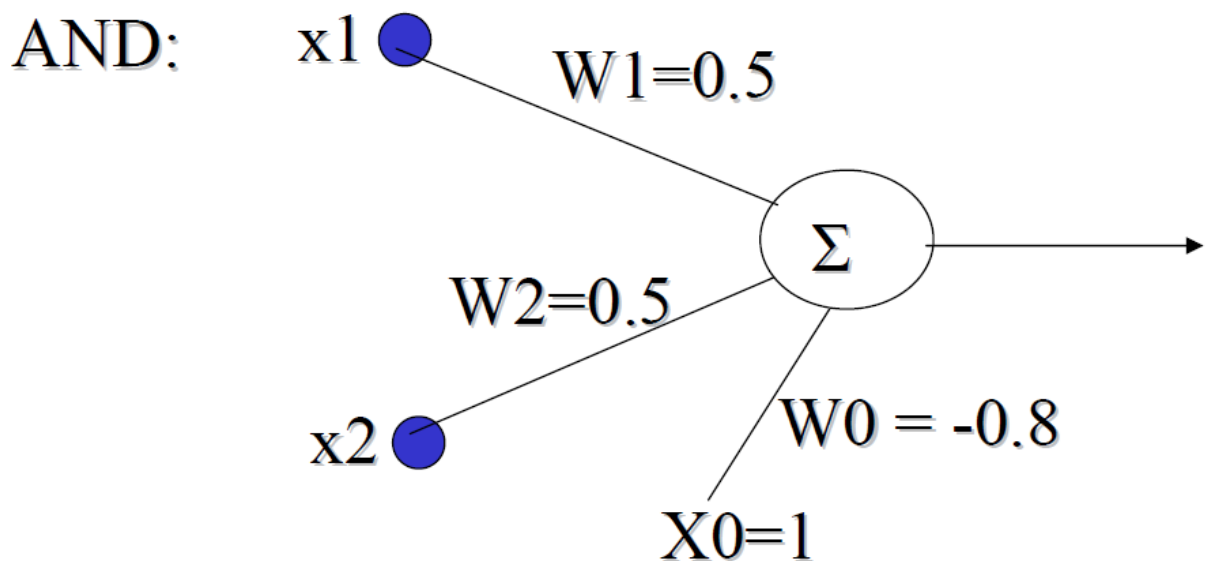


Fig. AND operation on inputs by a single perceptron

Multi-layered Perceptron (MLP): It has a layered architecture consisting of input, hidden and output layers. Each layer consists of a number of perceptrons. The output of each layer is

transmitted to the input of nodes in other layers through weighted links. Usually, this transmission is done only to nodes of the next layer, leading to what are known as feed forward networks. MLPs were proposed to extend the limited information processing capabilities of simple perceptrons, and are highly versatile in terms of their approximation ability. Training or weight adaptation is done in MLPs using supervised backpropagation learning.

Adding a hidden layer:

The perceptron, which has no hidden layers, can classify only linearly separable patterns.

The MLP, with at least 1 hidden layer can classify *any* linearly non-separable classes also.

An MLP can approximate any continuous multivariate function to any degree of accuracy, provided there are sufficiently many hidden neurons (Cybenko, 1988; Hornik et al, 1989). A more precise formulation is given below.

A serious limitation disappears suddenly by adding a single hidden layer.

It can easily be shown that the XOR problem which was not solvable by a Perceptron can be solved by a MLP with a single hidden layer containing two neurons.

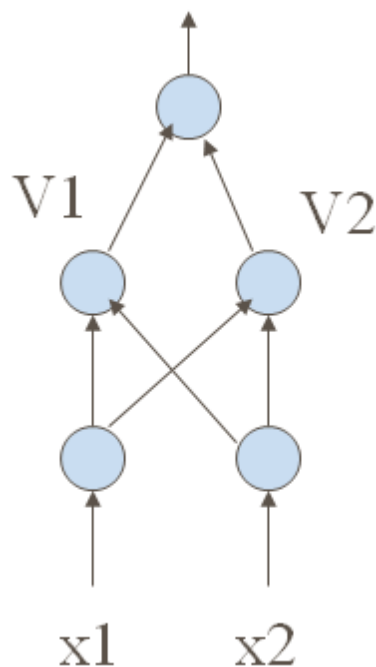


Figure 6.2.1.1: MLP for solving Xor

Recurrent Neural Networks: RNN topology involves backward links from output to the input and hidden layers. The notion of time is encoded in the RNN information processing scheme. They are thus used in applications like speech processing where inputs are time sequences data.

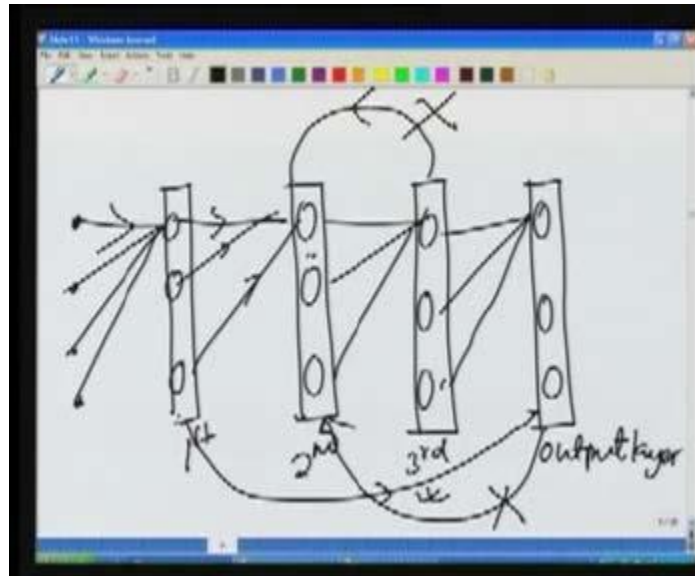


Fig. Multilayer feed back network (Recurrent Neural Network)

Self-Organizing Maps: SOMs or Kohonen networks have a grid topology, with unequal grid weights. The topology of the grid provides a low dimensional visualization of the data distribution. These are thus used in applications which typically involve organization and human browsing of a large volume of data. Learning is performed using a winner take all strategy in an unsupervised mode. It is described in detail later.

Single layer Network:

A neural net with only input layer and output layer is called single layer neural network. A neural network with input layer, one or more hidden layers and an output layer is called a multilayer neural network. A single layer network has limited capabilities when compared to the multilayer neural networks.

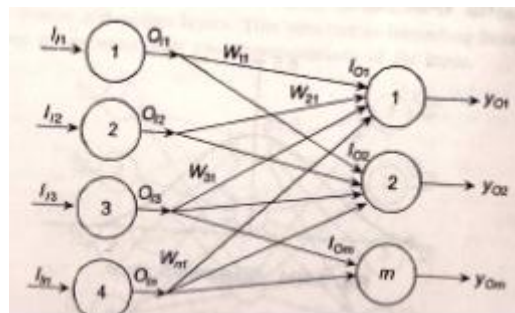


Fig. Single Layer feed forward Neural Network

LECTURE-3

Steps in developing NN:

1.1. Network formation

Neural network consists of an input layer, an output layer and a hidden layer. While a neural network is constructed, the number of neurons in each layer has to be fixed. The input layer will have neurons whose number will be equal to the number of features extracted. The number of neurons in the output layer will be equal to the number of pattern classes. The

number of neurons in the hidden layer is decided by trial and error basis. With a minimum number of neurons in the hidden layer, the neural network will be constructed and the convergence will be checked for. Then the error will be noted. The number of neurons for which the error is minimum, can be taken and will be checked for reduced error criterion.

1.2. Data preprocessing and normalization

Data selection and pre processing can be a demanding and intricate task. Neural net is as good as the input data used to train it. If important data inputs are missing, then the effect on the neural network's performance can be significant. The most appropriate raw input data must be preprocessed. Otherwise the neural network will not produce accurate results. Transformation and normalization are two widely used preprocessing methods. Transformation involves manipulating raw data inputs to create a single input to a net, while normalization is a transformation performed on a single data input to distribute the data evenly and scale it into an acceptable range for the network. Knowledge of the domain is important in choosing preprocessing methods to highlight the features in the data, which can increase the ability of the network to learn the association between inputs and outputs. Data normalization is the final preprocessing step. In normalizing data, the goal is to ensure that the statistical distribution of values should be scaled to match the range of the input neurons. The simplest method of normalization can be done using the formula

$X \text{ normalized} = (X - \mu) / \sigma$ where μ and σ are the mean and standard deviation of the input data.

Perceptron Learning

Learning a perceptron means finding the right values for W. The hypothesis space of a perceptron is the space of all weight vectors.

The perceptron learning algorithm can be stated as below.

1. Assign random values to the weight vector
2. Apply the *weight update rule* to every training example
3. Are all training examples correctly classified?
 - a. Yes. Quit
 - b. No. Go back to Step 2.

There are two popular weight update rules.

- i) The perceptron rule, and
- ii) Delta rule

The Perceptron Rule

For a new training example $X = (x_1, x_2, \dots, x_n)$, update each weight according to this rule:

$$w_i = w_i + \Delta w_i$$

Where $\Delta w_i = \eta (t - o) x_i$

t: target output

o: output generated by the perceptron

η : constant called the learning rate (e.g., 0.1)

Comments about the perceptron training rule:

Example means training data.

- If the example is correctly classified the term $(t - o)$ equals zero, and no update on the weight is necessary.
- If the perceptron outputs -1 and the real answer is 1 , the weight is increased.
- If the perceptron outputs a 1 and the real answer is -1 , the weight is decreased.
- Provided the examples are linearly separable and a small value for η is used, the rule is proved to classify all training examples correctly (i.e, is consistent with the training data).

The Delta Rule

What happens if the examples are not linearly separable?

To address this situation we try to approximate the real concept using the delta rule.

The key idea is to use a *gradient descent search*. We will try to minimize the following error:

$$E = \frac{1}{2} \sum_i (t_i - o_i)^2$$

where the sum goes over all training examples. Here o_i is the inner product WX and not $\text{sgn}(WX)$ as with the perceptron rule. The idea is to find a minimum in the space of weights and the error function E .

The delta rule is as follows:

For a new training example $X = (x_1, x_2, \dots, x_n)$, update each weight according to this rule:

$$w_i = w_i + \Delta w_i$$

Where $\Delta w_i = -\eta E'(W)/w_i$

η : learning rate (e.g., 0.1)

It is easy to see that

$$E'(W)/w_i = \sum_i (t_i - o_i) (-x_i)$$

So that gives us the following equation:

$$w_i = \eta \sum_i (t_i - o_i) x_i$$

There are two differences between the perceptron and the delta rule. The perceptron is based on an output from a step function, whereas the delta rule uses the linear combination of inputs directly. The perceptron is guaranteed to converge to a consistent hypothesis assuming the data is linearly separable. The delta rule converges in the limit but it does not need the condition of linearly separable data.

There are two main **difficulties with the gradient descent method**:

1. Convergence to a minimum may take a long time.

2. There is no guarantee we will find the global minimum.

These are handled by using momentum terms and random perturbations to the weight vectors.

LECTURE-4

ADALINE & MADALINE:

The Adaline networks (ADAPtive LINEar Element) and Madaline (Multiple Adaline) were developed by Widrow. The structures use neurons and step/ sigmoidal activation function. ADALINE has one output neuron but MADALINE has many. The learning is different from a perceptron. It is here by Widrow-Hoff or LMS (Least Mean Square error) rule. Analogical input or output can be found by this network as minimum error function is searched before applying activation function.

ADALINE:

The structure includes an adaptive linear combiner (ALC) to obtain linear response that can be applied to other elements of bipolar commutation. IF O/P of ALC is +ve response of ADALINE is +1, and if -ve result of ADALINE is -1. It is represented by:

$$y(t+1) = \begin{cases} +1 & s > 0 \\ y(t) & s = 0 \\ -1 & s < 0 \end{cases}$$

The binary answer corresponding to the Adaline network is:

$$s = \sum_{j=0}^N w_j x_j = \mathbf{w}^T \mathbf{X} \quad (3.11)$$

The Adaline Network can be used to generate an analogical response using a sigmoid commuter, instead of a binary one, in such a case the output y will be obtained applying a sigmoidal function.

The Adaline and Madaline Networks use a supervised learning rule, off-line, denominated Least Mean Squared (LMS), through which the vector of weight, \mathbf{w} , should associate with success each input pattern with its corresponding value of desired output, d_k . Particularly, the network training consists in modifying the weights when the training patterns and desired outputs are presented. With each combination input-output an automatic process is made of little adjustments in the weights values until the correct outputs are obtained.

Concretely, the learning rule LMS minimizes the mean squared error, defined as:

$$E = \frac{1}{2P} \sum_{k=1}^P \varepsilon_k^2 \quad (3.13)$$

Where P is the quantity of input vectors (patterns) that form the training group, and ε_k is the difference between the desired output and the obtained when is introduced the k -th pattern, in the case of the Adaline network, it expressed as $\varepsilon_k = (d_k - s_k)$, where s_k corresponds to the exit of the ALC (expression 3.11), that is to say:

$$s_k = \sum_{j=0}^N w_j x_{jk} = \mathbf{w}^T \mathbf{X}_k \quad (3.14)$$

Even when the error surface is unknown, equation (3.13), the gradient descent method gets to obtain the local information of it. With this information it is decided what direction to take to arrive to the global minimum. Therefore, based in the gradient descent method, the rule known as delta rule or LMS rule is obtained. With this the weight are modified so that the new point, in the space of weights, is closer to the minimum point. In other words, the modification of the weights is proportional to the gradient descent of the error function $\Delta w_j = -\eta(\partial E_k / \partial w_j)$. Using the chain rule for the calculus of the derivative of expression (3.13), we obtain:

$$w_j(t+1) = w_j(t) + \eta \varepsilon_k x_{kj} \quad (3.15)$$

Based in this, the learning algorithm of an Adaline network contains the following steps:

1. A pattern is introduced $X_k = (x_{k1}, x_{k2}, \dots, x_{kN})$ in the entrances of Adaline
2. The linear output is obtained $s_k = \sum_{j=0}^N w_j x_{jk} = \mathbf{w}^T \mathbf{X}_k$ and it is calculated the difference with respect from what is expected $\varepsilon_k = (d_k - s_k)$.
3. The weights are actualized.

$$\begin{aligned} \mathbf{w}(t+1) &= \mathbf{w}(t) + \eta \varepsilon_k \mathbf{X}_k \\ w_j(t+1) &= w_j(t) + \eta \varepsilon_k x_{kj} \end{aligned}$$

4. Steps from 1 to 3 are repeated, with all the entrance vectors.
5. If the mean squared error

$$E = \frac{1}{2P} \sum_{k=1}^P \varepsilon_k^2$$

Has a reduced value, the learning process ends; if not, the process is repeated from step one with all the patterns.

Madaline network

The Madaline network (Multiple Adaline) is formed as a combination of Adaline modules, which are structured in layers (figure 3.4). Madaline overcomes some of the limitations of Adaline networks.

Given the differences between Madaline and Adaline, the training of these networks is not the same. The LMS algorithm can be applied to the output layer because its exit is already known for each one of the entrance patterns. However, the expected exit is unknown for the nodes of the hidden layer. Besides, the algorithm LMS works for the linear outputs (analogical), of the adaptive combiner and not for the digital of the Adaline.

To employ the algorithm LMS in the Madaline training is required to modify the activation function for a derivable continuous function (the step function is discontinuous in zero and therefore not derivable in this point).

Because of its similarities with the Multilayer Perceptron, the description of the Madaline learning rule will not be described, this will be handled lately.

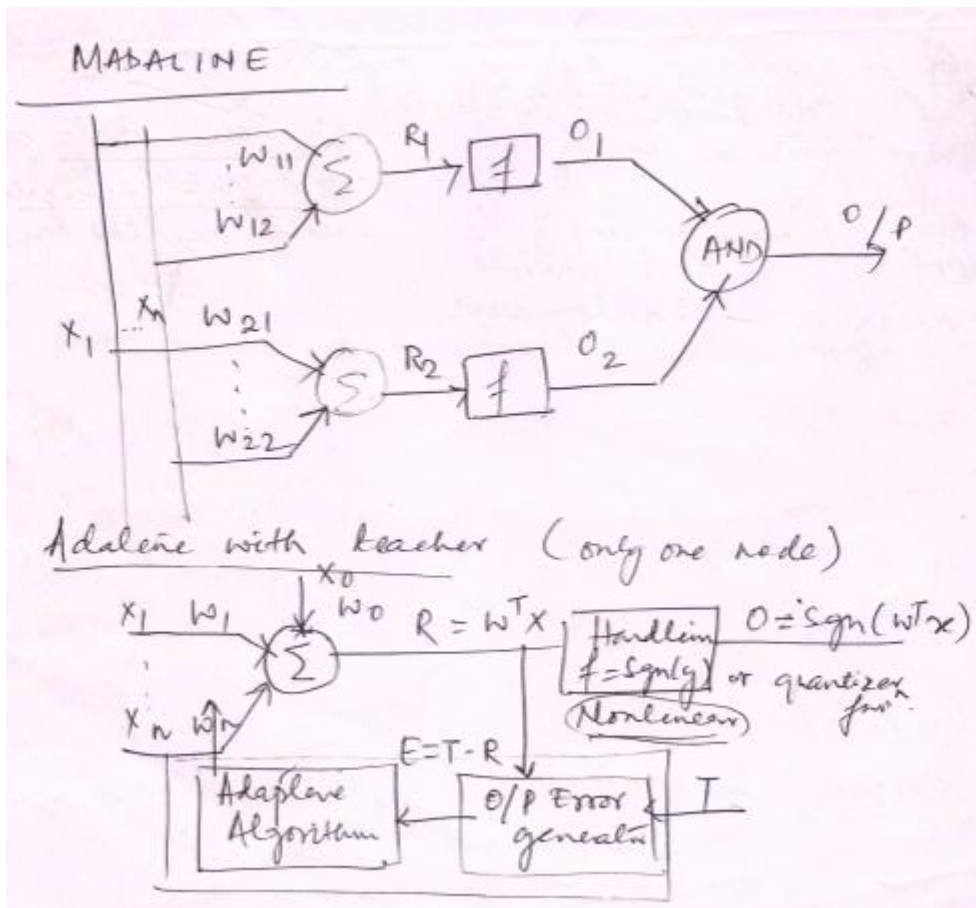


Fig. MADALINE network

LECTURE-5

The Multi-layered Perceptron training:

Improvements over Perceptron:

- 1) Smooth nonlinearity - sigmoid
- 2) 1 or more hidden layers

Training the hidden layer:

Not obvious how to train the hidden layer parameters.

The error term is meaningful only to the weights connected to the output layer. How to adjust hidden layer connections so as to reduce output error? – *credit assignment* problem.

Any connection can be adapted by taking a full partial derivative over the error function, but then to update a single weight in the first stage we need information about distant neurons/connections close to the output layer (locality rule is violated). In a large network with many layers, this implies that information is exchanged over distant elements of the network though they are not directly connected. Such an algorithm may be mathematically valid, but is biologically unrealistic.

The Backpropagation Algorithm:

As in Perceptron, this training algorithm involves 2 passes:
 The forward pass – outputs of various layers are computed
 The backward pass – weight corrections are computed
 Consider a simple 3-layer network with a single neuron in each layer.

Total output error over all patterns:
$$E = \sum_p E_p$$

Squared Output error for the p'th pattern:
$$E_p = \frac{1}{2} \sum_i e_i^2$$

Output error for the p'th pattern:
$$e_i = d - y_i$$

Network output:
$$y_i = g(h_i^s)$$

Net input to the output layer:
$$h_i^s = \sum_j w_{ij}^s V_j - \theta_i^s$$

Output of the hidden layer:
$$V_j^f = g(h_j^f)$$

Net input of the hidden layer:
$$h_j^f = \sum_k w_{jk}^f x_k - \theta_j^f$$

Update rule for the weights using gradient descent:

$$\Delta w_{ij}^s = -\eta \frac{\partial E_p}{\partial w_{ij}^s}; \quad \Delta \theta_i^s = -\eta \frac{\partial E_p}{\partial \theta_i^s}$$

$$\Delta w_{jk}^f = -\eta \frac{\partial E_p}{\partial w_{jk}^f}; \quad \Delta \theta_j^f = -\eta \frac{\partial E_p}{\partial \theta_j^f}$$

Updating w_{ij}^s :

$$\frac{\partial E_p}{\partial w_{ij}^s} = \frac{\partial E_p}{\partial e_i} \frac{\partial e_i}{\partial y_i} \frac{\partial y_i}{\partial h_i^s} \frac{\partial h_i^s}{\partial w_{ij}^s}$$

$$= e(-1)g'(h_i^s)V_j$$

The delta at the output layer, δ_i^s , is defined as,

$$\delta_i^s = e_i g'(h_i^s) \quad (6.2.2.1.11)$$

Therefore,

$$\Delta w_{ij}^s = -\eta \frac{\partial E_p}{\partial w_{ij}^s} = \eta \delta_i^s V_j \quad (6.2.2.1.12)$$

By similar arguments, it can be easily be shown that the update rule for the threshold term is,

$$\Delta \theta_i^s = -\eta \delta_i^s \quad (6.2.2.1.13)$$

Updating w_{jk}^f :

$$\begin{aligned} \Delta w_{jk}^f &= -\eta \frac{\partial E_p}{\partial w_{jk}^f} \\ \frac{\partial E_p}{\partial w_{jk}^f} &= \sum_i \frac{\partial E_p}{\partial e_i} \frac{\partial e_i}{\partial y_i} \frac{\partial y_i}{\partial h_i^s} \frac{\partial h_i^s}{\partial V_j} \frac{\partial V_j}{\partial w_{jk}^f} \\ &= \sum_i e_i (-1) g'(h_i^s) w_{ij}^s \frac{\partial V_j}{\partial w_{jk}^f} \\ &= \sum_i e_i (-1) g'(h_i^s) w_{ij}^s g'(h_j^f) x_k \\ &= \sum_i \delta_i^s w_{ij}^s g'(h_j^f) x_k \end{aligned} \quad (6.2.2.1.14)$$

Define an error term at the hidden layer as,

$$\delta_j^f = \sum_i \delta_i^s w_{ij}^s g'(h_j^f) \quad (6.2.2.1.15)$$

Therefore,

$$\Delta w_{jk}^f = -\eta \delta_j^f x_k \quad (6.2.2.1.16)$$

Similarly the update rule for the threshold term is,

$$\Delta \theta_j^f = -\eta \delta_j^f \quad (6.2.2.1.17)$$

weight correction = (learning rate) * (local δ from ‘top’) * (activation from ‘bottom’)

General formulation of the Backpropagation Algorithm:

Notation:

| | | |
|---------------------------------------|---|--|
| Input (at k`th input neuron) | - | x_k |
| Actual Output (at i`th output neuron) | - | y_i |
| Target output (at i`th output neuron) | - | d_i |
| Hidden neuron activation | - | V_j^l (of j`th neuron in l`th layer) |
| Layer number | - | $l=0$ (input layer) to L (output layer) |
| Net input | - | h_{jl} (for j`th neuron in l`th layer) |
| $g(h)$ | - | sigmoid nonlinearity = $1/(1+\exp(-\beta h))$ |

Steps:

1. Initialize weights with small random values
2. (Loop over training data)
3. Choose a pattern and apply it to the input layer

$$V_k^0 = x_k^p \quad \text{for all } k.$$

4. Propagate the signal forwards thro’ the network using:

$$V_j^l = g(h_j^l) = g\left(\sum_k w_{jk}^l V_k^{l-1} - b_j^l\right).$$

for each j, k and and ‘l’ until final outputs V_i^L have all been calculated.

5. Compute errors, δ ’s, for the output layer.

$$\delta_i^L = g'(h_i^L)[d_i(p) - V_i^L]$$

6. Compute δ ’s for preceding layers by backpropagation of error:

$$\delta_i^{l-1} = g'(h_i^{l-1}) \left[\sum_j w_{ji}^l \delta_j^l \right]$$

For $l = L, L-1, \dots, 1$

7. Update weights using the following:

$$\Delta w_{ij}^l = \eta \delta_i^l V_j^{l-1};$$

$$\Delta \theta_i^l = -\eta \delta_i^l$$

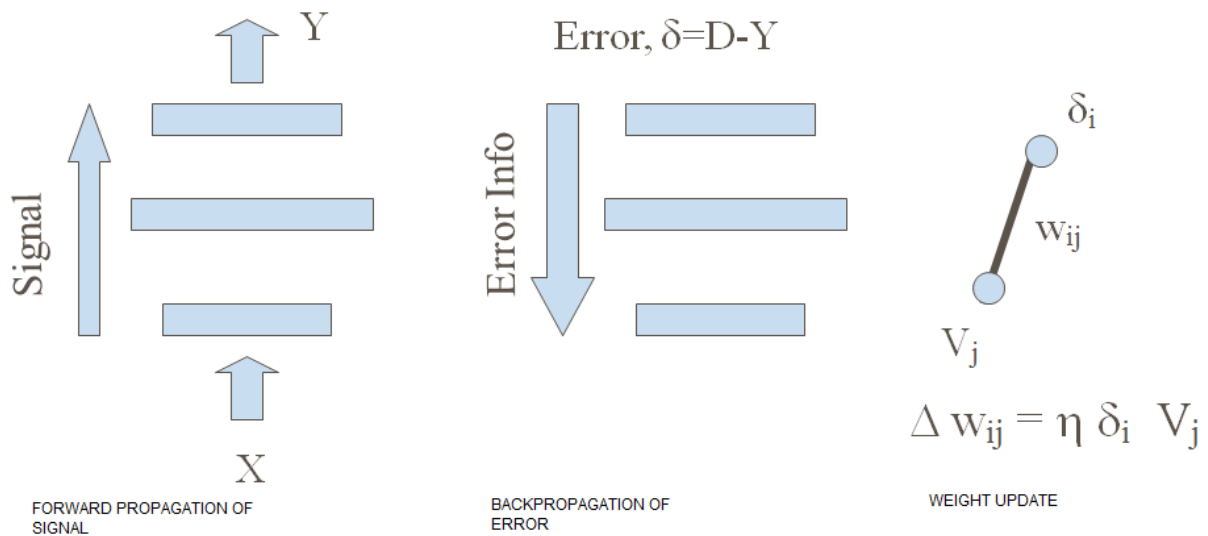


Fig. Training using Back propagation algorithm

Training:

Randomly initialize weights.

Train network using backprop eqns.

Stop training when error is sufficiently low and freeze the weights.

Testing

Start using the network.

Merits of MLP trained by BP:

- a) A general solution to a large class of problems.
- b) With sufficient number of hidden layer nodes, MLP can approximate arbitrary target functions.
- c) Backprop applies for arbitrary number of layers, partial connectivity (no loops).
- d) Training is local both in time and space – parallel implementation made easy.
- e) Hidden units act as “feature detectors.”
- f) Good when no model is available

Problems with MLP trained by BP:

- a) Blackbox approach
- b) Limits of generalization not clear
- c) Hard to incorporate prior knowledge of the model into the network
- d) slow training

e) local minima

LECTURE-6

Architectures of MLP:

If there is no nonlinearity then an MLP can be reduced to a linear neuron.

1. Universal Approximator:

Theorem:

Let $g(\cdot)$ be a nonconstant, bounded, and monotone-increasing continuous function. Let I_m denote the m -dimensional hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then given any function f in $C(I_m)$ and $\varepsilon > 0$, there exist an integer M and sets of real constants a_i , b_i and w_{ij} , where $i = 1, \dots, n$, and $j = 1, \dots, m$ such that we may define:

$$F(x_1, \dots, x_m) = \sum_{i=1}^n a_i g\left(\sum_{j=1}^m w_{ij} x_j + b_i\right)$$

As an approximate realization of the function $f(\cdot)$; that is,

$$|F(x_1, \dots, x_m) - f(x_1, \dots, x_m)| < \varepsilon$$

for all x_1, \dots, x_m that lie in the input space.

For the above theorem to be valid, the sigmoid function $g(\cdot)$ has to satisfy some conditions. It must be: 1) non-constant, 2) bounded, 3) monotone-increasing and 4) continuous.

All the four transfer functions described in the section on Perceptrons satisfy conditions #1,2 and 3. But the hardlimiting nonlinearities are not continuous. Therefore, the logistic function or the tanh function are suitable for use as sigmoids in MLPs.

2. In general more layers/nodes greater network complexity

Although 3 hidden layers with full connectivity are enough to learn any function often more hidden layers and/or special architectures are used.

More hidden layers and/or hidden nodes:

3-layer network:

Arbitrary continuous function over a finite domain

4-layer network

Neurons in a 3-layer architecture tend to interact globally.

In a complex situation it is hard to improve the approximation at one point without worsening it at another.

So in a 4-layer architecture:

1st hidden layer nodes are combined to construct locally sensitive neurons in the second hidden layer.

Discontinuous functions:

learns discontinuous (inverse function of continuous function) functions also (Sontag, 1992)

For hard-limiting threshold functions:

1st hidden layer: semi-infinite regions separated by a hyper-plane

2nd hidden layer: convex regions

3rd hidden layer: non-convex regions also

Training MLP:

1. **Initialization:** is VERY important.

$g'(\cdot)$ appears on the right side of all weight update rules (see sections 6.1.1, 6.1.2, 6.2.1). Note that $g'(\cdot)$ is high at the origin and falls on both sides. Therefore most learning happens when the net input (h) to the neurons is close to 0. Hence it is desirable to make initial weights small. A general rule for initialization of input weights for a given neuron is:

$$\text{Mean}(w(0)) = 0.$$

$$\text{std}(w(0)) = \frac{1}{\sqrt{m}} \text{ where } m \text{ is the number of inputs going into a neuron.}$$

2. Batch mode and Sequential mode:

Epoch: presentation of all training patterns is called an epoch.

Batch mode:

Updating network weights once every epoch is called batch mode update.

- memory intensive
- greater chance of getting stuck in local minima

Sequential mode:

Updating the network weights after every presentation of a data point is sequential mode of update.

- lesser memory requirement
- The random order of presentation of input patterns acts as a noise source lesser chance of local minima

Rate of learning:

We have already seen the tradeoffs involved in choice of a learning rate.

Small learning rate η , approximate original continuous domain equations more closely but slows down learning.

Large learning rate η , poorer approximation of original equations. Error may not decrease monotonically and may even oscillate. But learning is faster..

A good thumb rule for choosing eta ' η ':

$$\eta = 1/m$$

Where 'm' is the number of inputs to a neuron. This rule assumes that there are different η s for different neurons.

3. Important tip relating learning rate and error surface:

Rough error surface, slow down, low η

Smooth (flat) error surface, speed up, high η

i) Momentum:

- a) If $|a| < 1$, the above time-series is convergent.
- b) If the sign of the gradient remains the same over consecutive iterations the weighted sum Δw_{ji} grows exponentially i.e., accelerate when the terrain is clear.
- c) If the gradient changes sign in consecutive iterations, Δw_{ji} shrinks in magnitude i.e., slow down when the terrain is rough.

i) Momentum:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

Action of momentum:

$$\Delta w_{ji}(n) = -\eta \sum_{t=0}^n \alpha^{n-t} \delta_j(t) y_i(t) = -\eta \sum_{t=0}^n \alpha^{n-t} \frac{\partial E(t)}{\partial w_{ji}(t)}$$

ii) Separate eta for each weight:

- Separate η for each weight
- Every eta varies with time
- If $\delta(w)$ changes sign several time in the past few iters, decrease η
- If $\delta(w)$ doesn't change sign in the past few iters, increase η

Stopping Criteria: when do we stop training?

- Error < a minimum.
- Rate of change in error averaged over an epoch < a minimum.
- Magnitude of gradient $\|g(w)\| < a \text{ minimum.}$
- When performance over a test set has peaked.

Premature Saturation:

All the weight modification activity happens only when $|h|$ is within certain limits.
 $g'(h) \approx 0$, or $\delta(w) = 0$, for large $|h|$.

NN gets stuck in a shallow local minimum.

Solutions:

- Keep a copy of weights
- Retract to pre-saturation state
- Perturb weights, decrease η and proceed
- Reduce sigmoid gain (λ) initially
- Increase λ gradually as error is minimized

Network doesn't get stuck, but never settles either.

Testing/generalization:

Idea of overfitting or overtraining:

Using too many hidden nodes, may cause overtraining. The network might just learn noise and generalize poorly.

Example of polynomial interpolation:

Consider a data set generated from a quadratic function with noise added. A linear fit is likely to give a large error. Best fit is obtained with a quadratic function. Fit 10^{th} degree might give a low error but is likely to learn the variations due to noise also. Such a fit is likely to do poorly on a test data set. This is called overfitting or poor generalization.

This happens because there are many ways of generalizing from a given training data set.

The above Venn diagram illustrates the possibility of generalizing in multiple ways from a given training data set. U is the universe of all possible input-output patterns. F (the ellipse)

represents the set of I/O pairs that define the function to be learnt by the mlp. T (circle) denotes the training data set which is a subset of F. X denotes the test data set. The dotted rectangle denotes the actual function learnt by the NN, which is consistent with the training set T, but is completely non-overlapping with the test set X, and very different from the unknown function F.

A simple calculation from (Hertz et al 1991).

Boolean function

N-inputs, 1-output

2^N patterns and 2^{2^N} rules totally.

Assume,

p – training patterns say, represent the rule T.

Then there are $(2^N - p)$ test patterns and there are $2^{(2^N - p)}$ rules, R, consistent with rule T.

N = 30, p = 1000 patterns.

You have 2^{10^9} generalizations exist for the same training set T.

Applications of MLP

Three applications of MLPs that simulate aspects of sensory, motor or cognitive functions are described.

1. Nettalk
2. Past tense learning
3. Autonomous Land Vehicle in a Neural Network (ALVINN)

[LECTURE-6](#)

Multilayer Feed-Forward Network:

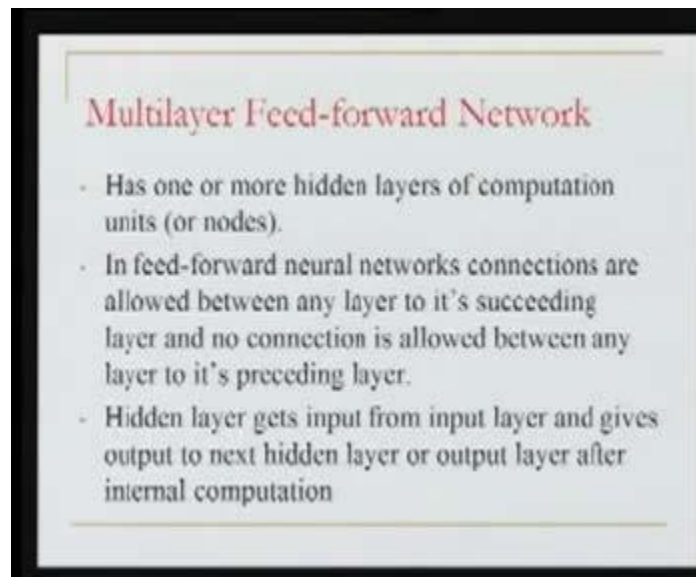


Fig. Characteristics of Multilayer feed-forward network

The algorithm that was derived using gradient descent for nonlinear neural networks with nonlinear activation function is popularly known as back propagation learning algorithm, although the learning algorithm still is derived using gradient descent rule.

Multilayer feed forward network has more hidden layers and again, when I say feed forward network, the connections are all allowed only from any layer to its succeeding layer, but the connections are not allowed from any layer to its preceding layer. The example is you see here there are four layers. These are all inputs. First hidden layer, second hidden layer, third hidden layer and this is output layer. When we say the number of layers, we do not count the input layer as one of the layers. When I say two layered network, then I have only one hidden layer and next layer becomes output layer.

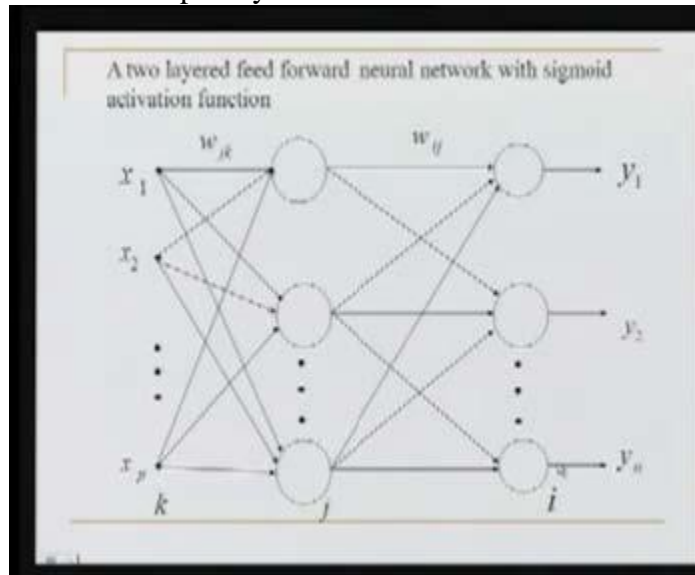


Fig. Multilayer feed forward network

This particular configuration means there are sub-units, sub-neurons here and this particular configuration, if I connect you will see why I say feed forward network, because I am able to connect any layer from its preceding layer. That means connections are allowed from the preceding layer to any layer, but cannot allow the feedback connection. (Refer Slide Time: 30:54) This is called feedback connection; this is not allowed. This is allowed. From this layer, I can connect to this layer. This is allowed, but I cannot allow from this layer to connect to this layer. These are called feedback connections. They are not allowed and that is why this is known as feed forward network.

Today, we will derive a two-layered feed forward neural network with sigmoid activation function. We can very easily see that this is 1 layer; this is the only hidden layer and this is the only output layer; output layer is always only one.

We have a certain convention that we will put while deriving a back propagation learning algorithm for this. The same simple principle; given training data, we allow the input to pass through the network, compute the error here, use the gradient descent rule and the back propagated error are used to modify the weights here that is between output layer and hidden layer and again another form of back propagated error here has to be used for modification of the weights between input layer and hidden layer. This is again the convention that we will use.

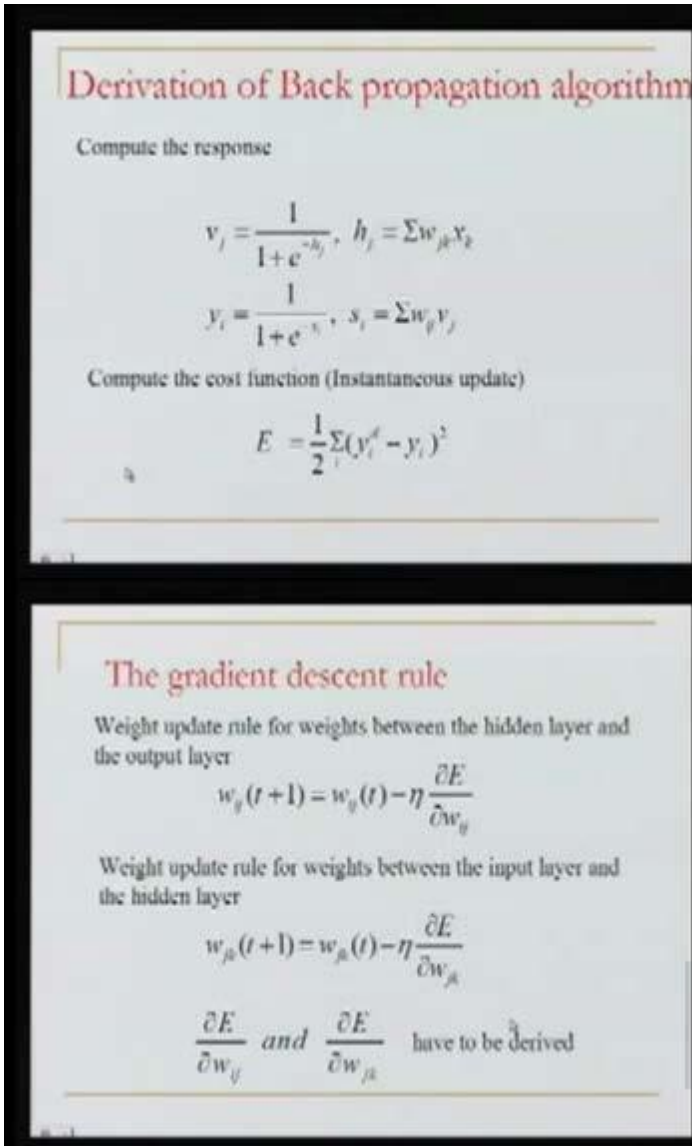


Fig. The Gradient descent rule

After choosing the weights of the network randomly, the backpropagation algorithm is used to compute the necessary corrections. The algorithm can be decomposed in the following four steps:

- i) Feed-forward computation
- ii) Backpropagation to the output layer
- iii) Backpropagation to the hidden layer
- iv) Weight updates

The algorithm is stopped when the value of the error function has become sufficiently small.

In the case of $p > 1$ input-output patterns, an extended network is used to compute the error function for each of them separately. The weight corrections The Backpropagation Algorithm are computed for each pattern and so we get, for example, for weight $w_{ij}^{(1)}$ the corrections

$$\Delta_1 w_{ij}^{(1)}, \Delta_2 w_{ij}^{(1)}, \dots, \Delta_p w_{ij}^{(1)}$$

The necessary update in the gradient direction is then

$$\Delta w_{ij}^{(1)} = \Delta_1 w_{ij}^{(1)} + \Delta_2 w_{ij}^{(1)} + \dots + \Delta_p w_{ij}^{(1)}$$

We speak of batch or off-line updates when the weight corrections are made in this way. Often, however, the weight updates are made sequentially after each pattern presentation (this is called on-line training). In this case the corrections do not exactly follow the negative gradient direction, but if the training patterns are selected randomly the search direction oscillates around the exact gradient direction and, on average, the algorithm implements a form of descent in the error function. The rationale for using on-line training is that adding some noise to the gradient direction can help to avoid falling into shallow local minima of the error function. Also, when the training set consists of thousands of training patterns, it is very expensive to compute the exact gradient direction since each epoch (one round of presentation of all patterns to the network) consists of many feed-forward passes and on-line training becomes more efficient.

Back Propagation Neural Network

Backpropagation is a training method used for a multi layer neural network. It is also called the generalized delta rule. It is a gradient descent method which minimizes the total squared error of the output computed by the net. Any neural network is expected to respond correctly to the input patterns that are used for training which is termed as memorization and it should respond reasonably to input that is similar to but not the same as the samples used for training which is called generalization. The training of a neural network by back propagation takes place in three stages 1. Feedforward of the input pattern 2. Calculation and Back propagation of the associated error 3. Adjustments of the weights After the neural network is trained, the neural network has to compute the feedforward phase only. Even if the training is slow, the trained net can produce its output immediately.

Architecture

A multi layer neural network with one layer of hidden units is shown in the figure. The output units and the hidden units can have biases. These bias terms are like weights on connections from units whose output is always 1. During feedforward the signals flow in the forward direction i.e. from input unit to hidden unit and finally to the output unit. During back propagation phase of learning, the signals flow in the reverse direction.

Algorithm

The training involves three stages 1. Feedforward of the input training pattern 2. Back propagation of the associated error 3. Adjustments of the weights. During feedforward, each input unit (X_i) receives an input signal and sends this signal to each of the hidden units Z_1, Z_2, \dots, Z_n . Each hidden unit computes its activation and sends its signal to each output unit. Each output unit computes its activation to compute the output or the response of the neural net for the given input pattern.

During training, each output unit compares its computed activation y_k , with its target value t_k to determine the associated error for the particular pattern. Based on this error the factor δ_k for all m values are computed. This computed δ_k is used to propagate the error at the output unit Y_k back to all units in the hidden layer. At a later stage it is also used for updation of weights between the output and the hidden layer. In the same way δ_j for all p values are computed for each hidden unit Z_j . The values of δ_j are not sent back to the input units but are used to update the weights between the hidden layer and the input layer. Once all the δ factors are known, the weights for all layers are changed simultaneously. The adjustment to all weights w_{jk} is based on the factor δ_k and the activation z_j of the hidden unit Z_j . The change in weight to the connection between the input layer and the hidden layer is based on δ_j and the activation x_i of the input unit.

Activation Function

An activation function for a back propagation net should have important characteristics. It should be continuous, Differentiable and monotonically non- decreasing. For computational efficiency, it is better if the derivative is easy to calculate. For the commonly used activation function, the derivative can be expressed in terms of the value of the function itself. The function is expected to saturate asymptotically. The commonly used activation function is the binary sigmoidal function.

Training Algorithm

The activation function used for a back propagation neural network can be either a bipolar sigmoid or a binary sigmoid. The form of data plays an important role in choosing the type of the activation function. Because of the relationship between the value of the function and its derivative, additional evaluations of exponential functions are not required to be computed.

Algorithm

Step 0: Initialize weights

Step 1: While stopping condition is false, do steps 2 to 9

Step 2: For each training pair, do steps 3 - 8 *Feed forward*

Step 3: Input unit receives input signal and propagates it to all units in the hidden layer

Step 4: Each hidden unit sums its weighted input signals

Step 5: Each output unit sums its weighted input signals and applied its activation function to compute its output signal.

Backpropagation Step 6: Each output unit receives a target pattern corresponding to the input training pattern, computes its error information term $\delta_k = (t_k - y_k) f'(y_{ink})$ Calculates its bias correction term $\Delta W_{ok} = \alpha \delta_k$ And sends δ_k to units in the layer below

Step 7: Each hidden unit sums its delta inputs Multiplies by the derivative of its activation function to calculate its error information term Calculates its weight correction term $\Delta v_{ij} = \alpha \delta_j x_i$ And calculates its bias correction term $\Delta v_{oj} = \alpha \delta_j$ *Update weights and biases*

Step 8: Each output unit updates its bias and weights $W_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}$ Each hidden unit updates its bias and weights $V_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}$

Step9:Test stopping condition

LECTURE-7

Radial Basis Function Networks:

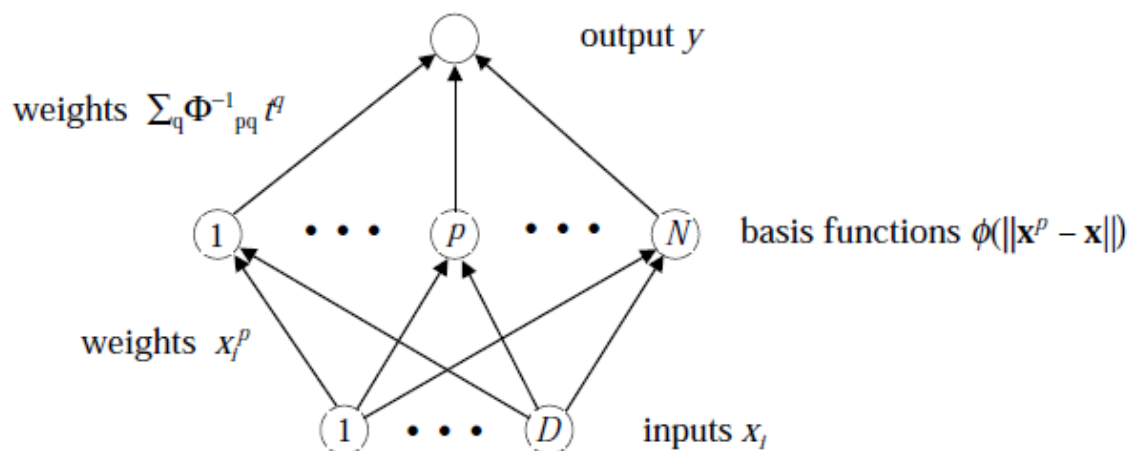


Fig. RBF network

- These are 3-layer networks that can approximate any continuous function through a basis function expansion.
- The basis functions here (which are data dependent as earlier) exhibit some radial symmetry.
- These networks have the so called perfect interpolation property.

The function represented by an RBF network with p hidden nodes can be written as

$$y = \sum_{j=1}^p w_j \phi (\|X - \theta_j\|),$$

X is the input to the network.

- w_j is weight from j^{th} hidden node to the output.
- $\phi (\|X - \theta_j\|)$ is the output of the j^{th} hidden node and θ_j is the parameter vector associated with j^{th}

hidden node, $j = 1, \dots, p$.

A very popular model is the Gaussian RBF network.

- Here the output is written as

$$y = \sum_{j=1}^p w_j \exp \left(- \frac{\|X - \theta_j\|^2}{2\sigma^2} \right)$$

- The θ_j is called the center of the j^{th} hidden or RBF node and σ is called the width.
- We can have different σ for different hidden nodes.

We next consider learning the parameters of a RBF network from training samples.

- Let $\{(X_i, d_i), i = 1, \dots, N\}$ be the training set.
- Suppose we are using the Gaussian RBF.
- Then we need to learn the centers (θ_j) and widths (σ) of the hidden nodes and the weights into the output node (w_j).

Like earlier, we can find parameters to minimize empirical risk under squared error loss function.

- Same as minimizing sum of squares of errors. Let

$$J = \sum_{i=1}^N \left(\sum_{j=1}^p w_j \exp \left(- \frac{\|X^i - \theta_j\|^2}{2\sigma^2} \right) - d^i \right)^2$$

J is a function of $\sigma, w_j, \theta_j, j = 1, \dots, p$.

We can find the weights/parameters of the network to minimize J.

- To minimize J, we can use the standard iterative algorithm of gradient descent.
- This needs computation of gradient which can be done directly from the expression for J.

- For this network structure there are no special methods to evaluate all the needed partial derivatives. Such a gradient descent algorithm is certainly one method of learning an RBF network from given training data.
- This is a general-purpose method for learning an RBF network.
- Like in the earlier case, we have to fix p , the number of hidden nodes.
- Such procedure would have the usual problems of converging to a local minimum of the error function.
- There are also other methods of learning an RBF network.
- If we have the basis functions, ϕ_j , then it is exactly same as a linear model and we can use standard linear least squares method to learn w_j .
- To fix ϕ_j , we need to essentially fix θ_j (and may be σ).
- So, if we can somehow fix centers and widths of the RBF nodes, then we can learn the w_j very easily.

As we have discussed earlier, these RBF networks use 'local' representations.

- What this means is that θ_j should be 'representative' points of the feature space and they should 'cover' the feature space.
- Essentially, the proof that these networks can represent any continuous function is based on having such centers for RBF nodes.
- We can use such ideas to formulate methods for fixing centers of RBF nodes.

One simple method of choosing centers, θ_j , is to randomly choose p of the training examples.

- We know that with N hidden nodes and centers same as training examples, we get perfect interpolation.
- Hence we can take some of the training examples as centers.
- There can be some variations on this theme.
- However, such a method does not, in general, ensure that we have representative points in the feature space as centers.

When we have p hidden nodes, we need p 'centers'.

- Hence we are looking for p number of 'representative' points in the feature space.
- The only information we have are the N training examples.
- Hence the problem is:

given N points, $X_i, i = 1, \dots, N$ in $\langle m \rangle$, find p 'representative' points in $\langle m \rangle$.

- This is the 'clustering problem' This is a problem of forming the data into p clusters.
- We can take the 'cluster centers' to be the representative points.
- The kind of clusters we get depends on how we want to formalize the notion of the p points being representative of the N data points.
- We now look at one notion of clustering that is popular.

Let $\theta_1, \dots, \theta_p$ represent the p cluster centers.

- Now we need an objective function that specifies how representative these are of the data $X_i, i = 1, \dots, N$.

Now we can define a cost function as

$$J = \sum_{j=1}^p \sum_{X^i \in S_j} \|X^i - \mu_j\|^2$$

- The J is a function of $\theta_j, j = 1, \dots, p$. (Note that S_j are also functions of the θ_j 's).

- For a given set of centers, $\{\theta_j\}$, J gives us the total error in approximating each of the training data by its nearest cluster center.

- Hence we want to choose centers to minimize J . We now discuss a simple algorithm to find centers to minimize J .

- This is known as K-means clustering algorithm.

(Originally proposed by Lloyd in the context of vector quantization).

- We are given N data points, $X_i, i = 1, \dots, N$.

We want to find p cluster centers $\theta_j, j = 1, \dots, p$, to minimize J .

- We first rewrite J in a different form to motivate our algorithm.

We think of the problem as finding the centers $\theta_1, \dots, \theta_p$ and assigning X_i to these clusters.

- Let $\mu_j, n = 1, \dots, N, j = 1, \dots, p$ be indicators of the cluster assignment.

- That is, if we assign X^n to cluster j , then we would have $\rho_j = 1$ and $\rho_p = 0$,

- Now we can rewrite J as

$$J = \sum_{n=1}^N \sum_{j=1}^p \rho_{nj} \|X^n - \mu_j\|^2$$

- Since we are taking the Euclidian norm, we have

$$J = \sum_{n=1}^N \sum_{j=1}^p \rho_{nj} (X^n - \mu_j)^T (X^n - \mu_j)$$

- Note (for later use) that gradient of J w.r.t. μ_j is

$$2 \sum_{n=1}^N \rho_{nj} (X^n - \mu_j)$$

We now have to find a way of minimizing J wrt all ρ_{nj} and μ_j .

- First consider finding optimal values for ρ_{nk} with all the μ_j fixed.

- We know that the variables $\{\rho_{nk}\}$ have to satisfy $\rho_{nk} \in \{0, 1\}$ and $\sum_k \rho_{nk} = 1, \forall n$. Recall

$$J = \sum_{n=1}^N \sum_{j=1}^p \rho_{nj} \|X^n - \mu_j\|^2$$

- Given the form of J , we can decouple optimization with respect to ρ_{nj} for different n .

- For a specific n we need to find ρ_{nk} , $k = 1, \dots, p$, to minimize

$$\sum_{j=1}^p \rho_{nj} \|X^n - \mu_j\|^2$$

- Given the constraints on ρ_{nj} , the optimum would be

$$\begin{aligned} \rho_{nj} &= 1 && \text{if } \|X^n - \mu_j\|^2 \leq \|X^n - \mu_k\|^2, \forall k \\ &= 0 && \text{otherwise} \end{aligned}$$

- Now fixing the ρ_{nj} , consider optimizing J w.r.t. μ_j .
- For a specific j , equating the gradient of J w.r.t. μ_j to zero, we get

$$\sum_{n=1}^N \rho_{nj} (X^n - \mu_j) = 0$$

- This gives us the optimum value for μ_j as

$$\mu_j = \frac{\sum_{n=1}^N \rho_{nj} X^n}{\sum_{n=1}^N \rho_{nj}}$$

Note that for a given n , ρ_{nj} is 1 for exactly one j (and it is zero otherwise).

- Thus the μ_j would be the mean of all data vectors assigned to the j^{th} cluster.
- This is the reason for the name K-means clustering.
- What we derived are optimum values for ρ_{nj} keeping μ_j fixed and optimum values for μ_j keeping ρ_{nj} fixed.
- Hence, in an algorithm we do this repeatedly.
- This is like the EM algorithm.

Commonly Used Radial Basis Functions

A range of theoretical and empirical studies have indicated that many properties of the interpolating function are relatively insensitive to the precise form of the basis functions $\phi(r)$. Some of the most commonly used basis functions are:

1. Gaussian Functions:

$$\phi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \quad \text{width parameter } \sigma > 0$$

2. Multi-Quadric Functions:

$$\phi(r) = (r^2 + \sigma^2)^{1/2} \quad \text{parameter } \sigma > 0$$

3. Generalized Multi-Quadric Functions:

$$\phi(r) = (r^2 + \sigma^2)^\beta \quad \text{parameters } \sigma > 0, 1 > \beta > 0$$

4. Inverse Multi-Quadric Functions:

$$\phi(r) = (r^2 + \sigma^2)^{-1/2} \quad \text{parameter } \sigma > 0$$

5. Generalized Inverse Multi-Quadric Functions:

$$\phi(r) = (r^2 + \sigma^2)^{-\alpha} \quad \text{parameters } \sigma > 0, \alpha > 0$$

6. Thin Plate Spline Function:

$$\phi(r) = r^2 \ln(r)$$

7. Cubic Function:

$$\phi(r) = r^3$$

8. Linear Function:

$$\phi(r) = r$$

Experiences or learning:

Learning algorithms use experiences in the form of perceptions or perception action pairs to improve their performance. The nature of experiences available varies with applications. Some common situations are described below.

Supervised learning: In supervised learning a teacher or oracle is available which provides the desired action corresponding to a perception. A set of perception action pair provides what is called a training set. Examples include an automated vehicle where a set of vision inputs and the corresponding steering actions are available to the learner.

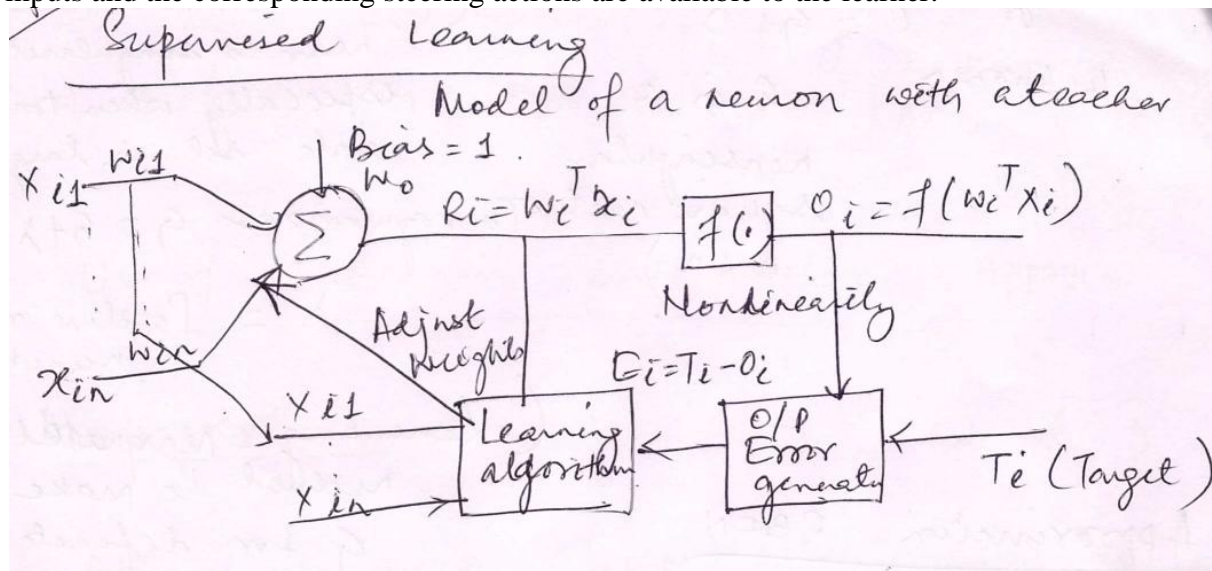


Fig. Supervised learning

Unsupervised learning: In unsupervised learning no teacher is available. The learner only discovers persistent patterns in the data consisting of a collection of perceptions. This is also called exploratory learning. Finding out malicious network attacks from a sequence of anomalous data packets is an example of unsupervised learning.

Active learning: Here not only a teacher is available, the learner has the freedom to ask the teacher for suitable perception-action example pairs which will help the learner to improve its performance. Consider a news recommender system which tries to learn an users preferences and categorize news articles as interesting or uninteresting to the user. The system may present a particular article (of which it is not sure) to the user and ask whether it is interesting or not.

Reinforcement learning: In reinforcement learning a teacher is available, but the teacher instead of directly providing the desired action corresponding to a perception, return reward and punishment to the learner for its action corresponding to a perception. Examples include a robot in a unknown terrain where its get a punishment when its hits an obstacle and reward when it moves smoothly.

In order to design a learning system the designer has to make the following choices based on the application.

LECTURE-9

Unsupervised Learning in Neural Networks:

Unsupervised learning mechanisms differ from supervised learning in that there is no "teacher" to instruct the network.

Competitive Learning:

Competitive learning is a form of unsupervised learning which performs clustering over the input data. In a competitive learning network with n -output neurons, each output neuron is associated with a cluster. When a data point from a cluster is presented to the network, only the neuron corresponding to that cluster responds, while all other neurons remain silent. The single neuron that responds is often called a “winner” and therefore a competitive learning network of the kind just described is also known as a “winner-take-all” network.

It is easiest to introduce CL mechanism as a slight variation of Hebb’s rule.

Kohonen Self-organizing Map:

It is also known as Kohonen feature map or topology-preserving map or Kohonen Self-organizing .

Information is often represented spatially in the two-dimensional neuronal sheets in the brain, in both the cortex and subcortical structures. We have learnt about the somatosensory, motor and visual maps in the corresponding sensory cortices in the brain. A map, in its ordinary sense, denotes a two-dimensional representation of a real-world domain, such that nearby points in the domain are mapped onto nearby points in the map.

Due to this “adjacency-preserving” property, these maps are also called *topographic* maps.

Self-organizing maps (SOM) are models of the topographic maps of the brain, first proposed by Teuvo Kohonen.

The SOM model can be presented as an extension of the competitive learning model described in the previous section. It is constructed by adding a biologically-relevant feature that is not originally present in the competitive learning network.

A key property of the SOM is that nearby or similar inputs activate nearby neurons in the map. The competitive learning network does not have this property.

Consider a hypothetical competitive learning network with 3 output neurons. The input space is two-dimensional. The weight vectors w_1 , w_2 , w_3 lie on a line as shown in Fig., with w_1 in between w_2 and w_3 . Note that such an arrangement is possible since there is no relation between the spatial position of the weight vectors and their indices.

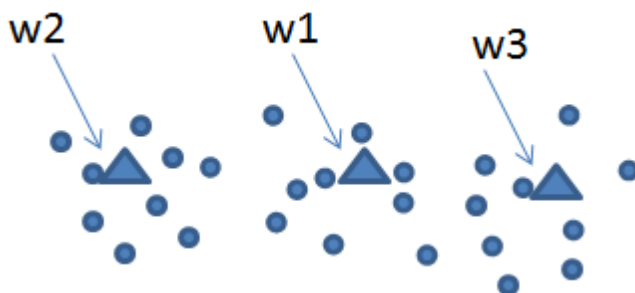


Fig. weight vectors and their indices when not related

The essence of the modification proposed in the SOM model, is a mechanism that ensures that the weight vectors remain spatially ordered, while they also move towards the data points that activate them maximally.

Unlike a competitive learning network, which consists of a single row of output neurons, a SOM consists of a m -dimensional grid of neurons. Usually two-dimensional SOMs are studied since SOMs were originally inspired by the two-dimensional maps in the brain. The topology of the grid is usually rectangular, though sometimes hexagonal topologies (Fig.) are also considered.

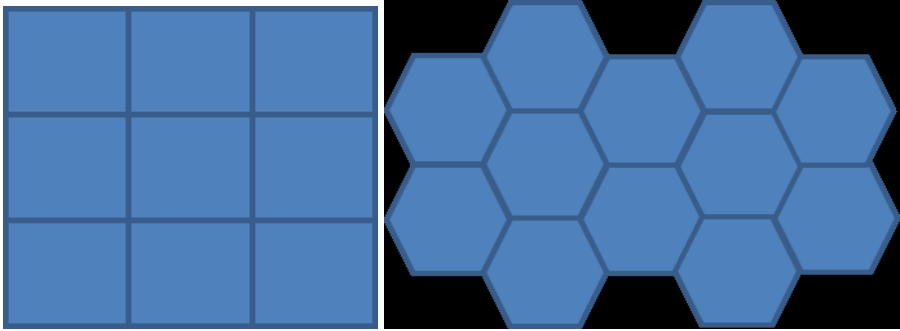


Figure: Rectangular and hexagonal trajectories of Kohonen's network

As in the case of competitive learning, the weight vector of the winner is moved towards the input, x . But addition, neurons close to the winner in the SOM are also moved towards the input, x , but with a lesser learning rate. Neurons that are nearby in the SOM are defined by a neighborhood N .

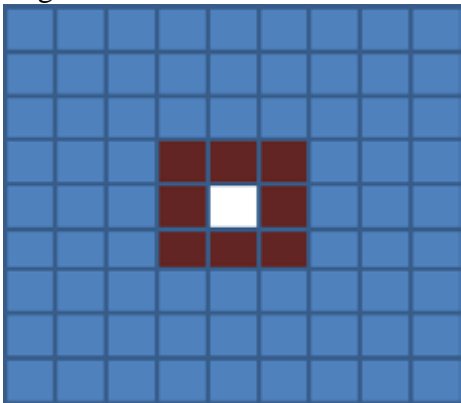


Fig. For the neuron in white (center) the neurons in red represent the neighborhood if we consider the neighborhood radius to be 1

Neighborhood size is large in the early stages, and is decreased gradually as training progresses.

Learning Vector Quantization(LVQ):

Vector quantization is nothing but clustering, where Given a set of vectors $\{x\}$, find a set of representative vectors $\{w_m; 1 \leq m \leq M\}$ such that each x is *quantized* into a particular w_m . $\{w_m\}$ locate at the mean (centroid) of the density distribution of each cluster. LVQ is an unsupervised pattern classifier where the actual class membership information is not used.

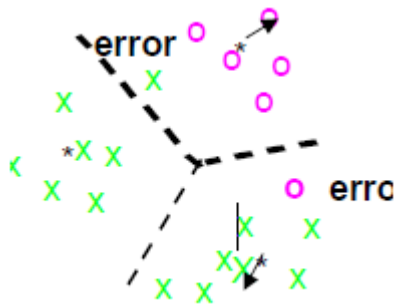


Fig. Clusters of data

Applications of LVQ:

Speech Recognition

- Robot Arm control
- Industrial process control
- automated synthesis of digital systems
- channel equalization for telecommunication
- image compression
- radar classification of sea-ice
- optimization problems
- sentence understanding
- classification of insect courtship songs

LECTURE-10

Linear neuron model: (Hebbian Learning)

Hebb described a simple learning method of synaptic weight change. In Hebbian learning, when 2 cells have strong responses, and fire simultaneously, their connection strength or weight increases. The weight increase is proportional to the frequency at which they fire together.

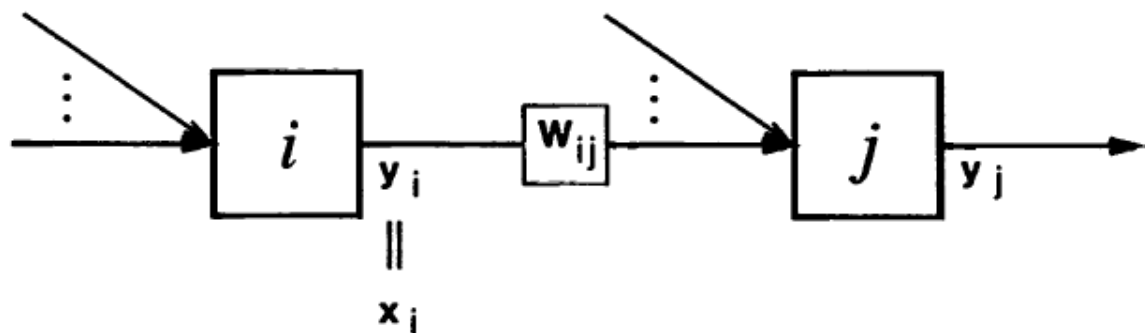


Fig. A simple network topology for Hebbian Learning, where W_{ij} resides between two neurons

$$\Delta w_{ij} = \eta f(\mathbf{w}_j^T \mathbf{x}) x_i$$

Where η is the learning rate, $f(\cdot)$ is the neuron function, \mathbf{x} is the input to the j th neuron. Since the weights are adjusted according to the correlation formula is a type of correlational learning rule.

A sequence of learning patterns indexed by p is presented to the network. Initial weights are taken zero. So updated weight after entire data set is:

$$w_{ij} = \eta \sum_p y_{ip} y_{jp}$$

Frequent input patterns have more impact on weights, giving largest output at end.

The objective function is maximized to maximize output.

$$J = \frac{1}{2} \sum_p y_p^2 = \frac{1}{2} \sum_p (\mathbf{x}_p^T \mathbf{w})^2$$

This rule causes unconstrained growth of weights. Hebbian rule was modified by Oja by normalization.

Modified Hebbian Learning:

$$w_i(t + 1) = \frac{w_i(t) + \eta y(t) x_i(t)}{\sqrt{\sum_{i=1}^n [w_i(t) + \eta y(t) x_i(t)]^2}}$$

For small learning rate expanding in Taylor's series weight update rule becomes

$$\Delta w_i = \eta y x_i - \eta y^2 w_i = \eta y (x_i - y w_i)$$

Here, a weight decay proportional to the squared output is added to maintain weight vector unit length automatically.

LECTURE-11

ANFIS: Adaptive Neuro-Fuzzy Inference Systems:

ANFIS are a class of adaptive networks that are functionally equivalent to fuzzy inference systems.

- ANFIS represent Sugeno & Tsukamoto fuzzymodels.
- ANFIS uses a hybrid learning algorithm

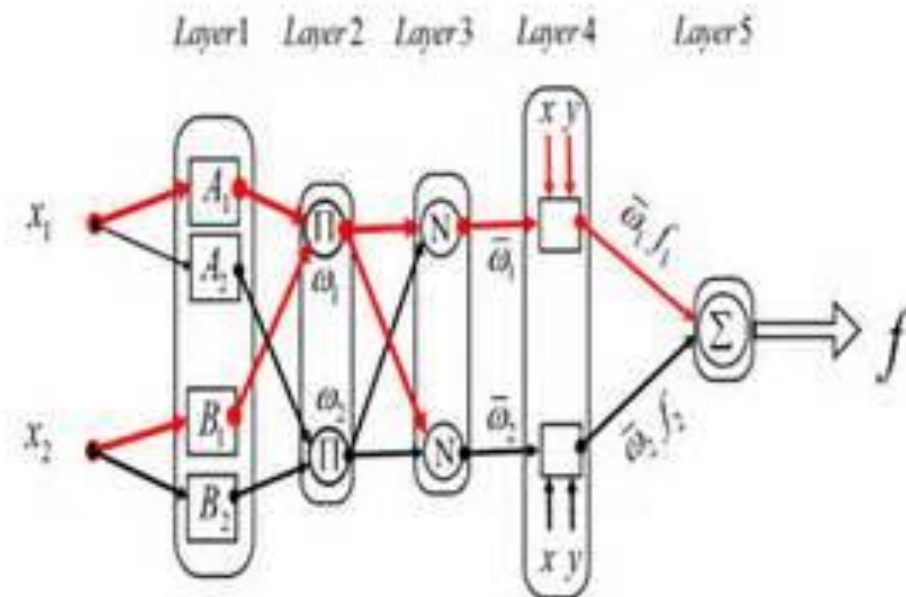


Fig. Architecture of ANFIS

\$O_{1,i}\$ is the output of the \$i^{th}\$ node of the layer 1.

- Every node \$i\$ in this layer is an adaptive node with a node function
 $O_{1,i} = \mu_{A_i}(x)$ for \$i = 1, 2\$, or $O_{1,i} = \mu_{B_{i-2}}(x)$ for \$i = 3, 4\$
- \$x\$ (or \$y\$) is the input node \$i\$ and \$A_i\$ (or \$B_{i-2}\$) is a linguistic label associated with this node
- Therefore \$O_{1,i}\$ is the membership grade of a fuzzy set \$(A_1, A_2, B_1, B_2)\$.

Typical membership function is Gaussian.

Every node in this layer is a fixed node labelled Prod.

- The output is the product of all the incoming signals.
- $O_{2,i} = w_i = \mu_{A_i}(x) \cdot \mu_{B_i}(y)$, \$i = 1, 2\$
- Each node represents the fire strength of the rule
- Any other T-norm operator that perform the AND operator can be used

Every node in this layer is a fixed node labelled Norm.

- The i th node calculates the ratio of the i th rule's firing strength to the sum of all rule's firing strengths.
- $O_{3,i} = w_i = w_{i1} + w_{i2}$, $i = 1, 2$
- Outputs are called normalized firing strengths.

Every node i in this layer is an adaptive node with a node function:

$$O_{4,i} = w_i f_i = w_i(p_i x + q_i y + r_i)$$

- w_i is the normalized firing strength from layer 3.
- $\{p_i, q_i, r_i\}$ is the parameter set of this node.
- These are referred to as consequent parameters.

The single node in this layer is a fixed node labeled sum, which computes the overall output as the summation of all incoming signals:

$$\text{overall output} = O_{5,1} = \sum_i p_i w_i = \sum_i p_i w_i$$

Hybrid Learning Algorithm:

The ANFIS can be trained by a hybrid learning algorithm presented by Jang in the chapter 8 of the book.

- In the forward pass the algorithm uses least-squares method to identify the consequent parameters on the layer 4.
- In the backward pass the errors are propagated backward and the premise parameters are updated by gradient descent.

| | Forward Pass | Backward Pass |
|-----------------------|-------------------------|------------------|
| Premise Parameters | Fixed | Gradient Descent |
| Consequent Parameters | Least-squares estimator | Fixed |
| Signals | Node outputs | Error signals |

Fig. Two passes in the hybrid learning algorithm for ANFIS.

Suppose that an adaptive network has L layers and the k th layer has $\#(k)$ nodes.

- We can denote the node in the i th position of the k th layer by (k, i) .
- The node function is denoted by O_{ki} .
- Since the node output depends on its incoming signals and its parameter set (a, b, c) , we have

$$O_i^k = O_{ki}^k(O_i^{k-1}, \dots, O_{\#(k-1)}^{k-1}, a, b, c)$$

- Notice that O_{ki} is used as both node output and node function. Assume that a training data set has P entries.
- The error measure for the p th entry can be defined as the sum of the squared error

$$E_p = \sum_{m=1}^{\#(L)} (T_{m,p} - O_{m,p}^L)^2$$

$T_{m,p}$ is the m th component of the p th target.

- $O_{m,p}^L$ is the m th component the actual output vector.
- The overall error is

$$E = \sum_{p=1}^P E_p$$

In order to implement the gradient descent in E we calculate the error rate ∂E ∂O for the pth training data for each node output O.

- The error rate for the output node at (L, i) is

$$\frac{\partial E_p}{\partial O_{i,p}^L} = -2(T_{i,p} - O_{i,p}^L)$$

For the internal node at (k, i), the error rate can be derived by the chain rule:

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k}$$

where $1 \leq k \leq L - 1$

- The error rate of an internal node is a linear combination of the error rates of the nodes in the next layer.

Consider α one of the parameters.

- Therefore

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha}$$

where S is the set of nodes

- The derivative of the overall error with respect to α is

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^P \frac{\partial E_p}{\partial \alpha}$$

The update formula for $\Delta \alpha$ is

$$\Delta \alpha = \eta \frac{\partial E}{\partial \alpha}$$

If the parameters are to be updated after each input-output pair (on-line training) then the update formula is:

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha}$$

With the batch learning (off-line learning) the update formula is based on the derivative of the overall error with respect to α :

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^P \frac{\partial E_p}{\partial \alpha}$$

Problems of the gradient descent are:

The method is slow.

- It is likely to be trapped in local minima.

Hybrid Learning Rule:

Combines:

- the gradient rule;
- the least squares estimate.

Consider that the adaptive network has only one output.

- output = F(I, S)
- I is the vector of input variables.
- S is the set of parameters.
- F is the function implemented by the ANFIS.
- If there exists a function H such that the composite function H ◦ F is linear in some elements of S then these elements can be identified by LSM.

More formally, if the parameter set S can be decomposed into two sets $S = S1 \oplus S2$ (\oplus direct sum), such that H ◦ F is linear in the elements of S2

- then applying H to output = F(I, S) we have H(output) = H ◦ F(I, S) (7) which is linear in the elements of S2.
- Given values of elements of S1, it is possible to plug P training data in equation 7.
- As a result we obtain a matrix equation $A_{-} = y$ where $-$ is the unknown vector whose elements are parameters in S2.
- This is the standard linear least-square problem.

Combining LSE and gradient descent:

- forward pass

In batch mode, each epoch is composed of a forward pass and a backward pass.

- In the forward pass an input vector is presented and the output is calculated creating a row in the matrices A and y.
- The process is repeated for all training data and the parameters S2 are identified by BLS or RLS.
- After S2 is identified the error for each pair is computed.

Combining LSE and gradient descent:

- backward pass

The derivative of the error measure with respect to each node output propagate from the output toward the input.

- The derivatives are:

$$\frac{\partial E_p}{\partial O_{i,p}^L} = -2(T_{i,p} - O_{i,p}^L)$$

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k}$$

The parameters in S2 are updated by the gradient method

$$\Delta \alpha = -\eta \frac{\partial E}{\partial \alpha}$$

Applications of ANFIS:

1. Printed Character recognition
2. Inverse Kinematics
3. Nonlinear System identification
4. Channel Equalization

5. Feed back control system
6. Adaptive noise cancellation

References:

5. Chapter 8,9,11,12,19 J.S.R.Jang, C.T.Sun and E.Mizutani, "*Neuro-Fuzzy and Soft Computing*", PHI, 2004, Pearson Education 2004.
6. Chapter 2, 3 & 7 S. Rajasekaran & GA Vijayalakshmi Pai "Neural Networks, Fuzzy Logic, and Genetic Algorithms synthesis and application", PHI
7. Chapter 2& 3 Stamatios V. Kartalopoulos "Understanding Neural Networks and Fuzzy Logic Basic concepts & Applications", IEEE Press, PHI, New Delhi, 2004.
8. Internet sources.

MODULE-III (10 HOURS)

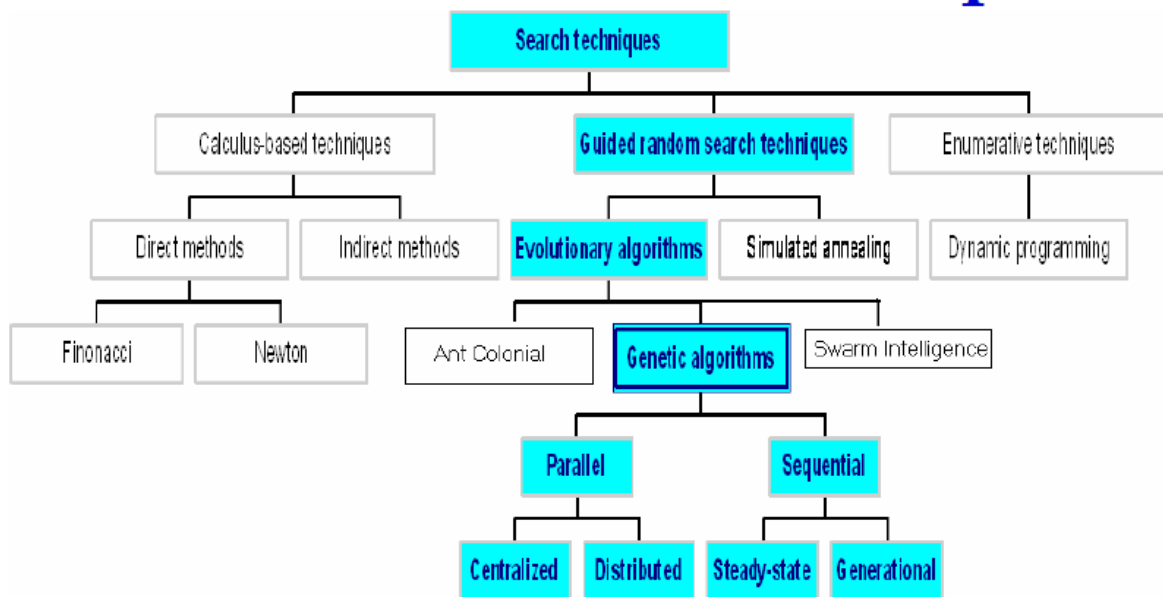
Derivative-free Optimization Genetic algorithms: Basic concepts, encoding, fitness function, reproduction. Differences of GA and traditional optimization methods. Basic genetic programming concepts Applications.

LECTURE-1

Introduction:

Most real world optimization problems involve complexities like discrete, continuous or mixed variables, multiple conflicting objectives, non-linearity, discontinuity and non-convex region. The search space (design space) may be so large that global optimum cannot be found in a reasonable time. The existing linear or nonlinear methods may not be efficient or computationally inexpensive for solving such problems. Various stochastic search methods like simulated annealing, evolutionary algorithms (EA) or hill climbing can be used in such situations. EAs have the advantage of being applicable to any combination of complexities (multi-objective, non-linearity etc) and also can be combined with any existing local search or other methods. Various techniques which make use of EA approach are Genetic Algorithms (GA), evolutionary programming, evolution strategy, learning classifier system etc.

Classes of Search Techniques



The principle of Darwinian evolution theory i.e., *survival of the fittest is evaluated by a fitness function* derived from objective function. Every individual in a population searches to be the best according to a fitness function in its own way (randomly).

Basic Concepts:

Optimization means to make the objective function max or min. That means in evolutionary computing where the individuals/ elements represent possible solutions, an element exists such that the fitness of that element is the maximum or minimum among all others' fitness depending on it is maximization or minimization problem.

Optimization can be classified as:

1. **Deterministic**-Uses derivative or gradient to reach final solution

2. **Stochastic**- Derivative free optimization, a type of random search, suitable for non-linearity, discontinuity escape from local optima and non-convex region

Components of Genetic Algorithm:

The individuals are genes which encode a trait or a parameter. The design space is to be converted to genetic space. It is parallel processing by a population used when single point approach of traditional methods cannot find a possible solution within the required time frame.

Important common aspects of evolutionary/swarm optimization algorithms: It is an iterative process where best solution is searched by a population in search space evaluating a fitness function.

1. **Search space**-Space for all feasible solutions is called search space.
2. **Solution**- It is the point with maximum or minimum value of fitness function.
3. **Fitness function**- A function derived from objective function
4. **Population size**- A number of points in a search space used in parallel for computing is called population, generally ranging from 30 to 200.
5. **Constraints**- Lower and upper bounds
6. **Stopping criteria**- it can be no. of iterations, or minimum value of error in fitness or minimum improvement from previous iteration

LECTURE-2

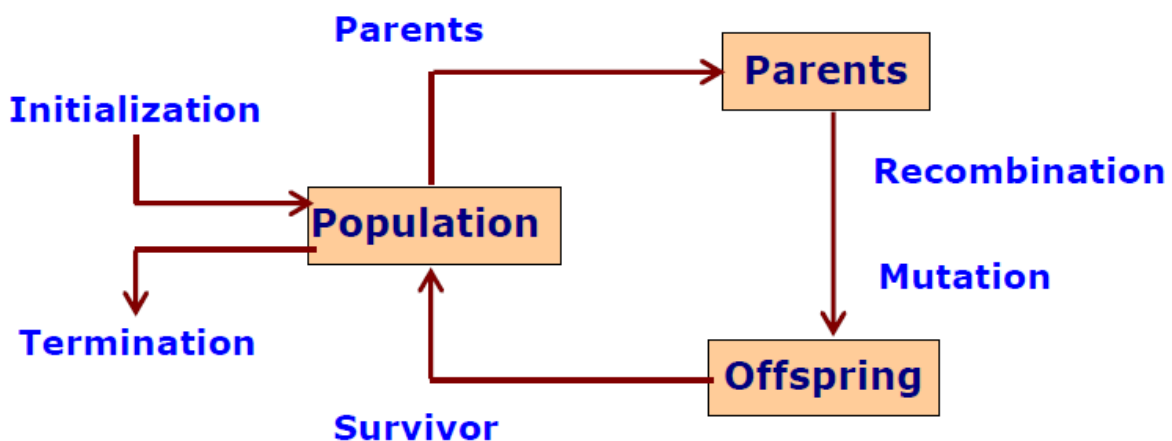


Fig. Basic cycle of EA

Basic flow chart of EA:

The initial population is usually generated randomly in all EAs. The termination condition may be a desired fitness function, maximum number of generations etc. In selection, individuals with better fitness functions from generation ' i ' are taken to generate individuals of ' $i+1$ 'th generation. New population (*offspring*) is created by applying *recombination* and *mutation* to the selected individuals (*parents*). Recombination creates one or two new individuals by swapping (crossing over) the genome of a *parent* with another. Recombined individual is then *mutated* by changing a single element (genome) to create a new individual. Finally, the new population is evaluated and the process is repeated. Each step is described in more detail below with special reference to GA. GA was proposed by Davis E. Goldberg.

```

BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END

```

Pseudocode of Genetics Algorithm

- Choose the initial population of individuals
- Evaluate the fitness of each individual in population
- Repeat until termination condition satisfied:
 - Selection: Select the individuals with greater fitness for reproduction
 - Crossover: Breed new individuals through crossover
 - Mutation: Apply probabilistic mutation on new individuals
 - Form a new population with these offsprings.
- Terminate

LECTURE-3

Special features of GA:

Encoding-

Objects forming possible solution sets to the original problem is called *phenotype* and the encoding (representation) of the individuals in the EA is called *genotype*. In GA each possible solution is coded in to genetic space. The coding may be binary coding, real coding, hexadecimal coding, value coding and tree coding.

Binary coding:

If each design variable is given a string of length ' l ', and there are n such variables, then the design vector will have a total string length of ' nl '. For example, let there are 3 design variables and the string length be 4 (not necessarily fixed for all problems, depends on accuracy in representing variable) for each variable. The variables are $x_1=4, x_2=7$ & $x_3=1$. Then the *chromosome* length is 12, where 4 bit in binary representing $x_1=0100$, $x_2=0111$, $x_3=0001$ are genes. So each string/ chromosome represents a different solution.

An individual consists a genotype and a fitness function. *Fitness* represents the quality of the solution (normally called *fitness function*). It forms the basis for selecting the individuals and thereby facilitates improvements.

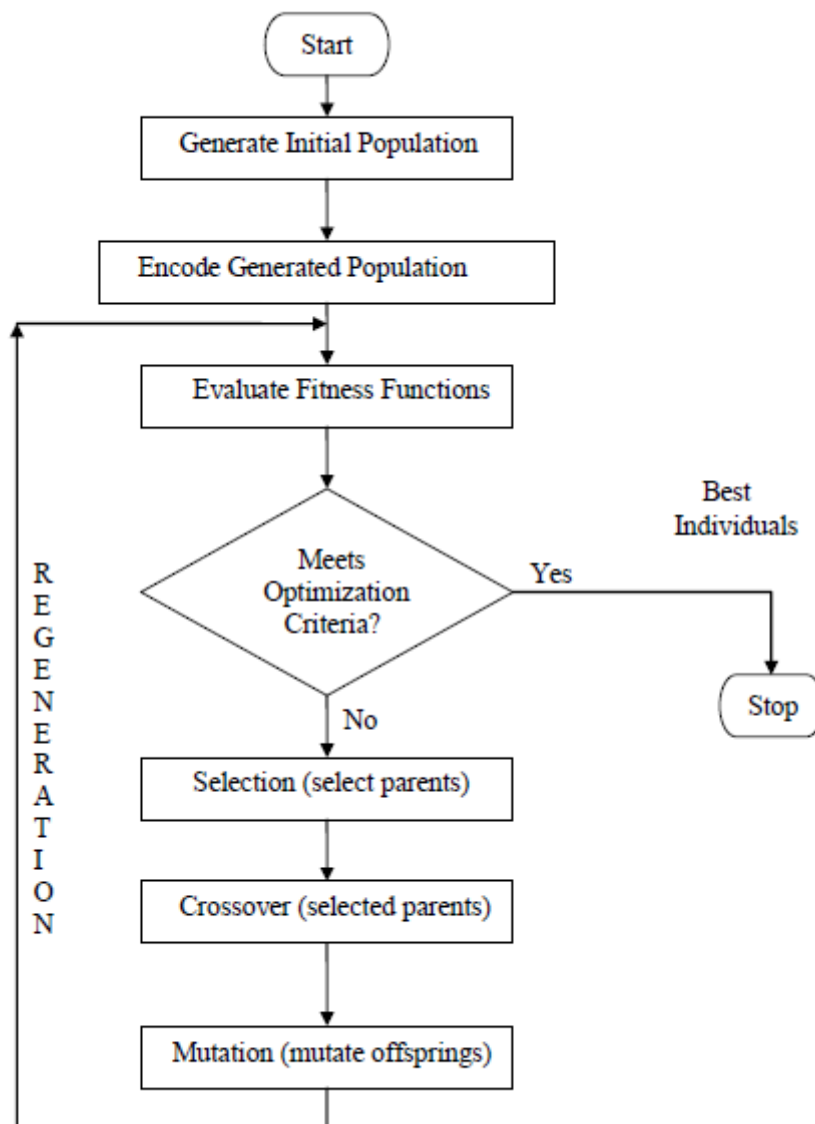


Fig. Flow chart of GA

Decoding:

If x_i^L & x_i^U correspond to 0000 to 0111 ; n_i the bit length of coding decoded value x_i will be

$$x_i = x_i^L + \frac{(x_i^U - x_i^L)}{(2^{n_i} - 1)} \times (\text{decoded value of string})$$

LECTURE-4

PARENT SELECTION:

After fitness function evaluation, individuals are distinguished based on their quality. According to Darwin's evolution theory the best ones should survive and create new offspring for the next generation. There are many methods to select the best chromosomes.

1. Roulette wheel selection
2. Boltzmann selection
3. Tournament selection
4. Rank selection
5. Steady state selection

The first one is briefly described.

Roulette Wheel Selection: Parents are selected according to their fitness i.e., each individual is selected with a probability proportional to its fitness value. In other words, depending on the percentage contribution to the total population fitness, string is selected for mating to form the next generation. This way, weak solutions are eliminated and strong solutions survive to form the next generation. For example, consider a population containing four strings shown in the Table 1. Each string is formed by concatenating four substrings which represents variables a,b,c and d. Length of each string is taken as four bits. The first column represents the possible solution in binary form. The second column gives the fitness values of the decoded strings. The third column gives the percentage contribution of each string to the total fitness of the population. Then by "Roulette Wheel" method, the probability of candidate 1 being selected as a parent of the next generation is 28.09%. Similarly, the probability that the candidates 2, 3, 4 will be chosen for the next generation are 19.59, 12.89 and 39.43 respectively. These probabilities are represented on a pie chart, and then four numbers are randomly generated between 1 and 100. Then, the likeliness that the numbers generated would fall in the region of candidate 2 might be once, whereas for candidate 4 it might be twice and candidate 1 more than once and for candidate 3 it may not fall at all. Thus, the strings are chosen to form the parents of the next generation.

Table 1

| Candidate | Fitness value | Percentage of total fitness |
|---------------------|---------------|-----------------------------|
| 1011 0110 1101 1001 | 109 | 28.09 |
| 0101 0011 1110 1101 | 76 | 19.59 |
| 0001 0001 1111 1011 | 50 | 12.89 |
| 1011 1111 1011 1100 | 153 | 39.43 |
| Total | 388 | 100 |

The one with higher probability p_i calculated is judged as per

$$p_i = F_i / \left(\sum_{j=1}^n F_j \right)$$

where F_i the fitness and n is the number of chromosome.

Choice of Methods of selecting chromosomes to be parents depends on

1. Population diversity- good ones exploited but new areas are explored but slow
2. Selective pressure-better individuals are preferred but chance of getting local max/min but fast

Generation:

Population of design vector obtained after one computation on all individuals.

Generation gap:

Proportion of individuals in the population that are replaced in each generation. It should be increased to improve results.

Reproduction:

Generally after choosing a method of selection population with above average fitness function in one generation are taken as parents and inserted in next population in multiple(clones) so that good trait is transferred to child.

LECTURE-5

CROSSOVER:

It is a recombination operator. Selection alone cannot introduce any new individuals into the population, i.e., it cannot find new points in the search space. These are generated by genetically-inspired operators, of which the most well known are *crossover* and *mutation*.

Types-

1. One-point
2. Two-point
3. Uniform
4. Arithmetic
5. Heuristic
6. Matrix

In one point crossover, selected pair of strings is cut at some random position and their segments are swapped to form new pair of strings.

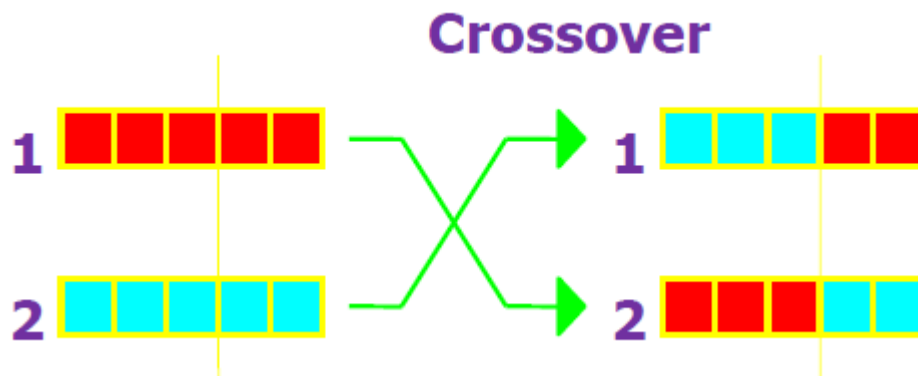


Fig. One point crossover

In two-point scheme, there will be two break points in the strings that are randomly chosen. At the break-point, the segments of the two strings are swapped so that new set of strings are formed. For example, let us consider two 8-bit strings given by '10011101' and '10101011'. Then according to one-point crossover, if a random crossover point is chosen after 3 bits from left and segments are cut as shown below: 100 | 11101 101 | 01011 and the segments are swapped to form 10001011 10111101 According to two-point crossover, if two crossover points are selected as 100 | 11 | 101 101 | 01 | 011 Then after swapping both the extreme segments, the resulting strings formed are 10001101 10111011 Crossover is not usually applied to all pairs of individuals selected for mating. A random choice is made, where the probability of crossover P_c being applied is typically between 0.6 and 0.9.

MUTATION:

Mutation is applied to each child individually after crossover. This creates diversity. It randomly alters each gene with a small probability P_m (generally not greater than 0.01).

Types-

1. Flip-bit
2. Boundary
3. Uniform
4. Non- Uniform
5. Gaussian

It injects a new genetic character into the chromosome by changing at random a bit in a string depending on the probability of mutation. Example: 10111011 is mutated as 10111111 It is seen in the above example that the sixth bit '0' is changed to '1'. Thus, in mutation process, bits are changed from '1' to '0' or '0' to '1' at the randomly chosen position of randomly selected strings. The mutation may be for one individual or a group.

| | |
|----------------------|---------------------------------|
| Original offspring 1 | 1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 0 |
| Original offspring 2 | 1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0 |
| Mutated offspring 1 | 1 1 0 0 1 1 1 0 0 0 0 1 1 1 1 0 |
| Mutated offspring 2 | 1 1 0 1 1 0 1 1 0 0 1 1 0 1 0 0 |

Fig. Mutation bits shown in red

LECTURE-6

ADVANTAGES AND DISADVANTAGES OF EA:

EA can be efficiently used for highly complex problems with multi-objectivity, non-linearity etc. It provides not only a single best solution, but the 2nd best, 3rd best and so on as required. It gives quick approximate solutions. EA methods can very well incorporate with other local search algorithms. There are some drawbacks also in using EA techniques. An optimal solution cannot be ensured on using EA methods, which are usually known as heuristic search methods. Convergence of EA techniques are problem oriented. Sensitivity analysis should be carried out to find out the range in which the model is efficient. Also, the implementation of these techniques requires good programming skill.

Differences and similarities between GA and other traditional methods:

Differences:

1. GA uses coding which discretizes search space even though function is continuous
2. A discrete function can be handled with no extra cost
3. Works with a population of points instead of single, so multiple optimal solutions can be captured at a time, reducing no. of run of algorithm

Similarities:

1. Search direction in traditional algorithms is used to find new point, where as 2 points are used to define search direction a in crossover in GA
2. Search direction is not fixed for all points, as mutation works in GA

New Variants of GA:

According to the encoding, selection, crossover or mutation methods and adaptive changing of the probabilities with convergence many variants like continuous GA, binary GA, RCGA, NSGA etc have been developed. In addition the new ones are

1. Messy GA
2. Parallel GA
3. Multiobjective GA

LECTURE-7

Issues for GA Parameter settings:

Choosing basic implementation issues:

1. Representation
2. Population size, mutation rate, ...
3. Selection, deletion policies
4. Crossover, mutation operators
5. Termination Criteria
6. Performance, scalability
7. Solution is only as good as the evaluation function (often hardest part)

Benefits of Genetic Algorithms

1. Concept is easy to understand
2. Modular, separate from application
3. Supports multi-objective optimization
4. Good for “noisy” environments
5. Always gives answer; answer gets better with time
6. Inherently parallel; easily distributed
7. Multiple ways to speed up and improve a GA-based application as knowledge about problem domain is gained
8. Easy to exploit previous or alternate solutions
9. Flexible building blocks for hybrid applications
10. Substantial history and range of use

Shortcomings of GA:

1. **Minimal deception problem-** Some objective functions may be very difficult to optimize by GA. **Representing the solution accuracy depends on coding.**
2. **GA drift(Bias)-** Loss of population diversity may seek suboptimal solution with a smaller population size
3. Real time& online issues- It does not guarantee response time, which is vital in real time issues. Works offline satisfactorily.
4. Computationally expensive and time consuming
5. Issues in representation of problem
6. Proper writing of fitness function
7. Proper values of size of population, crossover and mutation rate
8. Premature Convergence
9. No one mathematically perfect solution since problems of biological adaptation don't have this issue

LECTURE-8 to 10

Applications of Genetic Algorithms:

1. Optimization and design

- numerical optimization, circuit design, airplane design, factory scheduling, drug design, network optimization
2. Automatic programming
evolving computer programs (e.g., for image processing), evolving cellular automata
 3. Machine learning and adaptive control
robot navigation, evolution of rules for solving “expert” problems, evolution of neural networks, adaptive computer security, adaptive user interfaces
 4. Complex data analysis and time-series prediction
prediction of chaotic systems, financial-market prediction, protein-structure prediction
 5. Scientific models of complex systems
economics, immunology, ecology, population genetics, evolution, cancer

References:

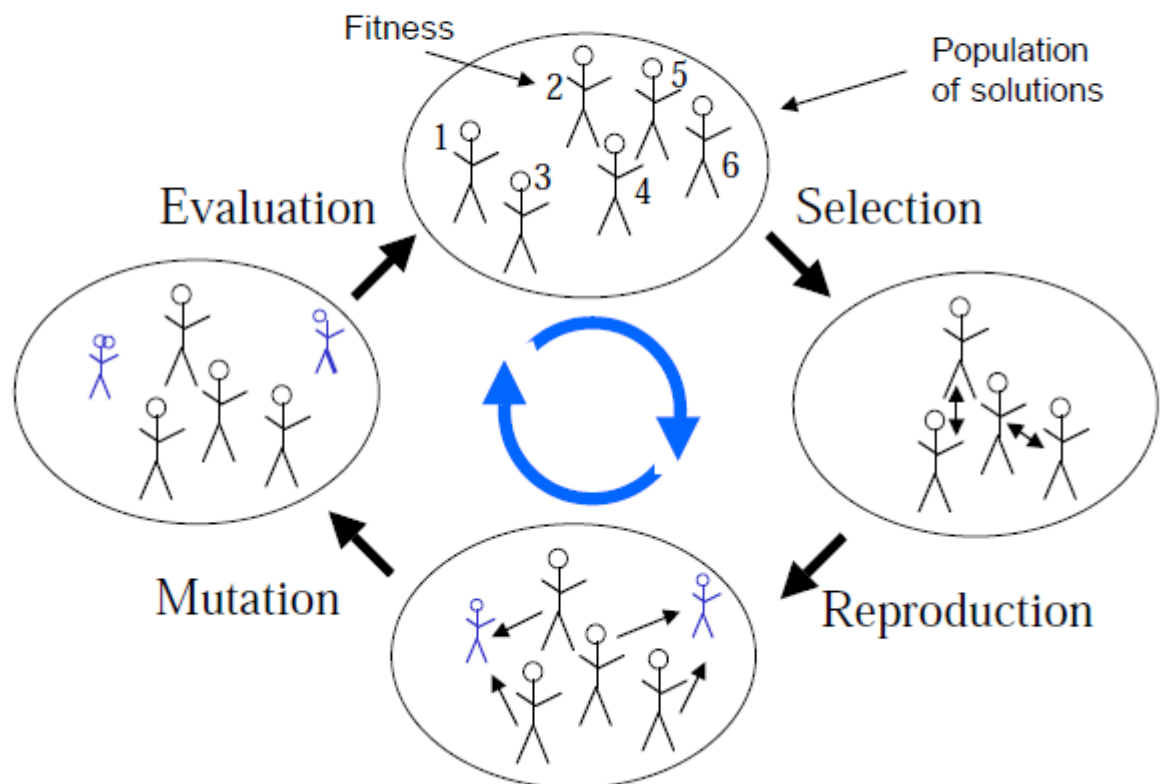
1. Chapter 7 J.S.R.Jang, C.T.Sun and E.Mizutani, “*Neuro-Fuzzy and Soft Computing*”, PHI, 2004, Pearson Education 2004.
2. Chapter 8 & 9 S. Rajasekaran & GA Vijayalakshmi Pai “Neural Networks, Fuzzy Logic, and Genetic Algorithms synthesis and application”, PHI
3. Internet sources

MODULE-IV (10 HOURS)

Evolutionary Computing, Simulated Annealing, Random Search, Downhill Simplex Search, Swarm optimization

LECTURE-1

Evolution and Evolutionary Algorithms



Drawback of traditional techniques

Computing tasks have to be

- well-defined
- fairly predictable
- computable in reasonable time with serial computers

Analogy-based algorithms

For any natural phenomenon you can think of, there will be at least one AI research group that will have a combinatorial optimization algorithm “based” on “analogies” and “similarities” with the phenomenon. Here’s the beginning of the list...

- Metal cooling annealing
- Evolution / Co-evolution / Sexual Reproduction
- Thermodynamics
- Societal Markets
- Management Hierarchies
- Ant/Insect Colonies
- Immune System
- Animal Behavior Conditioning

- Neuron / Brain Models
- Hill-climbing (okay, that's a stretch...)
- Particle Physics

LECTURE-2

Simulated Annealing

1. Let $X :=$ initial config
2. Let $E := \text{Eval}(X)$
3. Let $i =$ random move from the moveset
4. Let $E_i := \text{Eval}(\text{move}(X,i))$
5. If $E < E_i$ then
 - $X := \text{move}(X,i)$
 - $E := E_i$
 - Else with some probability, accept the move even though things get worse:
 - $X := \text{move}(X,i)$
 - $E := E_i$
6. Goto 3 unless bored.

Simulated Annealing

1. Let $X :=$ initial config
2. Let $E := \text{Eval}(X)$
3. Let $i =$ random move from the moveset
4. Let $E_i := \text{Eval}(\text{move}(X,i))$
5. If $E < E_i$ then
 $X := \text{move}(X,i)$
 $E := E_i$
 Else with some probability,
 accept the move even though
 things get worse:
 $X := \text{move}(X,i)$
 $E := E_i$
6. Goto 3 unless bored.

This may make the search fall out of mediocre local minima and into better local maxima.

How should we choose the probability of accepting a worsening move?

- *Idea One.* Probability = 0.1
- *Idea Two.* Probability decreases with time
- *Idea Three.* Probability decreases with time, and also as $E - E_i$ increases.

Simulated Annealing

If $E_i \geq E$ then definitely accept the change.

If $E_i < E$ then accept the change with probability

$$\exp(-(E - E_i)/T_i)$$

(called the Boltzman distribution)

...where T_i is a “temperature” parameter that gradually decreases. Typical cooling schedule:

$$T_i = T_0 \cdot r^i$$

High temp: accept all moves (Random Walk)

Low temp: Stochastic Hill-Climbing

When enough iterations have passed without improvement, terminate.

This idea was introduced by Metropolis in 1953. It is “based” on “similarities” and “analogies” with the way that alloys manage to find a nearly global minimum energy level when they are cooled slowly.

Simulated Annealing Issues

- MoveSet design is critical. This is the real ingenuity – not the decision to use simulated annealing.
- Evaluation function design often critical.
- Annealing schedule often critical.
- It's often cheaper to evaluate an incremental change of a previously evaluated object than to evaluate from scratch. Does simulated annealing permit that?
- What if approximate evaluation is cheaper than accurate evaluation?
- Inner-loop optimization often possible.

Advantages of Simulated Annealing

1. Simulated annealing is sometimes empirically much better at avoiding local minima than hill-climbing. It is a successful, frequently-used, algorithm. Basic hill climbing algorithm is so prone to getting caught in local optimums.

This is because a hill climber algorithm will simply accept neighbour solutions that are better than the current solution. When the hill climber can't find any better neighbours, it stops.

2. Not much opportunity to say anything formal about it (though there is a proof that with an infinitely slow cooling rate, you'll find the global optimum).

LECTURE-2

Random search algorithms

Random search algorithms are useful for many ill-structured global optimization problems with continuous and/or discrete variables. Typically random search algorithms sacrifice a guarantee of optimality for finding a good solution quickly with convergence results in probability. Random search algorithms include simulated annealing, tabu search, genetic algorithms, evolutionary programming, particle swarm optimization, ant colony optimization, cross-entropy, stochastic approximation, multistart and clustering algorithms, to name a few. They may be categorized as global (exploration) versus local (exploitation) search, or instance-based versus model-based. However, one feature these methods share is the use of

probability in determining their iterative procedures. This article provides an overview of these random search algorithms, with a probabilistic view that ties them together. A random search algorithm refers to an algorithm that uses some kind of randomness or probability (typically in the form of a pseudo-random number generator) in the definition of the method, and in the literature, may be called a Monte Carlo method or a stochastic algorithm. The term metaheuristic is also commonly associated with random search algorithms.

Generic Random Search Algorithm

Step 0. Initialize algorithm parameters Θ_0 , initial points $X_0 \subset S$ and iteration index $k = 0$.

Step 1. Generate a collection of candidate points $V_{k+1} \subset S$ according to a specific generator and associated sampling distribution.

Step 2. Update X_{k+1} based on the candidate points V_{k+1} , previous iterates and algorithmic parameters. Also update algorithm parameters Θ_{k+1} .

Step 3. If a stopping criterion is met, stop. Otherwise increment k and return to Step 1.

This generic random search algorithm depends on two basic procedures, the generator in

Step 1 that produces candidate points, and the update procedure in Step 2.

LECTURE-3

Hill-climbing

Hill-climbing: Attempt to maximize $\text{Eval}(X)$ by moving to the highest configuration in our moveset. If they're all lower, we are stuck at a "local optimum."

1. Let $X := \text{initial config}$
2. Let $E := \text{Eval}(X)$
3. Let $N = \text{moveset_size}(X)$
4. For ($i = 0 ; i < N ; i := i + 1$)
 - Let $E_i := \text{Eval}(\text{move}(X, i))$
5. If all E_i 's are $\leq E$, terminate, return X
6. Else let $i^* = \text{argmax}_i E_i$
7. $X := \text{move}(X, i^*)$
8. $E := E_{i^*}$
9. Goto 3

(Not the most sophisticated algorithm in the world.)

Randomized Hill-climbing

1. Let $X :=$ initial config
2. Let $E := \text{Eval}(X)$
3. Let $i =$ random move from the moveset
4. Let $E_j := \text{Eval}(\text{move}(X,i))$
5. If $E < E_j$ then
 $X := \text{move}(X,i)$
 $E := E_j$
6. Goto 3 unless bored.

What stopping criterion should we use?

Any obvious pros or cons compared with our previous hill climber?

LECTURE-4

Swarm Optimization

Swarm intelligence (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial. SI systems consist typically of a population of simple agents or boids interacting locally with one another and with their environment. The inspiration often comes from nature, especially biological systems. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents. Examples in natural systems of SI include ant colonies, bird flocking, animal herding, bacterial growth, fish schooling and Microbial intelligence. It is infact a multi-agent system that has self-organized behaviour that shows some intelligent behaviour.

Two principles in swarm intelligence

self-organization is based on:

activity amplification by positive feedback

activity balancing by negative feedback

amplification of random fluctuations

multiple interactions

stigmergy - stimulation by work - is based on:

work as behavioural response to the environmental state

an environment that serves as a work state memory

work that does not depend on specific agents

Roots in models of social insects behavior:

- Foraging behavior
- Division of labor and task allocation
- Cemetery organization
- Nest building
- Collaborative transport

Properties of collective intelligence systems:

- Distributed computation
- Direct and **indirect** interactions
- Agents equipped with simple computational capabilities
- Robustness
- Adaptiveness

Multiple interactions among agents

Simple agents (e.g., rule based)

Systems composed of many agents

Positive feedback

Amplification of random fluctuations and structure formation

Reinforcement of most common behavior patterns

Negative feedback

Saturation

Competition

Resource exhaustion

Particle Swarm Optimization

- Population initialized by assigning random positions *and* velocities; potential solutions are then *flown* through hyperspace.
- Each particle keeps track of its “best” (highest fitness) position in hyperspace.
- This is called “pbest” for an individual particle
- It is called “gbest” for the best in the population
- It is called “lbest” for the best in a defined neighborhood
- At each time step, each particle stochastically accelerates toward its pbest and gbest (or lbest).

LECTURE-5

Ant Algorithm:

Algorithms inspired by the behavior of real ants

Examples:

- Foraging
- Corpse clustering
- Division of labor

While walking ants deposit a substance called **pheromone** on the ground They choose with higher probability paths that are marked by stronger pheromone concentrations

Cooperative interaction which leads to the emergence of short(est) paths

LECTURE-6

Bees Algorithm:

- The queen moves randomly over the combs eggs are more likely to be layed in the neighbourhood of brood
- honey and pollen are deposited randomly in empty cells
- four times more honey is brought to the hive than pollen
- removal ratios for honey: 0.95; pollen: 0.6
- removal of honey and pollen is proportional to the number of surrounding cells containing brood

The above are few examples of SI, there are numerous others.

LECTURE-7-10

Swarm Optimization Applications:

1. Combinatorial optimization
2. Mixed integer-continuous optimization
3. Networks: AntNet
4. Data clustering and exploratory data analysis
5. Coordinated motion
6. Self-assembling

No free Lunch Theorem:

A number of “no free lunch” (NFL) theorems are establish that for any algorithm, any elevated performance over one class of problems is offset by performance over another class. These theorems result in a geometric interpretation of what it means for an algorithm to be well suited to an optimization problem. Applications of the NFL theorems to information-theoretic aspects of optimization and benchmark measures of performance are also presented.

References:

1. Yamille del Valle, Ganesh Kumar Venayagamoorthy, Salman Mohagheghi, and Ronald G. Harley, “Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems, IEEE Transactions On Evolutionary Computation, Vol. 12, No. 2, April 2008, pp. 171-195.
2. Krause, J., Ruxton, G. D., & Krause, S. (2010). Swarm intelligence in animals and humans. *Trends in ecology & evolution*, 25(1), 28-34.
3. Internet sources.
4. Chapter 7 J.S.R.Jang, C.T.Sun and E.Mizutani, “*Neuro-Fuzzy and Soft Computing*”, PHI, 2004, Pearson Education 2004.