

Adaptive Neuro Fuzzy Inference System

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering
by

KUMMARIKUNTLA TEJA (Reg No. 37110390)

KUMMATHI SAI CHARAN REDDY (Reg No. 37110391)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

JEPPIAR NAGAR, RAJIV GANDHI SALAI, CHENNAI – 600 119

MARCH - 2021

DECLARATION

I **KUMMARIKUNTLA TEJA** hereby declare that the Project Report titled "**ADAPTIVE NEURO FUZZY INTERFERENCE SYSTEM**" is done by me under the guidance of **Mrs. R. Subhashini, M.E., Ph.D.**, Department of Computer Science and Engineering at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean**, School of Computing, **Dr. S. Vigneswari, M.E., Ph.D.,** and **Dr. L. Lakshmanan, M.E., Ph.D., Heads of the Department** of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **DR. T. SASIKALA. M.E., PH. D, DEAN** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Nearly two decades back nonlinear system identification consisted of several ad-hoc approaches that were restricted to a very limited class of systems. However, with the advent of the various soft computing methodologies like neural networks, fuzzy logic and genetic algorithm combined with modern structure optimization techniques, a wider class of systems can be handled at present. Complex systems may be of diverse characteristics and nature. These systems may be linear or nonlinear, continuous or discrete, time-varying or time-invariant, static or dynamic, short term or long term, central or distributed, predictable or unpredictable, ill or well defined. System outputs may be measurable or unmeasurable. Models of real systems are of fundamental importance in virtually all disciplines and hence there is a strong demand for advanced modeling, identification, and controlling schemes. This is because models help in system analysis which in turn helps to get a better understanding of the system for predicting or simulating a system's behavior. Also, system models facilitate the application and validation of advanced techniques for controller design. Development of new processes and analysis of the existing ones along with their optimization, supervision, fault detection, and component diagnosis are all based on the models of the systems. Making this viable via Python makes things effective with ease.

TABLE OF CONTENT

S.NO.	TITLE	PAGE.NO
	ABSTRACT	V
	LIST OF FIGURES	VII
	LIST OF TABLES	VIII
1	INTRODUCTION	01
	1.1 PROBLEM STATEMENT	01
	1.2 SYNOPSIS	01
	1.3 SCOPE OF THE PROJECT	03
	1.4 OBJECTIVE OF THE PROJECT	03
2	LITERATURE REVIEW	04
3	SYSTEM DESIGN AND METHODOLOGY	12
	3.1 FUZZY RULES AND REASONING	12
	3.2 FUZZY INFERENCE SYSTEMS	17
	3.3 MACHINE LEARNING	22
	3.4 SUPERVISED AND UNSUPERVISED LEARNING	25
4	SOFTWARE DEVELOPMENT METHODOLOGY	30
	4.1 ARTIFICIAL INTELLIGENCE	30
	4.2 ADVANCED FUZZY NETWORKS	38
5	RESULT AND DISCUSSION	43
	5.1 RESULT	43
6	CONCLUSION AND FUTURE ENHANCEMENT	44
	6.1 CONCLUSION	44
	6.2 FUTURE ENHANCEMENT	44
	REFERENCE	45
	APPENDIX	47
	SOURCE CODE	47
	SNAP SHOTS	57
	JOURNAL PAPER	58
	PLAGIARISM REPORT	

LIST OF FIGURES

S.NO	TITLE	PAGE.NO
1.2.1	Hard computing versus soft computing	01
2.1	Simple Venn diagram	04
2.2	The Family Set A contains different sets	06
2.3	Union of Sets A and B	08
3.1.1	Curve showing all the rating points as specific values	13
3.1.2	Support, core, and boundary	15
3.1.3	Open Left, closed, and open right sets	16
3.2.1	The process of a Fuzzy Inference System	17
3.2.2	Fuzzy Inference System process	19
3.2.3	The TSK method	21
3.3.1	Machine Learning algorithms	22
3.3.2	Binary classification	23
3.3.3	Learning graph (gradient descent)	25
3.4.1	Supervised versus unsupervised learning.	26
3.4.2	Overfitting vs. underfitting	28
3.4.3	Bias and variance	29
4.1.1	Computational graph of ANNs	31
4.1.2	Typical perceptron architecture	33
4.1.3	Typical multi-layered ANN architecture	36
4.1.4	Process of convolving through filters	37
4.2.1	Neuro-Fuzzy Inference System	39
4.2.2	Clustering	39
4.2.3	C-Means Clustering	40
	anfis_apoch vs error	57
	anfis_trained_vs_originated	57

LIST OF TABLES

S.NO	TITLE	PAGE.NO
1.2.1	Basic constituents of soft computing	02
1.2.2	Applying brakes represented using crisp sets	03

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

On a critical challenge of pressure control across different chambers, with a valve combination in middle, a controller with regular operations failing to maintain the balance, which indeed causes critical issues in and out of the system, taking the advanced technologies of Soft Computing, the risk can be much reduced, the fuzzy logic with neural networks can deliver better results. We are on a mission to make this diversified across various domains with a programmatic implementation for ease, we are doing it with Python.

1.2 SYNOPSIS:

When you have crisply defined data that is precise and easy to understand, applying hard computing to it is perfect. Hard computing is based on binary logic, classical sets, crisp (precise) systems and software, basic numerical analysis, etc. But when you try to apply this same approach to real-world problems that

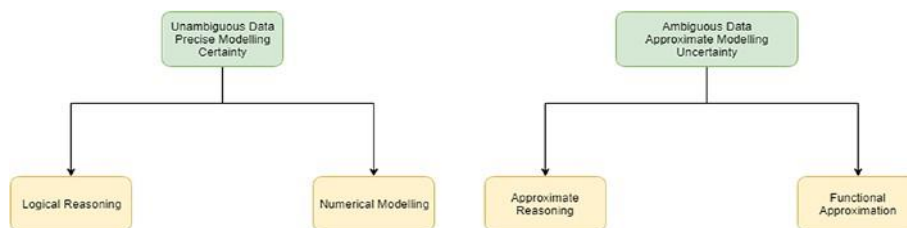


FIG 1.2.1 Hard computing versus soft computing

include imprecise data maybe the dataset is partially true, it has a lot of approximations, and so on hard computing fails. The best way to tackle this situation is to use the soft computing approach. A very basic example is $2+2$. In this scenario, you can use hard computing to arrive at 4. But when you change the equation to $2+x$, where x ranges from 0 to 5, soft computing always gives better results. Before you move on to understanding exactly what soft computing is, study the flowchart in Figure 1.2.1. It depicts the difference between hard and soft computing when it comes to problem solving.

Soft computing tries to imitate the human mind in order to make decisions. These models have cognitive abilities, which include:

- Ability to think.
- Ability to reason.
- Ability to organize.
- Ability to memorize.
- Ability to recognize.
- Ability to process.

When your data is *imprecise* (it has partial truths and is full of approximations), soft computing is the best approach. The following are features of soft computing-based problem-solving approaches:

- Biologically inspired.
- Fault tolerant.
- Full of optimizations.
- Helps make wiser and more intelligent machines.
- Helps in achieving robustness, tractability, and lower costs
- Heavy computation
- Goal driven

CONSTITUENTS	BENEFITS
Neural Networks	To learn and adapt based on uncertainty in data
Fuzzy Set Theory	Knowledge representation
Genetic Algorithm	For efficient searching
Traditional AI	Using mathematical approaches

Table 1.2.1. Basic Constituents of Soft Computing

Now that you have read about the basics of soft computing, turn your attention to understanding Fuzzy Systems. Fuzzy Systems are one of the core parts of soft computing, so you need to understand them. Fuzzy Systems, along with soft computing, make a very strong foundation for an inference system, also called an *Adaptive Neuro-Fuzzy Inference System*, which is discussed later in this book. Fuzzy Systems are comprised of Fuzzy Sets and not the normal, classical sets. In these systems, you try to follow Fuzzy Logic, because traditional logic cannot be applied to real-world applications. Let's look at Fuzzy Systems with the help of an example. When you drive a car, it's a combination of pressing the accelerator and the brake. Whenever you speed up, you press the accelerator, and whenever you want to slow down, you press the brakes. Suppose you are talking about self-driving cars. In this scenario, both systems should be managed simultaneously, without any manual intervention. If you consider traditional logic, it follows classical sets (called *crisp sets*). In these sets, the value is equal to 0 or 1. Suppose the functioning is defined as shown Table 1.2.2

Function	Set Code
Press the accelerator	1
Release the accelerator	0
Press the brakes	1
Release the brakes	0

Table 1.2.2: Applying Brakes Represented Using Crisp Sets

1.3. SCOPE:

Making the Neuro-Fuzzy Inference system much diversified with Python implementation can be a huge value added to the eco-system, with a common mission, the project can be widely used for various researches and Industrial implementation

1.4 OBJECTIVE:

The main goal is to enable ANFIS with Python, implementing the core logic behind ANFIS with Python programming language helps in various soft-computing researches.

CHAPTER 2

LITERATURE SURVEY

Classical sets, also called crisp sets, are a collection of objects. Objects can be anything belonging in the real world, and sometimes outside the domain as well.

For example:

Cars = {Audi, BMW, Mercedes, Porsche}

This set shows a list of premium cars. You denote a set by using the curly braces, {}.

Once you have effectively defined different sets, you can visualize them as well. A Venn diagram is a visual way to represent sets and their relationships with each other. Figure 1-2 shows a normal Venn diagram.

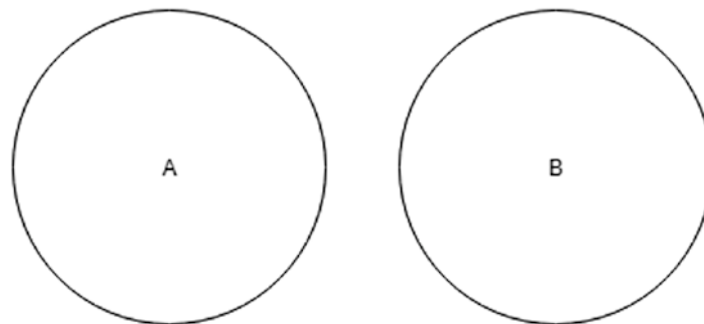


Figure 2-1. Simple Venn diagram

The circles in Figure 1-2 represent two sets, A and B. All the elements that are part of Set A will be present in Circle A, while all the elements that are part of Set B will be present in Circle B. The circles are called Venn diagrams of Set A and Set B.

Universe of Discourse

All the possible elements that can share a domain, or have similar characteristics, are contained in a set. The set of those elements is called the Universe of Discourse. Once you have this set, you can form various subsets as well. Formally speaking, the Universe of Discourse can be defined as follows:

Look at the Universe of Discourse with the following example:

$E = \{\text{Domain of Machine Learning}\}$

Let Set E be the Universe of Discourse. It has the complete domain of Machine Learning. Then the possible sets that can be part of E are:

- Set of Machine Learning algorithms
- Set of basic statistical methods
- Set of neural networks, etc.

Properties of Classical Sets

This section explains the various properties of classical sets. These are:

- Membership of elements
- Cardinality of sets
- Family of sets
- Null set
- Singleton set
- Subset
- Superset
- Powerset
- Countable set
- Uncountable set

Membership of Elements

If an element is a part of a set, it is called a member of the set. It is denoted by $X \in A$, which means that Element X is a member of Set A .

$A = \{1, 2, 3, 4, 5\}$

In the set of integers from 1 to 5, each number is called a member of Set A .

Cardinality of Sets

In a set, if you count the total number of elements present, that number is called the cardinality of the set. It can be denoted by $n(A)$ or $|A|$ or $\#A$.

$A = \{1, 2, 3, 4, 5\}$

This set has a cardinality of 5.

Family of Sets

A set can contain anything. But if a set contains a collection of different sets, then that set is referred to as a family of sets. For example:

$$A = \{(1,3,5), (2,4,6), (5,10,15)\}$$

Here, (1, 3, 5), (2, 4, 6), and (5, 10, 15) are individual sets contained in Set A. Therefore, Set A is the family of sets, as shown in Figure 1-3.

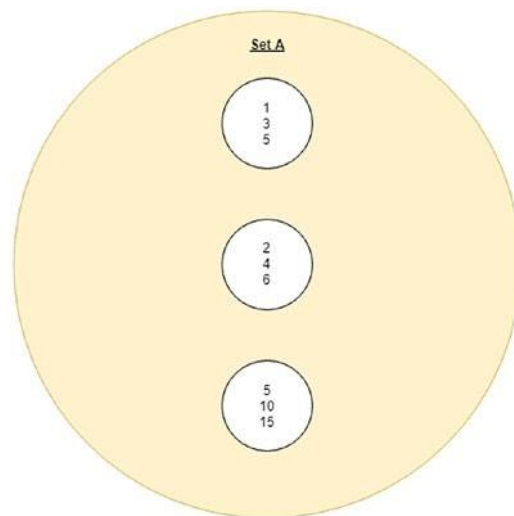


Figure 2.2. The Family Set A contains different sets

Null Set

If a set has a cardinality of 0, it's called a null set or empty set. That means that there are no elements inside it.

$A = \{\}$ is a null set.

Singleton Set

If a set has a cardinality of 1, it is called a singleton set. That means that there is only one element inside it.

$A = \{1\}$ is a singleton set.

Subset

Say you have two sets, A and X. If all the elements of X are part of A, then X is called a subset of A. It is represented by $X \subset A$. Figure 1-4 shows a Venn diagram representing Superset A containing Subset X.

Superset

Say you have two sets, A and X. If all the elements of X are part of A, then A is called the superset of X. It is represented by $A \supset X$. In the Figure 1-4, A is the superset of X.

Powerset

Say you have a set A. A set containing all the possible subsets from A, including the null set, is called the powerset. It is represented by $P(A)$. The cardinality of the powerset of A is $2^{|A|}$.

For example:

If $A = \{2010, 2011\}$

$P(A) = \{(), (2010), (2011), (2010, 2011)\}$

As you can see, $|A| = 2$, while $|P(A)| = 4$.

Countable Set

A countable set is where for every element can be labeled a unique natural number. Also, by the time you finish labeling the elements, more labels are left in the form of natural numbers, or otherwise you have exhausted all the natural numbers. Therefore, infinite sets can also be countable, but not every time.

Uncountable Set

An uncountable set is where, for every element present, you cannot label a unique natural number. That means, by the time you finish assigning labels, the natural numbers list is exhausted. For example, when you take real numbers, the list of natural numbers will be exhausted much before the labeling of a set of real numbers is finished.

Classical Set Operations

This section looks at some of the operations that can be applied to classical sets:

- Union
- Intersection

- Complement
- Difference

Union

A union of two sets merges the values in both sets into one single set.

$$A = \{1, 3, 5\}$$

$$B = \{2, 4\}$$

$$A \cup B = \{1, 2, 3, 4, 5\}$$

Figure 1-5 represents a union between two sets, A and B. The shaded region represents the union part.

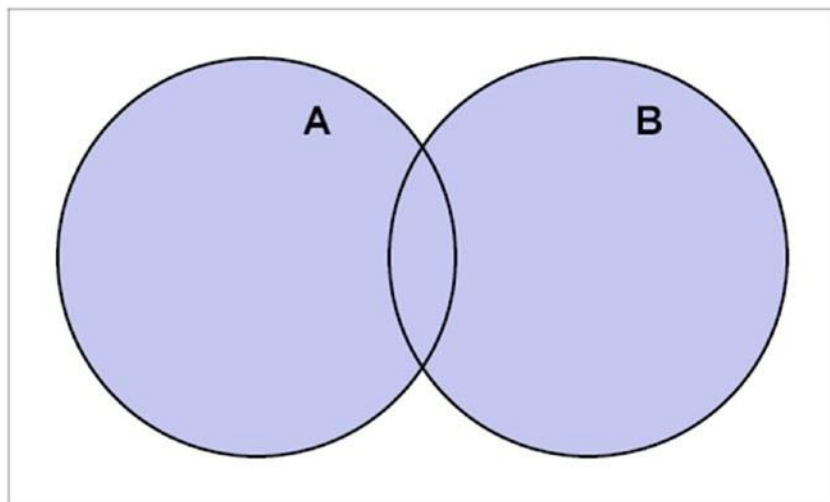


Figure 2.3. Union of Sets A and B

Intersection

An intersection of two sets finds the common values present in both sets and makes them one single set.

$$A = \{1, 3, 5\}$$

$$B = \{3\}$$

$$A \cap B = \{3\}$$

Complement

A complement of a set is all the values present in the Universe of Discourse, except the ones present in the set.

$$A = \{1, 3, 5\} \quad B = \{3\}$$

If A is the Universe of Discourse for B, then A^c will be all the values except the existing one. That is:

$$A^c = \{1, 5\}$$

Crisp Set Properties

This section discusses the following properties of classical/crisp sets:

- Law of Commutativity
- Law of Associativity
- Law of Distributivity
- Idempotent Law
- Identity Law
- Law of Absorption
- Involution Law
- Law of Transitivity
- Law of Excluded Middle
- Law of Contradiction
- De Morgan laws

Law of Commutativity

The union or intersection of two sets will produce the same the result, regardless of which set you list first. That means you can take A first or B first, but the results will always be the same.

$$(A \cup B) = (B \cup A)$$

$$(A \cap B) = (B \cap A)$$

Law of Associativity

If you have three sets, the Law of Associativity says that a union or intersection of the first two sets with the third one is equal to a union or intersection of the last two sets with the first one.

$$(A \cup B) \cup C = A \cup (B \cup C)$$

$$(A \cup B) \cap C = A \cup (B \cap C)$$

Law of Distributivity

The Law of Distributivity says that a union of the first set with an intersection of the last two sets is equal to a union of the first set with the second, a union of the first set with the third, and then an intersection of both outputs. This same rule applies when you interchange the union and intersection operations.

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

Idempotent Law

The Idempotent Law states that an intersection or union of a set with the same set is the same set itself.

$$A \cap A = A \quad A \cup A = A$$

Identity Law

If Φ is an empty set and E is a Universe of Discourse, then the Identity Law states that:

$$A \cap \Phi = \Phi \quad A \cap E = A \quad A \cup \Phi = A \quad A \cup E = E$$

In other words:

- A union of a set with an empty set leads to the same set
- An intersection of a set with an empty set leads to an empty set
- A union of a set with a Universe of Discourse leads to the Union of Discourse
- An intersection of a set with a Universe of Discourse leads to the same set

Law of Absorption

If A is a subset of B , then:

$$A \cap (A \cup B) = A \quad A \cup (A \cap B) = A$$

This means that the union of a set with its intersection with another set is the same set, and vice versa.

Involution Law

The Involution Law states that the double complement of a set is the same set:

$$(A^c)^c = A$$

Law of Transitivity

The Law of Transitivity states that if A is a subset of B and B is a subset of C, then A will be a subset of C.

If $A \subset B, B \subset C$, then $A \subset C$

Law of Excluded Middle

The Law of Excluded Middle states that a union of a set with its complement is its Universe of Discourse.

$$(A \cup A^c) = E$$

Law of Contradiction

The Law of Contradiction states that an intersection of a set with its complement is an empty set.

$$(A \cap A^c) = \emptyset$$

De Morgan Laws

If there are two sets and you find the complement of the union between them, then it is equal to the intersection of the complements of the individual sets. Similarly, if you find the complement of the intersection between the two sets, it is equal to the union of the complements of the individual sets. This rule is called the De Morgan Laws.

$$(A \cup B)^c = A^c \cap B^c$$

$$(A \cap B)^c = A^c \cup B^c$$

,

CHAPTER 3

SYSTEM DESIGN AND METHODOLOGY

3.1 Fuzzy Rules and Reasoning

The previous chapter discussed different kinds of sets as well as their properties and operations that can be performed on them. You also looked at some of the operations and their applications in Python. That chapter concluded with a small introduction to the different types of membership functions and their Python applications.

This chapter discusses the membership functions and their applications in detail. You will be looking at their diverse properties and operations. After understanding their roles, you will move on to Fuzzy Relations. You will learn what a Fuzzy Relation is and the properties that influence it.

After you have all the basic understanding required, you will finally move on to Fuzzy Rules and Reasoning, the core of Fuzzy Logic. You will learn about the different kinds of rules and how they are applicable. You will learn how to combine different kinds of rules, which will constitute Fuzzy Reasoning. You will see applications of these concepts in Python as well.

Membership Functions

Membership functions represent the degree of truth of a member in a defined Fuzzy Set. They are curves that define how each point in the input space is mapped to a degree of membership lying between 0 and 1. You may understand this better with the help of an example.

Suppose you want to rate the service of a particular restaurant. You might rate the service in the following ways:

- Awesome
- Average
- Worst

In classical sets, this can be represented as follows:

$X = \{\text{'Awesome'}, \text{'Average'}, \text{'Worst'}\}$

This can be coded and represented as $X = \{ \}$, where 2 represents Awesome, 1 represents Average, and 0 represents Worst.

But you might not want to rate the restaurant in only these three ways. You need different ways for customers to express their sentiments. Therefore, you could add these ratings as well:

- Awesome
- Nice
- Good
- Average
- OK
- Poor
- Worst

If you again use a classical set, it will contain a lot of code. Instead, you can define a function wherein each rating has a specific value. This function will allow you to go beyond the ratings. This function has an upper limit and a lower limit. Consider, for example, the sigmoid function (You learn about all the membership functions in detail, later in this chapter.) The sigmoid function has an upper limit of 1 and a lower limit of 0. That means that all the rating categories will have a value that will fall at a point on that curve (see Figure 2-1).

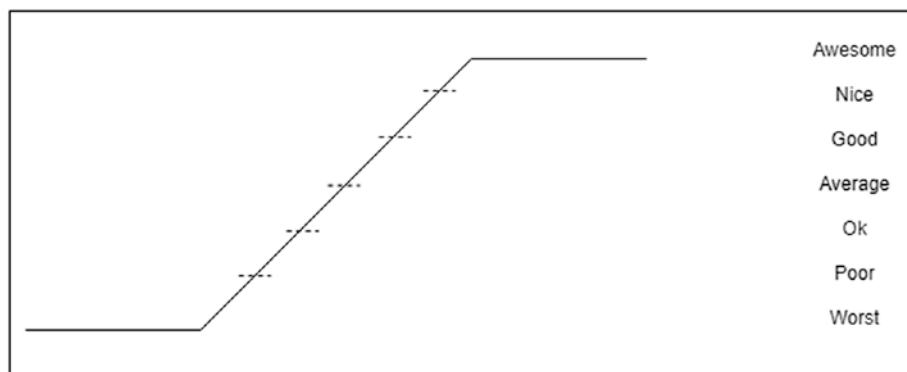


Figure 3.1.1. Curve showing all the rating points as specific values

Looking at this curve, you can redefine the crisp set as a Fuzzy Set having values between 0 and 1. Now, if a person gives a rating, the value (membership value) of that rating can be retrieved from the curve. This is what is meant when we say that membership functions represent the degree of truth of a member. You can see in this example that every rating has a value that tells about its degree of truth. Formal Definition of a Membership Function. You can write a formal definition of all the rating examples using Fuzzy Notations.

Suppose you have Fuzzy Set A with three members:

$$A = \{x_1, x_2, x_3\}$$

All three members will have a membership function value associated with them, which will define their degree of truth. It can be represented as shown here:

$$A = \{(x_1, \mu_A(x_1)), (x_2, \mu_A(x_2)), (x_3, \mu_A(x_3))\} \cup \{x_1, x_2, x_3\} \subseteq X$$

The higher the membership value, the higher the degree of belonging or truth inside the set. This process where every member of the Fuzzy Set gets a membership value associated with it is called Fuzzification. You can now extend the formal notation of the membership function value. If $\mu_A(x)$ is the membership value of elements and if $\mu_A(x)$ is equal to 1, you say that x is totally present in the Fuzzy Set A, or it has a full membership. If $\mu_A(x)$ is equal to 0, it is not part of A or it has no membership. Any value between 0 and 1 defines its membership, which can be termed part of the membership.

Terminology Related to Fuzzy Membership Functions

To understand Fuzzy Membership Functions in more detail, you must first understand some terminology related to them. This list of terminology will help you understand applications of membership functions better.

- Support
- Core
- Boundary
- Crossover
- Normality

- Fuzzy Singleton
- α - cut

Support

The support of a membership function for a Fuzzy Set is defined as that region of the universe that is characterized by nonzero membership in Set. It is a set of all the points whose membership value is greater than 0, as represented in Figure 3.1.2. Mathematically, it can be represented as follows:

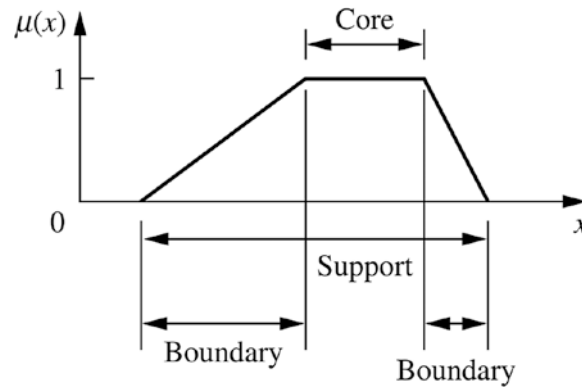


Figure 3.1.2 Support, core, and boundary

Convexity

A convex Fuzzy Set is described by a membership function whose membership values are strictly monotonically increasing, or whose membership values are strictly monotonically decreasing, or whose membership values are strictly monotonically increasing and then strictly monotonically decreasing, with increasing values for elements in the universe. In simpler terms, a Fuzzy Set is called convex if and only if it follows this rule:

$$m_A(\lambda x + (1 - \lambda)y) \geq \min\{m_A(x), m_A(y)\}$$

Bandwidth

If you have a set A, which is normal and convex, and if you find its crossover set and select two unique points, the distance between the two is called the bandwidth set. Simply speaking, for a normal and convex Fuzzy Set, the bandwidth is defined as the distance between the two unique crossover points:

$$|Bandwidth(A)| = x_2 - x_1 \text{ @ } m_A(x_1) = m_A(x_2) = 0.5$$

Symmetry

If the membership function of a Fuzzy Set A satisfies the following criteria, at a point c, it is called a symmetric set (see Figure 2-4).

$$m_A(x + c) = m_A(c - x)$$

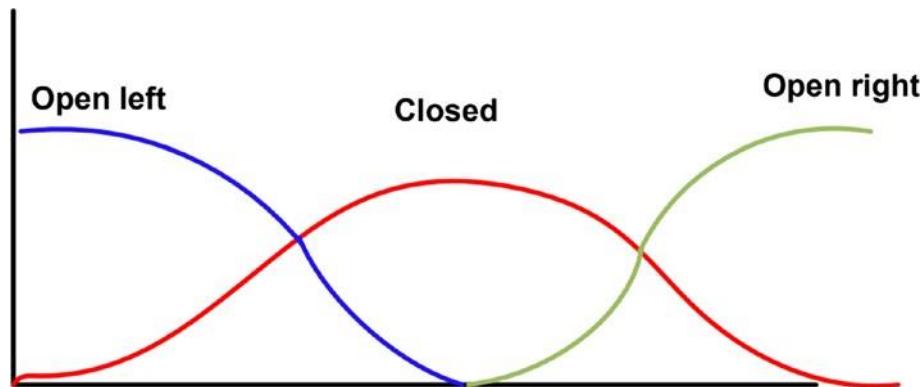


Figure 3.1.3 Open Left, closed, and open right sets

Properties of Fuzzy Relations

This section discusses the following properties of Fuzzy Relations:

- Projection
- Cylindrical extension
- Reflexive relation
- Anti-reflexive relation
- Symmetric relation
- Anti-symmetric relation
- Transitive relation
- Similarity relation
- Anti-similarity relation
- Weak similarity relation
- Order relation
- Pre-order relation
- Half order relation

3.2 Fuzzy Inference Systems

The previous two chapters explained the core concepts related to Fuzzy Logic. They discussed Fuzzy Sets and how they are different from the classical/crisp sets. You also learned about various operations that can be done on them and their properties. Then you learned about membership functions, which define the membership values of each element present in a Fuzzy Set. You learned about the different types of membership functions. Later, you learned about the Fuzzy Rules and reasoning approaches that utilize the concepts of membership functions to give various Fuzzy Solutions.

This chapter looks at real applications of all the concepts that you have learned so far. The chapter covers different types of Fuzzy Inference Systems, through which various real-life problems are solved in the industry. To understand these systems, you first need to understand the processes of Fuzzification and Defuzzification. You have already seen the Fuzzification process in the previous chapter, when you found the membership function values of each element of a set to make it a member of a Fuzzy Set. This chapter starts with the concept of Defuzzification and then moves on to different Fuzzy Inference Systems

Defuzzification is the process of converting a Fuzzy Set into a crisp set. You know that in most applications you have to use Fuzzy Sets, as people's opinions are never crisp. But when you incorporate these Fuzzy values and have to make a decision, you must convert the Fuzzy output into crisp values. Therefore, Defuzzification helps convert output given in a Fuzzy Set to crisp values. If control system functioning depends on input, the process of Defuzzification determines what exactly needs to be done once that input is provided. The general process of a Fuzzy System, of which Defuzzification is a part, is illustrated in Figure 3.2.1

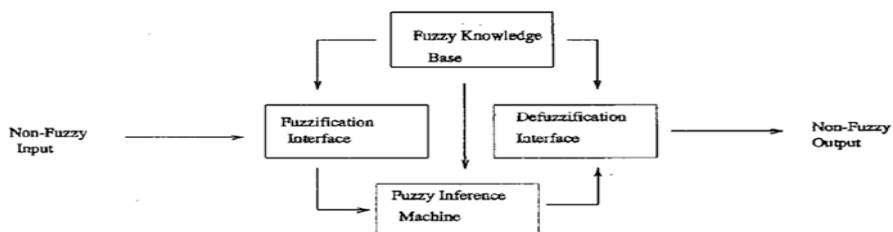


Figure 3.2.1. The process of a Fuzzy Inference System

Formally, you can define the Defuzzification process as follows:

“A Defuzzification method on a certain referential set V as a mapping from the class of fuzzy Subsets of V into V . A nonre- stricive coherence condition is that the associated point must belong to the support of the original fuzzy subset.”

There are different types of Defuzzification approaches. The most common ones are shown in Figure 3-2.

λ CUT METHOD:

Suppose you have a Fuzzy Set given by:

$$A = \{ \begin{matrix} 1 & 0.9 & 0.6 & 0.3 & 0.01 & 0 \end{matrix} \begin{matrix} a & b & c & d & e & f \end{matrix} \}$$

First/Last/Mean of Maximum Method

This method takes the union of all the possible Fuzzy Outputs and finds the smallest value with the maximum membership degree.

$$\text{First of Maximum} = z^* = z \wedge \inf \{z \in Z \mid m(z) = \text{hgt}(c)\} \quad \text{Last of Maximum} = z^* = z \wedge \sup \{z \in Z \mid m(z) = \text{hgt}(c)\}$$

$$\text{hgt}(c) = z \wedge \sup \{m(z)\}$$

In the previous equation, $\text{hgt}(c)$ represents the highest height present in the diagram of the union.

Fuzzy Inference Systems

When you have to design a system that is quite uncertain, one of the best approaches is using Fuzzy Inference Systems. Fuzzy Logic is used when you have a fixed set of rules and need to create systems based on that. But, when you add uncertainties inside the process, it requires some kind of inference of the process from the existing data. Using a Fuzzy Inference System is the way to infer those processes.

A Fuzzy Inference System (FIS) provides a way of mapping an input space to an output space with Fuzzy Logic. FIS tries to mimic the process with which humans solve any

problem statement using reasoning. FIS does that by using Fuzzy Logic, especially Fuzzy If-Then rules. Figure 3-17 represents the Fuzzy Inference System structure.

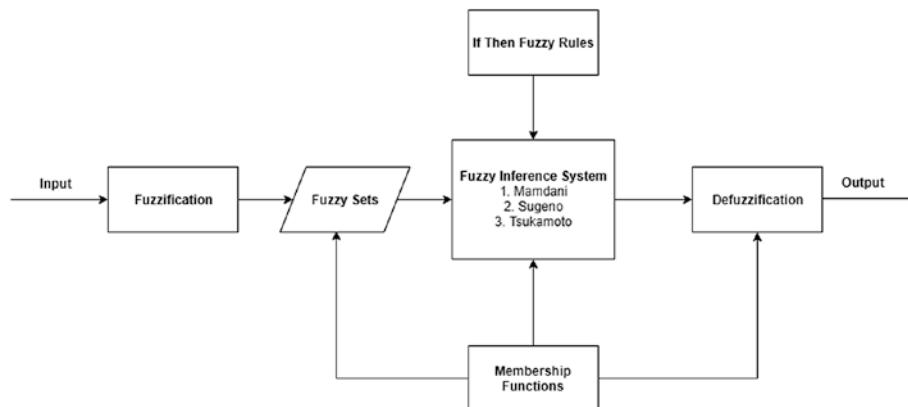


Figure 3.2.2. Fuzzy Inference System process

All the blocks in the diagram in Figure 3-17 are explained here:

- A database of all the Fuzzy If-Then Rules describing a system
- Database of membership functions
- Inference operations on Fuzzy Rules
- Defuzzification of Fuzzy Results into crisp outputs

When you combine all the rules and the membership functions database, it is called a knowledge base.

Now that you know the basic structure of the Fuzzy Inference System, you can look at some of its types. Fuzzy Inference Systems can be divided into three types:

- Mamdani model
- Takagi-Sugeno model
- Tsukamoto model

Mamdani Fuzzy Inference System

The Mamdani method is the most widely used Fuzzy Inference System. Because of its simple structure, it is used to solve all general decision- making problems. Mamdani FIS follows these general steps:

1. Step 1: Fuzzify the input.
2. Step 2: Find and evaluate the antecedent of each rule.
3. Step 3: Find the consequent of each rule.

4. Step 4: Aggregate the consequents.
5. Step 5: Defuzzify the results.

First, you convert the crisp inputs to Fuzzy Sets (aka, Fuzzification). For each input, you try to find the membership value. Suppose the antecedent has multiple parts. In that case, you use an aggregation operation like

T-Norm or T-Co-Norm to get a single membership value.

You can better understand this with the help of an example. Suppose the rulebase says the following: “If the product reviews are excellent or the product is beautiful.” You can divide the product reviews and the product aesthetics in a rating between 1 and 5. A review of 1 means poor and a review of 5 means excellent. For the product, 1 means a bad design and 5 means an awesome design. Now that you have defined the preliminaries, you can look at the first part: Fuzzification.

If you use sigmoid membership function.

Suppose you got input about the review as 1, and input about the design as 4. If you apply membership functions over the items, you may get 0.0 in the sigmoid curve for reviews and 0.7 in the sigmoid curve for the product.

Takagi-Sugeno-Kang Fuzzy Inference System

Takagi-Sugeno-Kang Fuzzy Inference Systems are used to model complex non-linear systems. The entire process of applying a Fuzzy Operator and then Fuzzifying the inputs is the same as with the Mamdani approach. The only change comes in the output membership function, which is either linear or constant. This section looks at the TSK approach.

Whatever output membership function you get, you apply a weighted average method of Defuzzification and get the final crisp output. As you have seen, there are different implication operators—the Mamdani or Sugeno approach. Here’s what the different operators are composed of:

- For AND operations in rulebase, you use T-Norm
- For OR operations in rulebase, you use T-Co-Norm
- For implication operations, you use T-Norm
- For aggregation operations, you use T-Co-Norm

Using the same example you saw in Mamdani FIS, you can see how the TSK method works (see Figure 3-24).

As you can see, the process remains the same, but the Defuzzification process changes. Therefore, the final result of this output will be given by the following formula:

$$Z = \frac{W1 Z1 + W2 Z2}{W1 + W2}$$

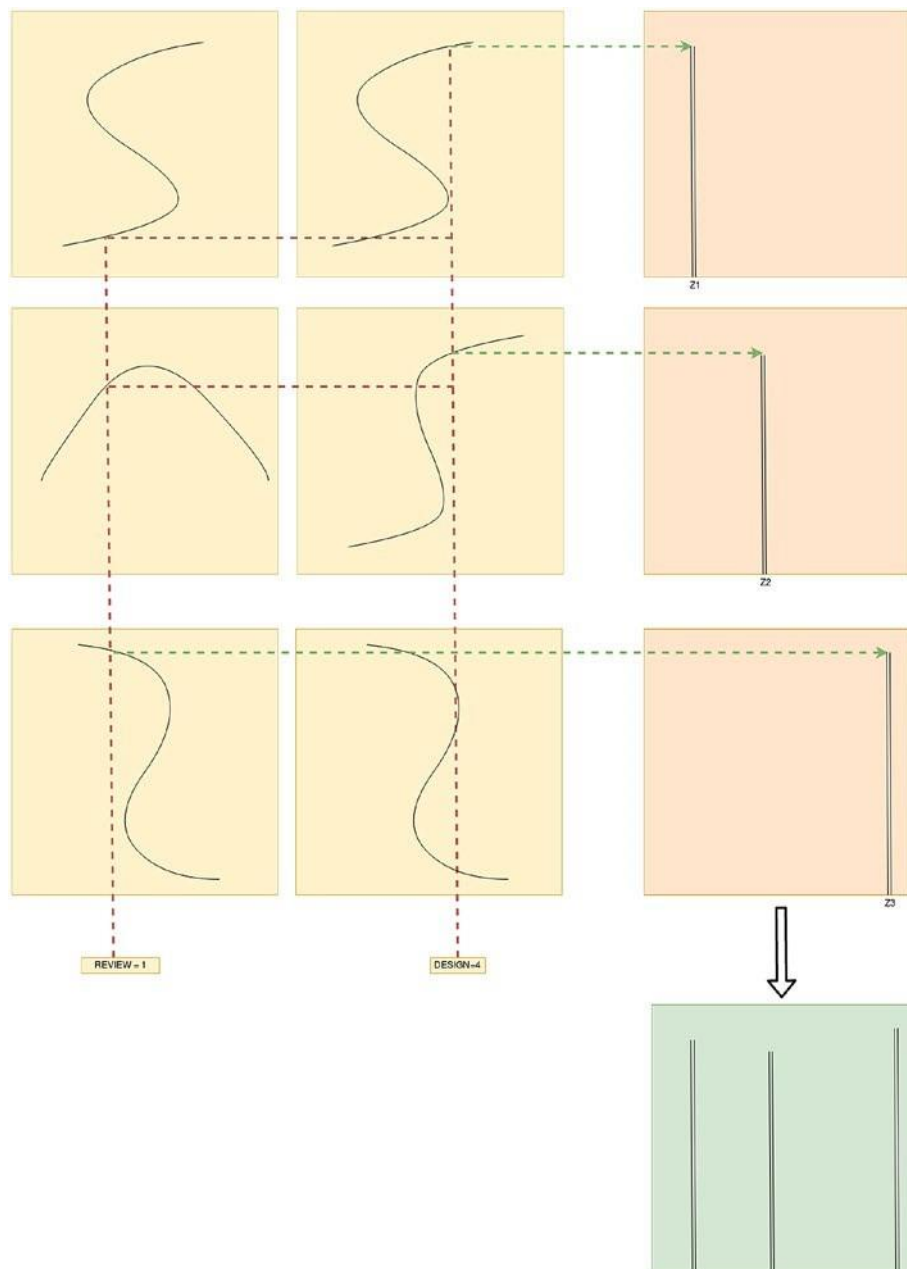


Figure 3.2.3. The TSK method

3.3 Machine Learning

Figure 4-1 illustrates the categories of Machine Learning. Classification Machine Learning (ML) models help you solve the problems when the output is categorical. A categorical output includes the variables that can be nominal or ordinal. For example, based on some specific traits, you ask your model to determine whether the stock price of a particular company is going to move up or down in the coming months. This is a classification problem.

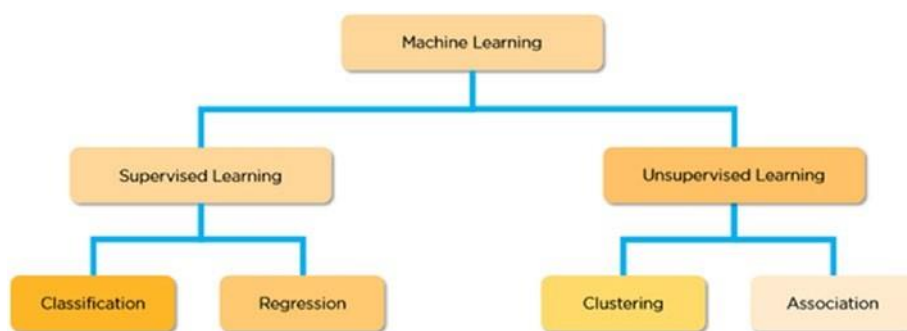


Figure 3.3.1. Machine Learning algorithms

Similarly, if you provide an image and want the ML model to classify whether the image is a person or an animal, this is again a classification problem. Conversely, instead of asking the ML model to classify the data, if you ask it to directly predict the output in the form of a numerical value, that's called a regression problem. For example, say you try to predict.

the price of a stock next month or the age of a person based on an image provided. This is a regression problem.

As discussed in the previous section, when an ML model tries to predict the category based on the data, that's a classification problem. An ML model can classify data into two classes or multiple classes. When you classify data into two classes, the ML problem is called a binary.

classification problem (see Figure 4-2). If there are more than two classes, it's called a multiclass classification problem.

Regression Machine Learning Problems

An ML model that tries to predict an actual numerical value based on the data is within the domain of a regression ML problem (see Figure 4-3).

Here are some examples of regression problems:

- Predicting a stock market price
- Predicting a company's revenue in the next quarter
- Finding the optimal speed of a self-driving car

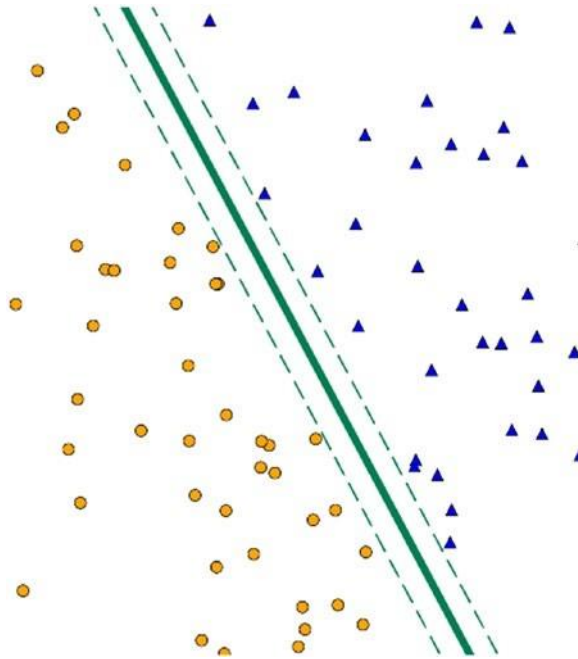


Figure 3.3.2. Binary classification

Classification Machine Learning Problems

The following are a few examples of binary and multiclass classification problems:

- Given an image, is it a cat or a dog—binary classification
- Whether a person will be able to pay their loan or not—binary classification
- Classifying an ECG signal to one of 13 health issues—multiclass classification
- A chatbot sending issues to different departments based on the question

asked—multiclass classification

- Classifying different types of driver distractions—multiclass classification

Regression Machine Learning Problems

An ML model that tries to predict an actual numerical value based on the data is within the domain of a regression ML problem (see Figure 4-3).

Here are some examples of regression problems:

- Predicting a stock market price
- Predicting a company's revenue in the next quarter
- Finding the optimal speed of a self-driving car

The Experience

Experience for an ML model is simply the data that you give to it, from which it learns the hidden patterns and then solves a problem. Data given to the model is specific to the type of problem that you expect the model to solve. When you talk about learning from the data in relation to classification and regression problems, different features are present in it along with the feature that you want to predict or classify. For example, if you want to predict the stock price or classify the stock price movement,

you may have data containing different features, such as information about high, low, open, and close to a specific stock. Past movement of the stock or past stock price information is also provided. So, the model looks at the entire dataset and then the actual value for that specific time. Based on that, it learns the relationship between all the variables. Once the learning process is done and you provide new data, the model looks at all the new features, understands the relationship, and then tries to predict or classify the new data.

This value that you are predicting or classifying is called the target variable or dependent variable (say, Y). All the other variables are called independent variables (say, x). You can say that the target variable is a function of all the independent variables, as given here. The function can be linear or nonlinear. Figure 3.3.3 shows gradient descent graph, which is one of the learning algorithms.

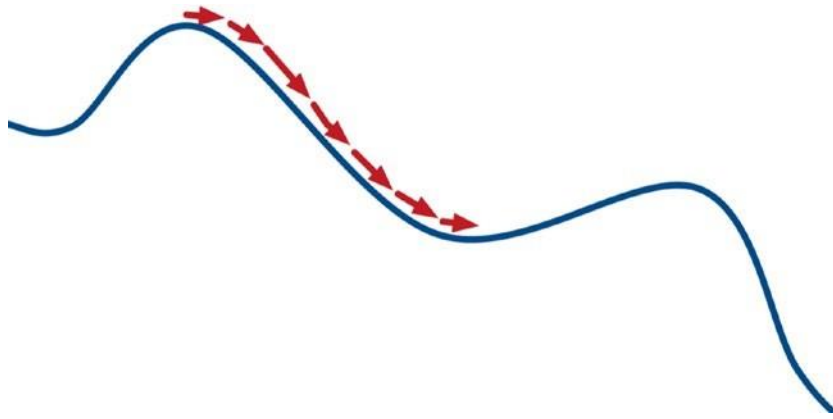


Figure 3.3.3. Learning graph (gradient descent)

In classification and regression problems, the data always contains a dependent variable, which is the variable that you want to predict or classify. There is one field of ML where the dependent variable is not given. These problems are the clustering or association types. Therefore, based on the data available, Machine Learning approaches fall into two areas:

- Supervised learning
- Unsupervised learning

3.4 Supervised and Unsupervised Learning

If you are crossing a road with the support of someone, this can be called a supervised approach. But, if you start crossing the road without anyone's help, it's an unsupervised approach. Taking a hint from this example, you can say that a supervised learning approach includes data that contains dependent variables. That's when the model looks at the input and output variables and tries to learn the relationship between them.

With unsupervised learning, the data doesn't contain the target variable (Y, as specified in supervised learning). It must look at the dataset and then find the similarities/differences/patterns between them. Based on that, you can have different unsupervised learning approaches. You have already seen the supervised learning approaches, which are classification and regression. Unsupervised

learning approaches include clustering, decomposition, and association. Figure 3.4.1 represents both problem statements visually.



Figure 3.4.1. Supervised versus unsupervised learning.

The following algorithms use supervised learning approaches:

- Linear/logistic regression
- K-nearest neighbor
- Naive Bayes theorem
- Decision trees
- Ensemble trees
- Support vector machines
- Neural networks

The following algorithms use unsupervised learning approaches:

- Hierarchical clustering
- K-means clustering
- Apriori rule
- Neural networks

Once you have decided what kind of problem your Machine Learning model needs to solve and select one of the algorithms from these lists, you come to the next phase, which is finding the performance of the model.

The Measure of Performance

Once your model is ready, it's time to ensure that you get accurate results. For that, you measure the performance of the model, based on the problem that it is trying to solve—classification or regression.

For Classification problems, the following are some of the measures you use:

- Accuracy
- Precision
- Recall or sensitivity
- F1-score
- Confusion matrix

For regression problems, here are some of the metrics:

- Root mean squared error
- AIC
- R2-score

The next sections discuss these measurements and metrics using the Titanic and House Price datasets.

Overfitting versus Underfitting

The next chapter looks at some of these Machine Learning approaches, but before that, you must be able to determine if your model is giving you poor performance. This can either be due to overfitting or underfitting of the data.

The data that you get to make the model is only a sample of all the data present in the universe. You can say that the data is incomplete and noisy. That's why, when you train the model, it tries to learn how well it is generalizing to new data. In other words, you can say that whatever the model has learned, whether it is able to apply these concepts to new data successfully is referred to as generalization.

If the model is applied to the data a little too well, it is called an overfitting problem. Sometimes there are too many details present in the data and a lot of unnecessary information is also present. If the model learns from this highly specific data, especially the details and the extra noise, it could lead to overfitting. This negatively impacts the performance. In underfitting, the model is not able to learn from the data, so it can't perform well on the new unseen data.

Underfitting and overfitting happen when you have imbalanced datasets. Suppose

for a binary classification problem, you have 90% of the data from one category and the remaining from the other. In this case, the model will learn most of the things related to the first category and very little related to the second category. This can also happen if you train the model a little too vigorously or add parameters to the model.

Overfitting happens when you train the model a little too long, while underfitting happens when the model is not trained long enough. In other words, you can say that if you train the model to such an extent where error starts to increase, overfitting may happen. But, if you stop training the model when the error is high and can still be reduced, underfitting may happen.

The best approach to solve this issue is to find the perfect spot between overfitting and underfitting. Figure 3.4.2 shows this best method. This concept is called a bias-variance tradeoff, and it's explained in the next section.

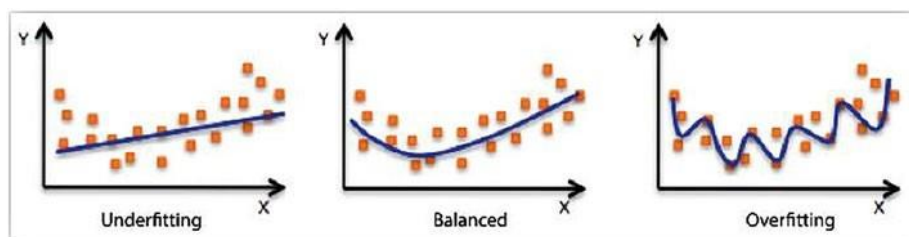


Figure 3.4.2. Overfitting vs. underfitting

Bias and Variance

When a model is trained by giving very little attention to the training data, then it becomes a biased model. In this scenario, the difference between the prediction and the original value becomes quite high. In other words, the error is large.

When overfitting happens in the model, that is when a lot of attention is given to the data, including the noise and the details. The variance of the model becomes high. Therefore, in this scenario, the model performs really well in the training data but not on the unseen data.

To solve these problems, you use the concept of a Bias-Variance Tradeoff. In this scenario, you try to find a middle approach where neither the bias nor the variance is high. In other words, you try to avoid both overfitting and underfitting in the data.

Figure 3.4.3 shows how bias and variance are related to the concepts of overfitting and underfitting.

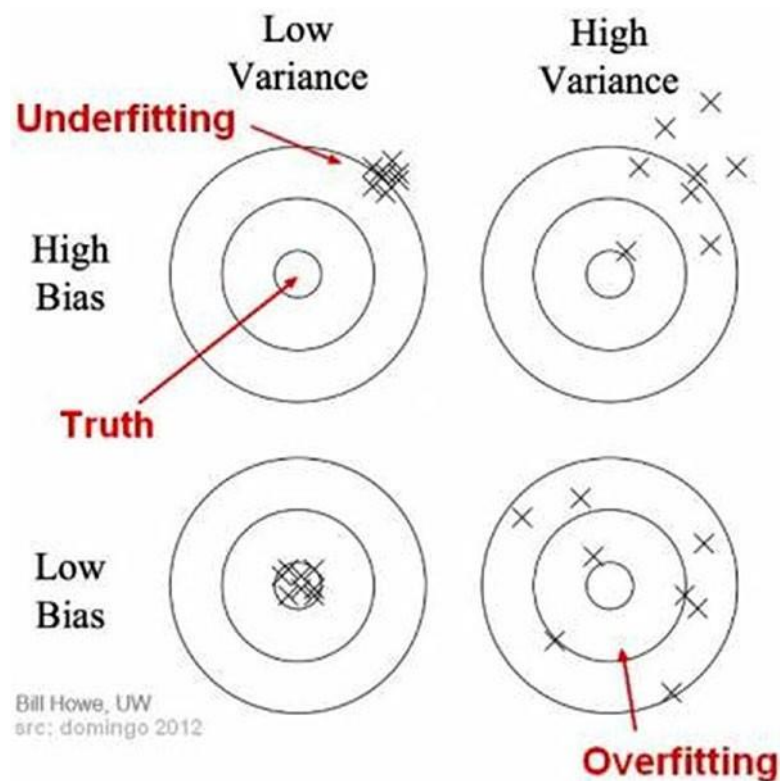


Figure 3.4.3. Bias and variance

Here are some approaches that can be used to solve the problem of overfitting.

- Adding cross-validations
- Training with more data
- Removing features
- Early stopping of training
- Adding regularizations
- Using the concept of ensembling

CHAPTER 4

SOFTWARE DEVELOPMENT METHODOLOGY

4.1 Artificial Neural Networks

The previous chapter discussed various terminology related to Machine Learning and some of the metrics used to check the accuracy of the model. This chapter discusses the concepts of neural networks. This chapter begins by explaining artificial neural networks and their components. It covers some of these components in detail, like activation functions, layers, etc. It then covers some of the advanced architectures of neural networks, like convolutional neural networks, recurrent neural networks, long short-term memory, and gated recurrent units. It also shows some applications of these concepts in Python.

4.1.1 Artificial Neural Networks Primer

Artificial neural networks (ANNs) are inspired by the functioning of biological neurons. Core Machine Learning algorithms use statistical concepts to learn different patterns present in the data. ANNs try to mimic human brain and neurons as much as possible to learn patterns. By using the mathematical techniques of linear algebra and calculus, ANNs learn from the data and try to find patterns.

4.1.2 An ANN consists of the following layers:

- \ Input layer
- \ Hidden layer
- \ Output layer

The input layer contains the examples with which you want to train the network. All the data you have, with which you want to make the system learn, is provided to the input layer. Training the network means that the machine tries to find all the possible

patterns present in the data and then learn it. The benefit of training is that when you give a new set of data, the machine tries to apply the learned patterns to the new set. If the patterns match, a decision is made based on what was done to the training data following this pattern. Hidden layers try to look at different combinations of the input layer and decide which of them is important, and how much importance should be given to them. They do this with the help of weights. Therefore, you can say that the hidden layers take weighted input. Once all the processing is done, the output layer computes all the outputs of the program and provides the results. Figure 5-1 shows the basic representation of all three layers. Neural networks operate by applying the concepts of forward and backpropagation. Therefore, it is imperative that you understand these concepts before moving on to ANN architectures in detail. The graph in Figure 5-1 is also called a computational graph. Each node in the graph is represented by a circle and represents a variable. This variable can be a scalar, vector, or tensor. Sometimes it can also be another variable. Each node is computed by applying some operation on the previous node. Therefore, in Figure 5-1, the hidden nodes are computed from input nodes and the output nodes are computed from the hidden nodes. This process where the output node is computed by the operations of previous nodes and the information provided from the input nodes are passed right to the output node is called forward propagation.

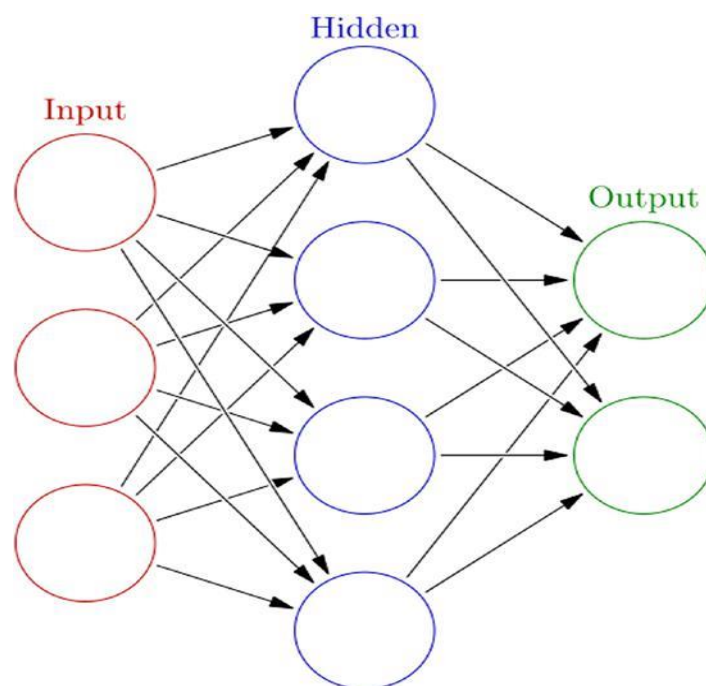


Figure 4.1.1 Computational graph of ANNs

Once the output nodes are created and their values are computed using the forward propagation approach, there is a need to compute the gradient, which requires the information to flow backward from the output node to input node. This concept is called backpropagation. In neural networks, it becomes very important to calculate gradients using the backpropagation approach because it helps minimize the cost function, which leads to much better and more accurate predictions.

Mathematically, the process of backpropagation can be represented by:

$$\frac{d}{dx} f(x, y)$$

You can calculate the gradient of a function $f()$, where x is the set of variables whose derivatives are required and y are the variables whose derivatives are not required (for example, input nodes). In learning algorithms like neural networks, this output function is called a cost function, and it's represented by:

$$J(q)$$

For a binary classification problem, the loss function $J(\theta)$ can be defined as follows:

$$J(q) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(p(y_i)) + (1 - y_i) \log(p(1 - y_i))]$$

So this equation is differentiated with respect to the input nodes, until the time when the loss value is minimum. This minimum value is known by reaching a minimum point, called the global minima, in the gradient descent process. Figure 5-2 shows the gradient descent process.

Every time you differentiate the cost function to find the gradients, you move down the parabolic curve shown in Figure 5-2. The aim of backpropagation is to reach a point at the bottom of the curve where the value of the loss function is at a minimum. As the value becomes minimum, you find the values of variables that gave that

value. In the case of neural networks, these variables are called weights and bias. You use these variables to predict the next set of values in the test set.

Keeping these concepts in mind, it's time to dive deeper into ANNs.

ANNs can be classified into two types:

- \ Perceptrons
- \ Multi-layer ANNs

The next sections explain these two ANNs and then the chapter moves forward to explain some of the complex neural network architectures used in the domain of computer vision and natural language processing.

Perceptrons

A perceptron is a single-layered neural network, which means it only has the input layer. Using different parameters, you can get the output. Its architecture may look like Figure 4.1.2

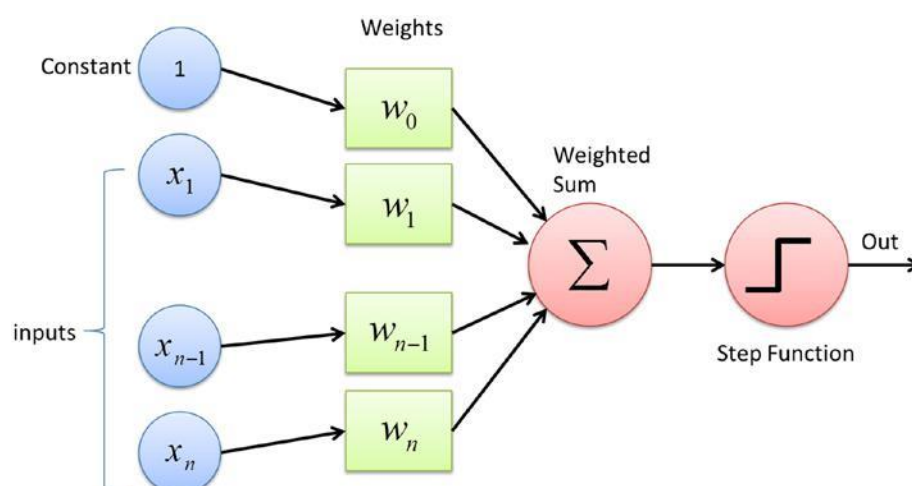


Figure 4.1.2. Typical perceptron architecture

Perceptron is mainly used as a linear (binary) classifier. A binary classifier classifies the input into only two categories. A perceptron consists of the following parts, as depicted in Figure 5-3:

- \ Input layer
- \ Weights and bias parameters
- \ Summation of weighted inputs
- \ Applying activation/step function on the weighted sum to get the output

So, in a perceptron, all the inputs are multiplied with the learned weights (w). These weights are learned through the process of backpropagation. The summation of the weighted input takes place and the output is passed to the activation function, which provides the final output. There are different types of activation functions, which give different outputs based on the formula. To get a binary classification, you use the sigmoid or ReLU activation functions. Before you move further to multi-layered perceptrons, take a look at activation functions.

Each neuron in the hidden layer or output layer has its own activation function. This helps to decide whether the output from a particular neuron is important or not. The weights that the system has learned are multiplied with the input neuron value. Then, a bias value is added. The output value is based on the activation function, which determines the importance. When you have an activation function in the output neuron, it takes the output of all the previous neurons, where activation function is applied, and gives a final answer by performing the weighted sum. There are different types of activation functions, some of which are:

- \ Sigmoid
- \ Tanh
- \ Softmax

- \ ReLU

- \ Leaky ReLU

4.1.3 Sigmoid Activation Function

A sigmoid activation function has an s-shaped curve. Its range lies from 0 to 1. Since its upper and lower limit is 0 and 1, it is most widely used with binary classification problems. Here is the formula and the curve of the sigmoid activation function (see Figure 5-4), followed by the Python implementation.

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

```
import numpy as np
```

```
def sigmoid(x):
```

```
    return 1 / (1 + np.exp(-x))
```

Once the system has learned the weights and the bias of each neuron, and you give ReLU as an activation function, it usually gives the same output as it received as the input. But since any values that are less than 0 are converted to 0, the neurons may not be able to differentiate between the inputs. This problem is called the dying ReLU problem.

This makes a neuron practically dead. Even the slope of negative values is zero. If you keep moving forward and don't heed this problem, eventually a large part of the neural network will end up doing nothing. This problem generally happens when the learning rate is too high. By making the learning rate smaller or changing the activation function, for example to leaky- ReLU, you can solve the problem.

4.1.4 Multi-Layer ANNs

There are different types of multi-layer artificial neural networks. This chapter discusses the most relevant ones. The general architecture of a multi-layer ANN is shown in Figure 4.1.3.

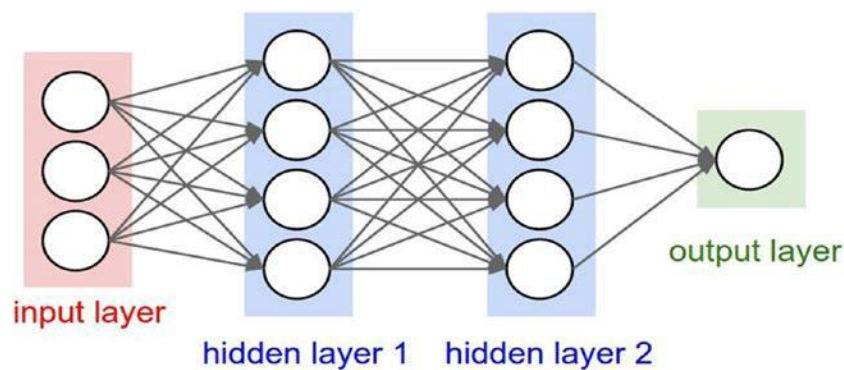


Figure 4.1.3. Typical multi-layered ANN architecture

The following sections review these ANNs:

- \ Convolutional neural networks
- \ Recurrent neural networks
- \ Long short-term memory
- \ Gated recurrent units

Convolutional neural networks are used in areas like image recognition, image classification, object detection, face recognition, etc. Before looking at the architecture and the process involved in a CNN, you should first understand how a computer looks and understands an image. Computers break an image into a matrix of pixels and store the color code for each pixel.

4.1.5 The Convolutional Layer

The convolution layer is the main building block of a convolutional neural network. The convolution layer is used to understand the patterns present in the image and extract the interesting features from them. The total number of convolutional layers

defines the total number of features that you want to extract from the image. For example, five convolutional layers means that five features are learned from an image. These features can be decipherable, like finding the edges or the threshold image, etc. or it they may be too complex for a human to understand. Hence, this is the main layer responsible for learning features, such as what unique features an image containing a human has and how they are different from the features present in an image of an animal. The values of all the filters are learnable, which means you must provide the matrix dimensions to a CNN, and it will automatically learn the best values for the convolution. This matrix of convolution is also called a kernel or filter. Mathematically, each kernel operation happens in a way represented in Figure 4.1.4

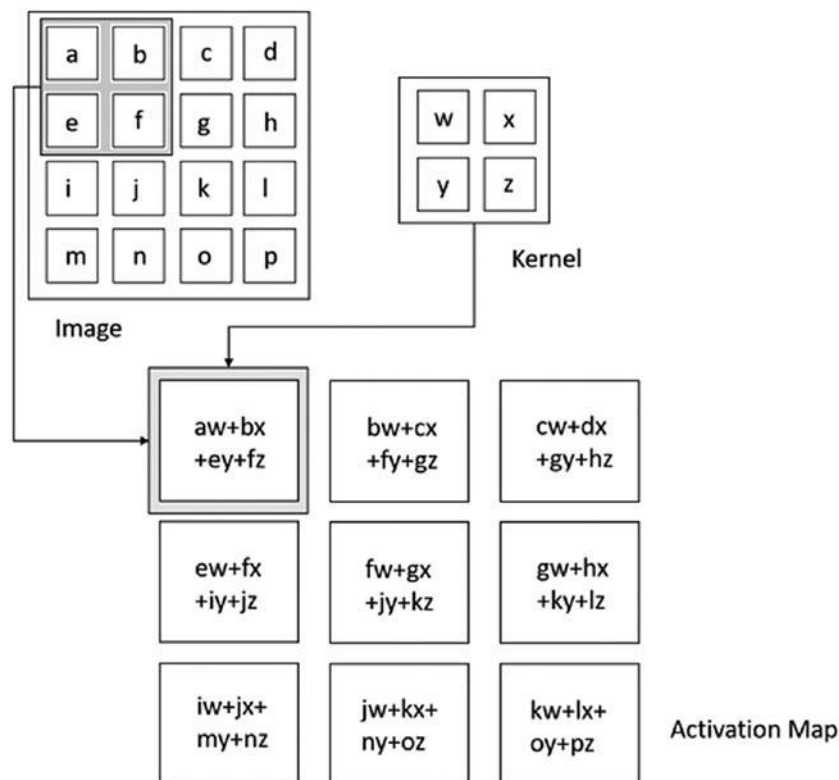


Figure 4.1.4. Process of convolving through filters

Based on the size of the kernel matrix, a matrix dot product happens at a specific portion of the image. This operation is continued until the kernel covers the entire image. This results in a new image with a smaller number of dimensions, as compared to the original image, but the depth of image is higher. The output is called

an activation map and the process is called strides.

Padding

The filter might fit inside the image perfectly. But if it does not, you use the concept of padding. In this process, you add a few extra values in the input image so that the filter fits in well. The most commonly used value is 0 (called zero-padding). Alternatively, you can drop the part of the image where the filter doesn't fit.

Activation Function

The actual decision of classification is taken in this layer. The most used activation function in CNN is ReLU, the rectified linear unit. This helps the neurons give exact pixel values as output for all the positive values, but for all the negative values the output is always zero. This results in sparse matrix generation and hence means lesser computation time with better learning. Generally, pixel values are positive, so the problem of dying ReLU doesn't apply.

Pooling Layer

If an image is taken from a DSLR camera, the resolution will be very high. Since the resolution is high, the number of pixels will be high. Even though you use filters with a smaller dimension than the input image, the computational time will still be long. Therefore, to overcome this issue, you can use pooling. It is used to reduce the size of the input image, as well as the outcomes of convolutions. Because of this, the number of parameters that need to be analyzed is smaller. Hence, the computational time decreases. This layer operates on each feature map (the outcome of individual convolutions) independently.

4.2 Advanced Fuzzy Networks

Figure 4.2.1 shows a classic Neuro Fuzzy System. But before you get too far, it's important to know some of the core components that are used in building these systems. This chapter starts by discussing the Fuzzy Clustering method. Then it moves on to genetic algorithms and wraps up by reviewing the most commonly used architectures belonging to the domain of advanced Fuzzy Networks.

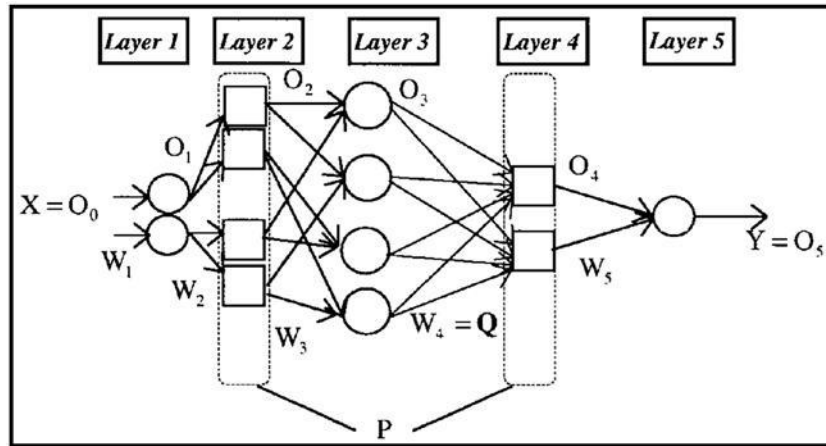


Figure 4.2.1. Neuro-Fuzzy Inference System

This chapter begins by discussing Fuzzy Clustering—their requirements and their applications.

4.2.1 Fuzzy Clustering

Clustering is a method you use to group data into a few categories, based on the similarities present. These categories are termed clusters (see Figure 7-2).

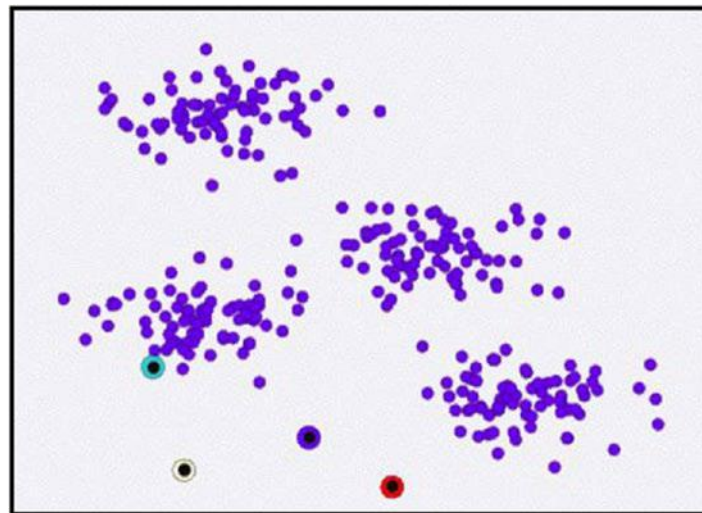


Figure 4.2.2. Clustering

Machine Learning uses different methods for clustering, such as K-Means Clustering, Hierarchical Clustering, DBScan, and so on. Similarly, when we talk about Fuzzy Networks, there are methods like Fuzzy C-Means Clustering, Gaussian Application to find clusters, and so on. This chapter discusses Fuzzy C-Means

Clustering in detail.

Fuzzy C-Means Clustering

Suppose I have a set of data points, X , which I want to put into a k number of clusters based on some similarity metrics. In the form of a set, suppose the data looks like the following:

$$X = \{ x_1, x_2, x_3, \dots, x_m \}$$

To divide them into k number of clusters using the C-Means Clustering approach, the first thing that you do is randomly take k number of points from the data. Assume that these k points are the centroids

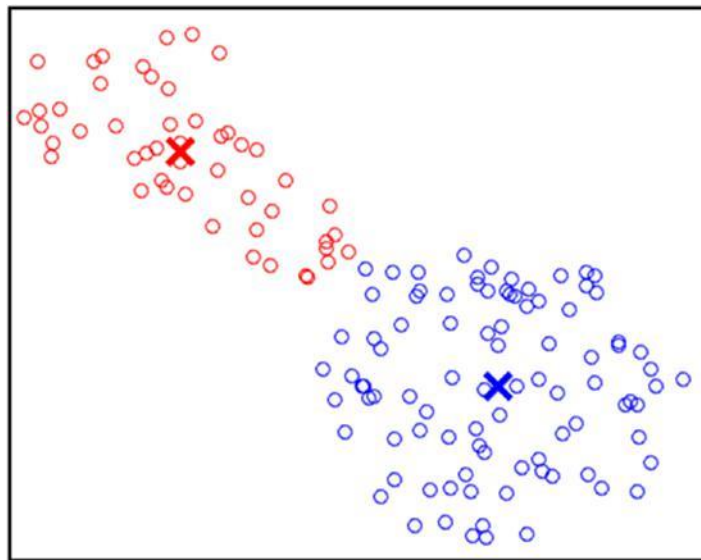


Figure 4.2.3 C-Means Clustering

Using the Manhattan or Euclidean distance, you then find out that all the remaining points are closer to the cluster centroid point. This is how you get the first set of data assigned to k number of clusters. But you started by randomly selecting k points and calling them centroids. In reality, they are not. Therefore, you need to continue the process to rectify this assumption.

Euclidean Distance

A normal straight-line distance between any two points is called a Euclidean Distance. To get this distance, you use the following formula:

$$d = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

In the next step, you find the actual centroids of the clusters that you defined. Once you find that, some of the points may lose the original membership of that cluster. They may now be much closer to another cluster, as compared to the cluster that they were originally part of. Hence, you start reassigning these points based on the new centroids. This will help you get a revised k number of clusters with new member points. Again, you find the new centroid and repeat the process. You keep on repeating the process until the new clusters don't make the points move from one cluster to another.

Let's look at the entire algorithm mathematically:

1. Define N data points that need to be clustered:

$$x_i, \text{ where } i = 1, 2, 3, \dots, N$$

2. Assume the number of cluster to be made is represented by C, where $2 \leq C \leq N$
3. Define the cluster fuzziness represented by f, where $f > 1$
4. Define a membership matrix U having dimension of $N \times C \times M$. This matrix should be defined randomly following these conditions:
 - a. $U_{ijm} \in [0,1]$, and $\sum_{j=1}^C U_{ijm} = 1$
 - b. $\sum_{m=1}^M U_{ijm} = 0$ for each i and fixed value of m.i=1

5. Determine the cluster centers. This can be done by using the following equation:

$$C_j = \frac{\sum_{i=1}^N U_{ijm}^f x_{im}}{\sum_{i=1}^N U_{ijm}^f}$$

where j represents the cluster and m represents the dimension.

6. Calculate the Euclidean Distance. You can find this equation by using the

following equation:

$$D_{ijm} = (x_{im} - C_{jm})$$

where i represents data point, j represents cluster, and m represents dimension

\ 7.\ After finding the Euclidean Distance, you have to update the membership matrix defined in Step 4 with the new value. This can be done by using the following equation:

$$U_{ijm} = 1$$

$$\frac{1}{D_{ijm}^2} \propto D_{ijm}^{-1}$$

$$c = 1 \div D_{ijm}$$

$$\frac{1}{D_{ijm}^2}$$

We apply the above equation only for the data points where $D_{ijm} > 0$. If $D_{ijm} = 0$ then we have full membership and the value initialized is 1.0

7. Repeat the Steps 1 to 5 until the value of $U < \epsilon$, where ϵ is the termination criteria.

CHAPTER 5

RESULTS AND DISCUSSION

Our Project diversifies the usage and implementation of the soft computing methodologies, wherein the results are fairly decent with 90 percent accurate as shown in the complete documentation, each and every methodology are driven by code quality and seriously building up a good software for the complete ecosystem. The complications during the entire development process helped us to undergo complete research and make it more accurate for building the intersection of powerful technologies, Fuzzy logic, and Neural Networks.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

In this project, the notion of implementing Adaptive Neuro-Fuzzy Inference System as a reliable Python programming language package, we have reached almost excellent outcomes over multiple challenges and open for future enchantments with a constructive code base, the coming up feature can always be appended in the future, this makes the complete developer and researcher eco-system much easier towards gaining ease in work.

The complications during the entire development process helped us to undergo complete research and make it more accurate for building the intersection of powerful technologies, Fuzzy logic, and Neural Networks.

6.2 FUTURE ENHANCEMENT

The relational algebra uses set union, set difference, and Cartesian product from set theory, but adds additional constraints to these operators. For set union and set difference, the two relations involved must be union-compatible—that is, the two relations must have the same set of attributes.

REFERENCES

- 1 Jang, Jyh-Shing R (1991). Fuzzy Modeling Using Generalized Neural Networks and Kalman Filter Algorithm (PDF). Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14–19. 2. pp. 762–767.

- 2 Jang, J.-S.R. (1993). "ANFIS: adaptive-network-based fuzzy inference system". IEEE Transactions on Systems, Man and Cybernetics. 23 (3): 665–685. doi:10.1109/21.256541. S2CID 14345934.

- 3 Abraham, A. (2005), "Adaptation of Fuzzy Inference System Using Neural Learning", in Nedjah, Nadia; de Macedo Mourelle, Luiza (eds.), Fuzzy Systems Engineering: Theory and Practice, Studies in Fuzziness and Soft Computing, 181, Germany: Springer Verlag, pp. 53–83, CiteSeerX 10.1.1.161.6135, doi:10.1007/11339366_3, ISBN 978-3-540-25322-8

- 4 Jang, Sun, Mizutani (1997) – Neuro-Fuzzy and Soft Computing – Prentice Hall, pp 335–368, ISBN 0-13-261066-3

- 5 Tahmasebi, P. (2012). "A hybrid neural networks-fuzzy logic-genetic algorithm for grade estimation". Computers & Geosciences. 42: 18–27. Bibcode:2012CG.....42...18T. doi:10.1016/j.cageo.2012.02.004. PMC 4268588. PMID 25540468.

- 6 Tahmasebi, P. (2010). "Comparison of optimized neural network with fuzzy logic for ore grade estimation". Australian Journal of Basic and Applied Sciences. 4: 764–772.

- 7 Kamal, Mohasinina Binte; Mendis, Gihan J.; Wei, Jin (2018). "Intelligent Soft Computing-Based Security Control for Energy Management Architecture of Hybrid Emergency Power System for More-Electric Aircrafts [sic]". IEEE Journal of Selected Topics in Signal Processing. 12 (4): 806. Bibcode:2018ISTSP..12..806K. doi:10.1109/JSTSP.2018.2848624. S2CID

51908378.

- 8 Karaboga, Dervis; Kaya, Ebubekir (2018). "Adaptive network based fuzzy inference system (ANFIS) training approaches: a comprehensive survey". *Artificial Intelligence Review*. 52 (4): 2263–2293. doi:10.1007/s10462-017-9610-2. ISSN 0269-2821. S2CID 40548050.
- 9 J.-S.R. Jang (1992). "Self-learning fuzzy controllers based on temporal backpropagation". *IEEE Transactions on Neural Networks*. Institute of Electrical and Electronics Engineers (IEEE). 3 (5): 714–723. doi:10.1109/72.159060. PMID 18276470.
- 10 Anish Pandey and Saroj Kumar and Krishna Kant Pandey and Dayal R. Parhi (2016). "Mobile robot navigation in unknown static environments using ANFIS controller". *Perspectives in Science*. Elsevier BV. 8: 421–423. doi:10.1016/j.pisc.2016.04.094.

APPENDIX

SOURCE CODE

```
import itertools
import numpy as np
from anfis.membership import mfDerivs
import copy

class ANFIS:
    """Class to implement an Adaptive Network Fuzzy Inference System: ANFIS"
    Attributes:
        X
        Y
        XLen
        memClass
        memFuncs
        memFuncsByVariable
        rules
        consequents
        errors
        memFuncsHomo
        trainingType
    """

    def __init__(self, X, Y, memFunction):
        self.X = np.array(copy.copy(X))
        self.Y = np.array(copy.copy(Y))
        self.XLen = len(self.X)
        self.memClass = copy.deepcopy(memFunction)
        self.memFuncs = self.memClass.MFList
        self.memFuncsByVariable = [[x for x in range(len(self.memFuncs[z]))] for z in
range(len(self.memFuncs))]
```

```

self.rules = np.array(list(itertools.product(*self.memFuncsByVariable)))
self.consequents = np.empty(self.Y.ndim * len(self.rules) * (self.X.shape[1] +
1))
self.consequents.fill(0)
self.errors = np.empty(0)
self.memFuncsHomo = all(len(i)==len(self.memFuncsByVariable[0]) for i in
self.memFuncsByVariable)
self.trainingType = 'Not trained yet'

```

```

def LSE(self, A, B, initialGamma = 1000.):
    coeffMat = A
    rhsMat = B
    S = np.eye(coeffMat.shape[1])*initialGamma
    x = np.zeros((coeffMat.shape[1],1)) # need to correct for multi-dim B
    for i in range(len(coeffMat[:,0])):
        a = coeffMat[i,:]
        b = np.array(rhsMat[i])
        S = S -
(np.array(np.dot(np.dot(np.dot(S,np.matrix(a).transpose()),np.matrix(a)),S)))/(1+(n
p.dot(np.dot(S,a),a)))
        x = x + (np.dot(S,np.dot(np.matrix(a).transpose()),(np.matrix(b)-
np.dot(np.matrix(a),x))))
    return x

```

```

def trainHybridJangOffLine(self, epochs=5, tolerance=1e-5, initialGamma=1000,
k=0.01):

```

```

    self.trainingType = 'trainHybridJangOffLine'
    convergence = False
    epoch = 1

```

```

    while (epoch < epochs) and (convergence is not True):

```

```

        #layer four: forward pass

```

```

[layerFour, wSum, w] = forwardHalfPass(self, self.X)

#layer five: least squares estimate
layerFive = np.array(self.LSE(layerFour,self.Y,initialGamma))
self.consequents = layerFive
layerFive = np.dot(layerFour,layerFive)

#error
error = np.sum((self.Y-layerFive.T)**2)
print('current error: '+ str(error))
average_error = np.average(np.absolute(self.Y-layerFive.T))
self.errors = np.append(self.errors,error)

if len(self.errors) != 0:
    if self.errors[len(self.errors)-1] < tolerance:
        convergence = True

# back propagation
if convergence is not True:
    cols = range(len(self.X[0,:]))
    dE_dAlpha = list(backprop(self, colX, cols, wSum, w, layerFive) for colX
in range(self.X.shape[1]))

if len(self.errors) >= 4:
    if (self.errors[-4] > self.errors[-3] > self.errors[-2] > self.errors[-1]):
        k = k * 1.1

if len(self.errors) >= 5:
    if (self.errors[-1] < self.errors[-2]) and (self.errors[-3] < self.errors[-2]) and
(self.errors[-3] < self.errors[-4]) and (self.errors[-5] > self.errors[-4]):
        k = k * 0.9

## handling of variables with a different number of MFs

```

```

t = []
for x in range(len(dE_dAlpha)):
    for y in range(len(dE_dAlpha[x])):
        for z in range(len(dE_dAlpha[x][y])):
            t.append(dE_dAlpha[x][y][z])

eta = k / np.abs(np.sum(t))

if(np.isinf(eta)):
    eta = k

## handling of variables with a different number of MFs
dAlpha = copy.deepcopy(dE_dAlpha)
if not(self.memFuncsHomo):
    for x in range(len(dE_dAlpha)):
        for y in range(len(dE_dAlpha[x])):
            for z in range(len(dE_dAlpha[x][y])):
                dAlpha[x][y][z] = -eta * dE_dAlpha[x][y][z]
else:
    dAlpha = -eta * np.array(dE_dAlpha)

for varsWithMemFuncs in range(len(self.memFuncs)):
    for MFs in range(len(self.memFuncsByVariable[varsWithMemFuncs])):
        paramList = sorted(self.memFuncs[varsWithMemFuncs][MFs][1])
        for param in range(len(paramList)):
            self.memFuncs[varsWithMemFuncs][MFs][1][paramList[param]] =
self.memFuncs[varsWithMemFuncs][MFs][1][paramList[param]] +
dAlpha[varsWithMemFuncs][MFs][param]
    epoch = epoch + 1

self.fittedValues = predict(self,self.X)
self.residuals = self.Y - self.fittedValues[:,0]

```

```
return self.fittedValues
```

```
def plotErrors(self):  
    if self.trainingType == 'Not trained yet':  
        print(self.trainingType)  
    else:  
        import matplotlib.pyplot as plt  
        plt.plot(range(len(self.errors)),self.errors,'ro', label='errors')  
        plt.ylabel('error')  
        plt.xlabel('epoch')  
        plt.show()
```

```
def plotMF(self, x, inputVar):  
    import matplotlib.pyplot as plt  
    from skfuzzy import gaussmf, gbellmf, sigmf  
  
    for mf in range(len(self.memFuncs[inputVar])):  
        if self.memFuncs[inputVar][mf][0] == 'gaussmf':  
            y = gaussmf(x,**self.memClass.MFList[inputVar][mf][1])  
        elif self.memFuncs[inputVar][mf][0] == 'gbellmf':  
            y = gbellmf(x,**self.memClass.MFList[inputVar][mf][1])  
        elif self.memFuncs[inputVar][mf][0] == 'sigmf':  
            y = sigmf(x,**self.memClass.MFList[inputVar][mf][1])  
  
        plt.plot(x,y,'r')  
  
    plt.show()
```

```
def plotResults(self):  
    if self.trainingType == 'Not trained yet':  
        print(self.trainingType)  
    else:
```

```

import matplotlib.pyplot as plt
plt.plot(range(len(self.fittedValues)),self.fittedValues,'r', label='trained')
plt.plot(range(len(self.Y)),self.Y,'b', label='original')
plt.legend(loc='upper left')
plt.show()

def forwardHalfPass(ANFISObj, Xs):
    layerFour = np.empty(0,)
    wSum = []

    for pattern in range(len(Xs[:,0])):
        #layer one
        layerOne = ANFISObj.memClass.evaluateMF(Xs[pattern,:])

        #layer two
        miAlloc = [[layerOne[x][ANFISObj.rules[row][x]] for x in
range(len(ANFISObj.rules[0]))] for row in range(len(ANFISObj.rules))]
        layerTwo = np.array([np.product(x) for x in miAlloc]).T
        if pattern == 0:
            w = layerTwo
        else:
            w = np.vstack((w,layerTwo))

        #layer three
        wSum.append(np.sum(layerTwo))
        if pattern == 0:
            wNormalized = layerTwo/wSum[pattern]
        else:
            wNormalized = np.vstack((wNormalized,layerTwo/wSum[pattern]))

        #prep for layer four (bit of a hack)
        layerThree = layerTwo/wSum[pattern]

```

```

        rowHolder = np.concatenate([x*np.append(Xs[pattern,:],1) for x in
layerThree])
        layerFour = np.append(layerFour,rowHolder)

w = w.T
wNormalized = wNormalized.T

layerFour = np.array(np.array_split(layerFour,pattern + 1))

return layerFour, wSum, w

def backprop(ANFISObj, columnX, columns, theWSum, theW, theLayerFive):

    paramGrp = [0]* len(ANFISObj.memFuncs[columnX])
    for MF in range(len(ANFISObj.memFuncs[columnX])):

        parameters = np.empty(len(ANFISObj.memFuncs[columnX][MF][1]))
        timesThru = 0
        for alpha in sorted(ANFISObj.memFuncs[columnX][MF][1].keys()):

            bucket3 = np.empty(len(ANFISObj.X))
            for rowX in range(len(ANFISObj.X)):
                varToTest = ANFISObj.X[rowX,columnX]
                tmpRow = np.empty(len(ANFISObj.memFuncs))
                tmpRow.fill(varToTest)

                bucket2 = np.empty(ANFISObj.Y.ndim)
                for colY in range(ANFISObj.Y.ndim):

                    rulesWithAlpha =
np.array(np.where(ANFISObj.rules[:,columnX]==MF))[0]
                    adjCols = np.delete(columns,columnX)

```

```

senSit =
mfDerivs.partial_dMF(ANFISObj.X[rowX,columnX],ANFISObj.memFuncs[columnX
][MF],alpha)
# produces d_ruleOutput/d_parameterWithinMF
dW_dAlpha = senSit *
np.array([np.prod([ANFISObj.memClass.evaluateMF(tmpRow)[c][ANFISObj.rules[r
][c]] for c in adjCols]) for r in rulesWithAlpha])

bucket1 = np.empty(len(ANFISObj.rules[:,0]))
for consequent in range(len(ANFISObj.rules[:,0])):
    fConsequent =
np.dot(np.append(ANFISObj.X[rowX,:],1.),ANFISObj.consequents[((ANFISObj.X.s
hape[1] + 1) * consequent):(((ANFISObj.X.shape[1] + 1) * consequent) +
(ANFISObj.X.shape[1] + 1)),colY))
    acum = 0
    if consequent in rulesWithAlpha:
        acum = dW_dAlpha[np.where(rulesWithAlpha==consequent)] *
theWSum[rowX]

    acum = acum - theW[consequent,rowX] * np.sum(dW_dAlpha)
    acum = acum / theWSum[rowX]**2
    bucket1[consequent] = fConsequent * acum

sum1 = np.sum(bucket1)

if ANFISObj.Y.ndim == 1:
    bucket2[colY] = sum1 * (ANFISObj.Y[rowX]-
theLayerFive[rowX,colY])*(-2)
else:
    bucket2[colY] = sum1 * (ANFISObj.Y[rowX,colY]-
theLayerFive[rowX,colY])*(-2)

sum2 = np.sum(bucket2)
bucket3[rowX] = sum2

```

```

        sum3 = np.sum(bucket3)
        parameters[timesThru] = sum3
        timesThru = timesThru + 1

    paramGrp[MF] = parameters

    return paramGrp

def predict(ANFISObj, varsToTest):

    [layerFour, wSum, w] = forwardHalfPass(ANFISObj, varsToTest)

    #layer five
    layerFive = np.dot(layerFour,ANFISObj.consequents)

    return layerFive

if __name__ == "__main__":
    print("I am main!")

test code

from anfis.anfis import ANFIS
import membership.mfDerivs
import membership.membershipfunction
import numpy

ts = numpy.loadtxt("trainingSet.txt",
usecols=[1,2,3])#numpy.loadtxt('c:\\Python_fiddling\\myProject\\MF\\trainingSet.txt'
,usecols=[1,2,3])
X = ts[:,0:2]
Y = ts[:,2]

```

```
mf = [
    [
        ['gaussmf',{'mean':0.,'sigma':1.}],
        ['gaussmf',{'mean':-1.,'sigma':2.}],
        ['gaussmf',{'mean':-4.,'sigma':10.}],
        ['gaussmf',{'mean':-7.,'sigma':7.}]
    ],
    [
        ['gaussmf',{'mean':1.,'sigma':2.}],
        ['gaussmf',{'mean':2.,'sigma':3.}],
        ['gaussmf',{'mean':-2.,'sigma':10.}],
        ['gaussmf',{'mean':-10.5,'sigma':5.}]
    ]
]
```

```
mfc = membership.membershipfunction.MemFuncs(mf)
anf = ANFIS(X, Y, mfc)
anf.trainHybridJangOffLine(epochs=100)
print(round(anf.consequents[-1][0],6))
print(round(anf.consequents[-2][0],6))
print(round(anf.fittedValues[9][0],6))
if round(anf.consequents[-1][0],6) == -5.275538 and round(anf.consequents[-
2][0],6) == -1.990703 and round(anf.fittedValues[9][0],6) == 0.002249:
    print('test is good')
anf.plotErrors()
anf.plotResults()
anf.model_report()
```

SNAP SHOTS

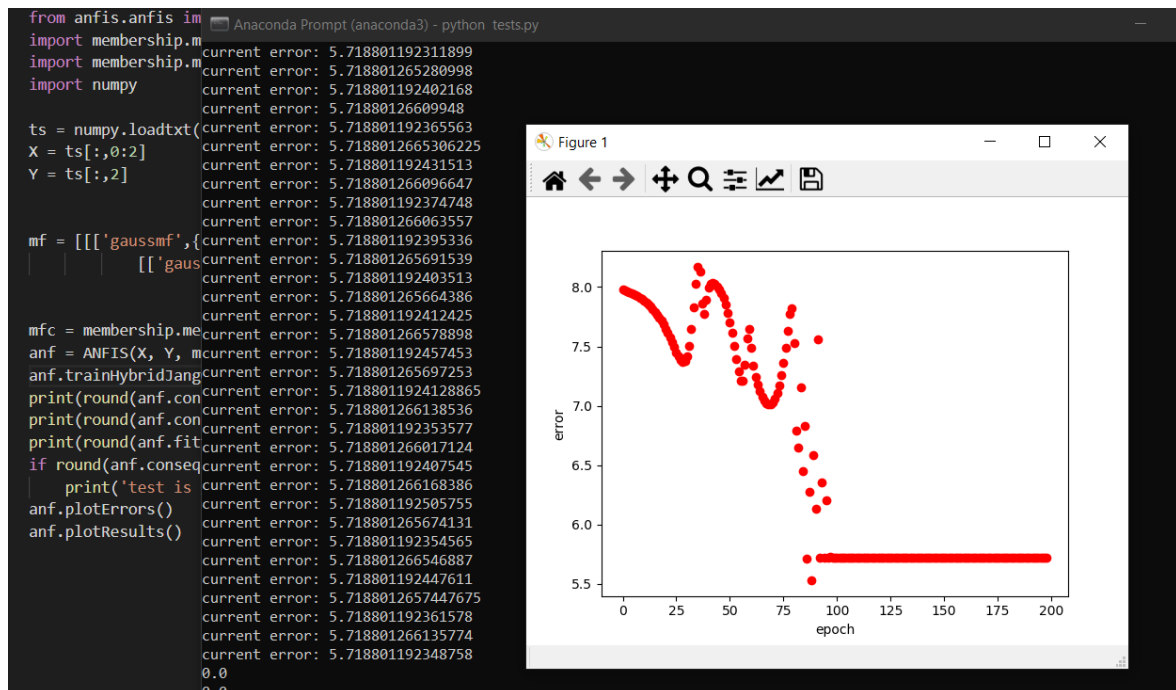


Fig 1: anfis_apoch vs error

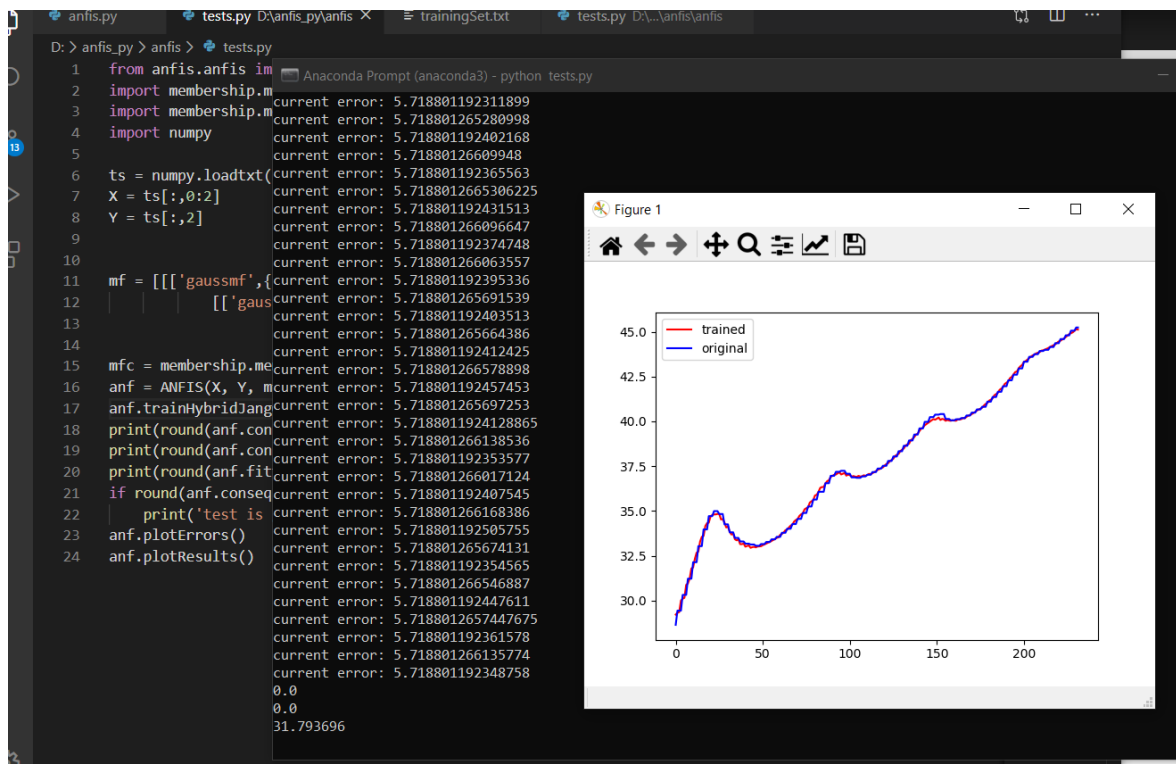


Fig 2: anfis_trained_vs_originated