"Modified OnLine Sequential Fuzzy Extreme Learning Machine"

A Dissertation Submitted towards the Partial Fulfillment of Award of Degree of

MASTER OF TECHNOLOGY
in
MICROWAVE AND OPTICAL COMMUNICATION ENGINEERING

Submitted by

Akhilesh Chandra Bhatnagar

Roll No: 2k09/MOC/03

Under the guidance of

Mr. Mahipal Singh Choudhary

Asst. Professor
Department of Electronics & Communication Engineering

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

IN ASSOCIATION WITH

DEPARTMENT OF APPLIED PHYSICS

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Main Bawana Road, Shahabad Daulatapur, New Delhi – 110042

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY,

NEW DELHI- 110042

CERTIFICATE

This is to certify that the minor project titled **"Modified Online Sequential Fuzzy Extreme Learning Machine"** is the bonafied work of Mr. Akhilesh Chandra Bhatnagar under our guidance of Mr. Mahipal Singh Choudhary (Asst Professor , Delhi Technological university, DTU, Delhi, INDIA) and supervision of Dr. Sundaram Suresh (Asst Professor, School of Computer Engineering, NTU, SINGAPORE) in fulfillment of requirement towards the degree of Master of Technology in Electronics and Communication with specialization in Microwave and Optical Communication Engineering from Delhi Technological University, Delhi, INDIA.

Mr. Mahipal Singh Choudhary
Assistant Professor & Supervisor
Electronics & Communication
Engineering ,
Delhi Technological University
Delhi, INDIA.
Date: /06/2011

Dr. Rajiv Kapoor
Head of Department
Electronics & Communication
Engineering,
Delhi Technological University,
Delhi, INDIA.
Date: /06/2011

ACKNOWLEDGEMENT

I express my sincere gratitude to my Dr. Rajiv kapoor, HOD ECE, Dr. R.K. Sinha, HOD Physics and Mr. Mahipal Singh Choudhary, A.P. ECE for his useful guidance, encouragement and contribution offered throughout all phases of this project. I also express my sincere gratitude to Dr Sundaram Suresh and his PhD. student kartik of NTU, for his useful guidance, encouragement and contribution offered throughout all phases of this project. Their persistent encouragement, everlasting experience and valuable inspiration helped me a lot in building a present shape of project.

I also express my gratitude to the Department of Electronics and Communication Engineering, Delhi Technological University for providing the access to Microwave Laboratory and the much needed technical material from Central Library. I should not fail to mention my parents who have always been a source of inspiration. I am grateful to my friends for their valuable support and help.

[illegible]

Akhilesh Chandra Bhatnagar
Roll No: 2k09/MOC/03
M.Tech. (Final Semester)
Microwave & Optical Communication Engg.
Delhi Technological University
Delhi, INDIA.

ABSTRACT

This report addresses modification for recently developed sequential learning algorithm (OS-Fuzzy ELM) and its performance evaluation is done using multicategory classification Data Sets of VC, GI and IS and Binary classification data sets like liver disorder from UCI. There are two main sections to the report. The first of these is the presentation of research gathered on fuzzy neural networks and the possible purpose they could serve in communications, as well as giving background information on the individual disciplines.
The second half of the report is concerned with Modified OS-Fuzzy ELM algorithm and its performance evaluation and comparison of results with recently developed sequential learning algorithm for Self-adaptive Re- source Allocation Network classifier (SRAN).

Keywords:- Resource allocation network, self-adaptive control parameters, sequential learning, multi-category classification, Online Sequential Extreme learning Machine.

TABLE OF CONTENTS

TABLE OF FIGURES (Soure :Wikipedia)

SECTION 1: INTRODUCTION

1.1 Project Objectives

This report has two main objectives, the first is to present the beneficial features of neural networks and fuzzy logic and the way in which hybrid systems containing each discipline can be beneficial. This will be a purely theoretical discussion drawing on the research carried out at the beginning of the project.
The second section is concerned with presenting performance evaluatin of modified version of recently developed OS Fuzzy ELM and SRAN using multi category classification problems via VC,GI and IS Data Sets from UCI, Machine learning repository (A repository of databases, domain theories and data generators that are used by machine learning community for empirical analysis of machine).

1.2 Report Structure

This report is divided into seven Sections which each cover certain aspects of the project. This Section serves as an overview of what the aims of the report are and what the author has decided to include in each of the sections.

Section 1 is concerned with giving a brief introduction to project objectives including report structure.

Section 2 is concerned with giving a brief introduction to neural networks and fuzzy logic covering each discipline in turn. It then focuses on certain hybrid systems which combine both neural networks and fuzzy logic.

Section 3 is concerned with giving a brief introduction to varius adaptive algorithms eg Least Mean Square (LMS), Recursive Least Square (RLS) Algorithms.

Section 4 is concerned with Matlab Basics

Section 5 is concerned with Extreme learning machine " Theory and applications ".

Section 5.1 is concerned with SRAN Learning Algorithm.

Section 6.0 is concerned with Modified OS-Fuzzy ELM.

Section 7 is concerned with Summary of the simulation results using MATLAB.

Section 7.1 is concerned with Conclusion.

Section 8 is concerned with Suggestion for possible ways to improve further this work within the area and any other methods which could be investigated.

SECTION 2 : THEORY (Neural Fuzzy Networks)

2.1 Neuro-fuzzy systems; Overview

Over the last few decades, neural networks and fuzzy systems have established their reputation as alternative approaches to signal processing. Both have certain advantages over conventional methods, especially when vague data or prior knowledge is involved. However, their applicability suffered from several weaknesses of the individual models.

Therefore, combinations of neural networks with fuzzy systems have been proposed, where both models complement each other. These neural fuzzy or neuro-fuzzy systems overcome some of the individual weaknesses and offer some appealing features. Detailed in the next section is a brief overview of what neural networks are and a brief description of fuzzy logic. The final section deals in depth with a few examples of neuro-fuzzy systems and the benefits they have over conventional non-hybrid alternatives.

2.2 Artificial Neural Networks

Neural Networks is system modelled on the human brain and has many different names such as neuro-computing or intelligent systems. It is an attempt to simulate within specialized hardware or sophisticated software, the multiple layers of simple processing elements called neurons. Each neuron is linked to the one next to it with varying coefficients of connectivity which represents the strengths of these connections. Learning is accomplished by adjusting these strengths to cause the overall network to output appropriate results.

Basic components of neural networks are modelled on the structure of the brain. Some neural network structures are not closely linked to the brain and some do not have an exact biological counterpart. However, neural networks are generally modelled on the biological brain and therefore a great deal of the terminology tends to be borrowed from neuroscience.

2.2.1 The Biological Neuron

The basic elements of the human brain are a specific type of cell/☹neurons), which provides us with the ability to remember, think, and apply previous experiences to our current actions. These cells are known as neurons, each of these neurons can connect with up to 200000 other neurons. The power of the brain comes from the numbers of these basic components and the multiple connections between them. The brain is around 10,000,000,000 times more efficient than the computer chip in terms of energy consumption per operation. The brain is a multi layer structure (about 6-7 layers of neurons) that works as a parallel computer capable of learning from the "feedback" it receives from the world and changing its design (think of the computer hardware changing while performing the task) by growing new neural links between neurons or altering the activities of existing ones.

All natural neurons have four basic components, which are dendrites, soma, axon, and synapses. A biological neuron receives inputs from other sources, combines them in some way, performs a general nonlinear operation on the result, and then outputs the final result. The figure below shows a simplified biological neuron and the relationship of its four components.

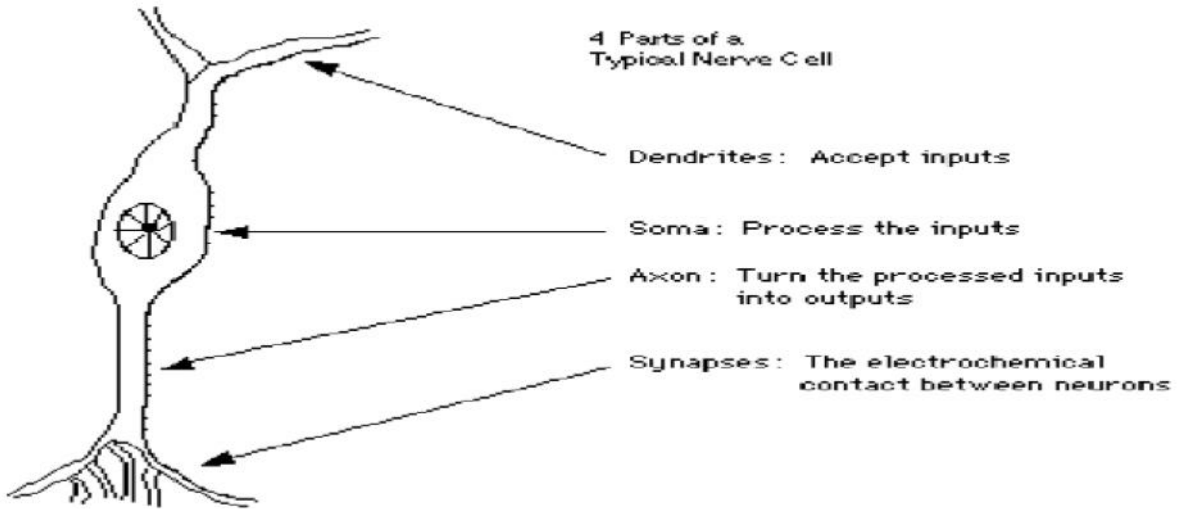
4 Parts of a
Typical Nerve Cell
Dendrites: Accept inputs
Soma: Process the inputs
Axon: Turn the processed inputs into outputs
Synapses: The electrochemical contact between neurons

Figure 1: Biological Neuron (sourse : Wikipedia)

2.2.2 The Artificial Neuron

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

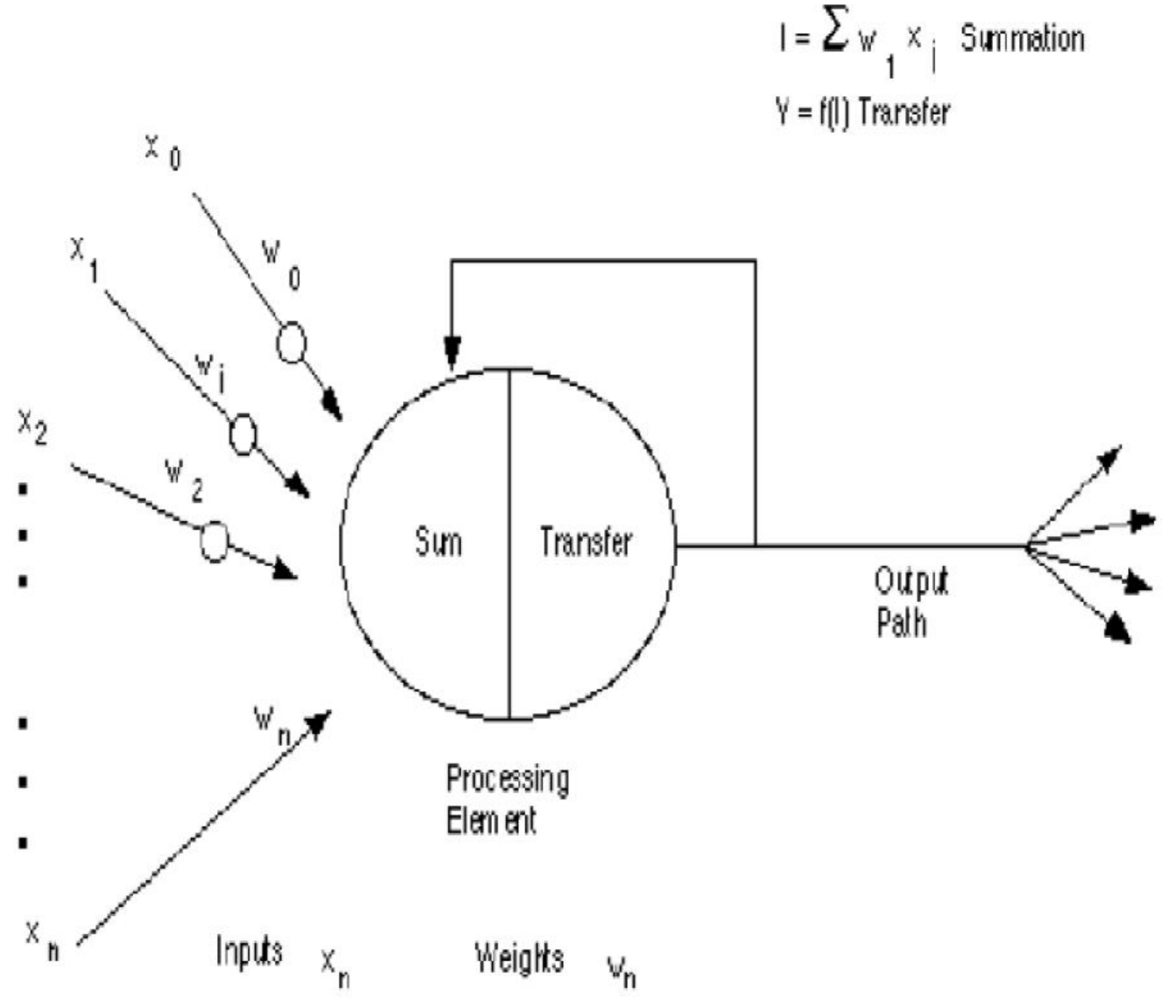

Figure 2: Artificial Neuron (sourse : Wikipedia)

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

2.2.3 Layers

Biologically, neural networks are constructed in three dimensions from microscopic components, and these neurons seem capable of nearly unrestricted interconnections which are not possible in any man-made network. Artificial neural networks are the simple clustering of primitive artificial neurons. This clustering occurs by creating layers, which are then connected to one another. All artificial neural networks have a similar topology, some of the neurons interface with the real world to receive inputs and other neurons provide the real world with the network's outputs. All the rest of the neurons are hidden from view.

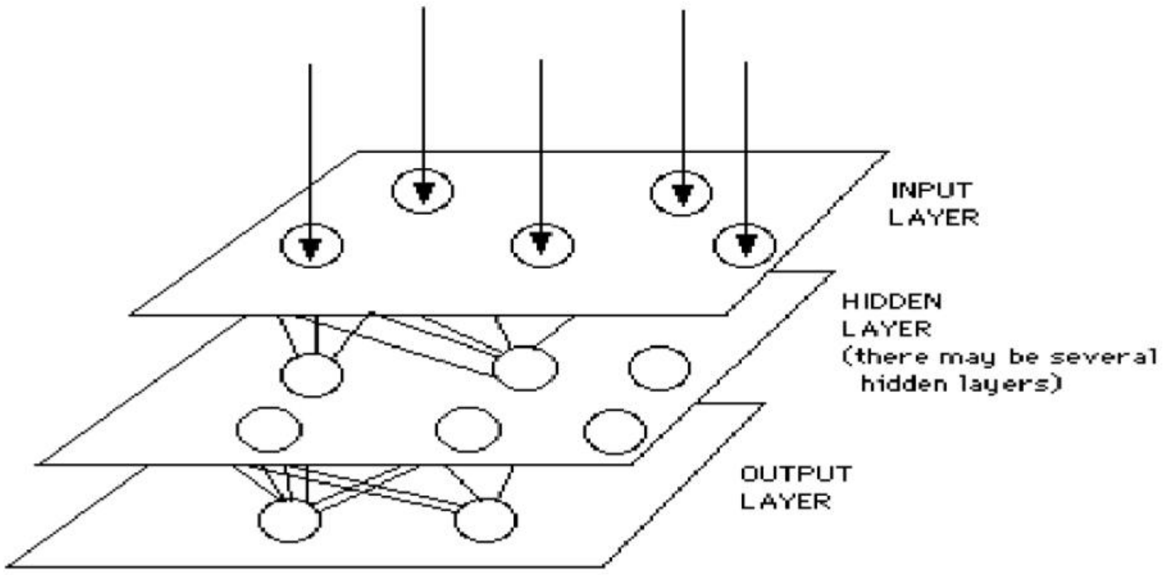

Figure 3: Neural Network Layers (sourse : Wikipedia)

The simple case above shows, the neurons are grouped into three separate layers, the input layer, the hidden layer and the output layer.

Input layer consists of neurons that receive inputs form external environment. The output layer consists of neurons that communicate the outputs of the system to the user or external environment. There are usually a number of hidden layers between these two layers; the figure above shows a simple structure with only one hidden layer.

When the input layer receives the input its neurons produce output, which becomes input to the other layers of the system. The process continues until a certain condition is satisfied or until the output layer is invoked and fires their output to the external environment.

2.2.4 Learning

Neural networks are sometimes called machine learning algorithms, because changing of its connection weights (training) causes the network to learn the solution to a problem. The strength of connection between the neurons is stored as a weight-value for the specific connection. The system learns new knowledge by adjusting these connection weights. The learning ability of a neural network is determined by its architecture and by the algorithmic method chosen for training. The training method usually consists of one of three schemes :

Unsupervised Learning - The hidden neurons must find a way to organize themselves without help from the outside world. In this approach, no sample outputs are provided to the network against which it can measure its predictive performance for a given vector of inputs.

Reinforcement learning - This method works on reinforcement from the outside. The connections among the neurons in the hidden layer are randomly arranged and reshuffled as the network is told how close it is to solving the problem. Reinforcement learning is also called supervised learning, because it requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the network results.

Both unsupervised and reinforcement suffers from relative slowness and inefficiency relying on a random shuffling to find the proper connection weights.

Backpropogation - This method has proven highly successful in training of multilayered neural networks. The network is not just given reinforcement for how it is doing on a task. Information about errors is also filtered back through the system and is used to adjust the connections between the layers, thus improving performance.

Neural networks can be used if training data is available. It is not necessary to have a mathematical model of the problem of interest and there is no need to provide any form of prior knowledge. On the other hand the solution obtained from the learning process usually cannot be interpreted. Although there are some approaches to extract rules from neural networks most neural network architectures are black boxes. They cannot be checked whether their solution is plausible, i.e. their final state cannot be interpreted in terms of rules. This also means that a neural network usually cannot be initialized with prior knowledge if it is available and thus the network must learn from scratch. The learning process itself can take very long, and there is usually no guarantee of success.

Real-life applications : The tasks artificial neural networks are applied to tend to fall within the following broad categories:

Function approximation, or regression analysis, including time series prediction, fitness approximation and modeling.

- Classification, including pattern and sequence recognition, novelty etection and sequential decision making.
- Data processing, including filtering, clustering, blind source separation and compression.
- Robotics, including directing manipulators, Computer numerical control.

Application areas include system identification and control (vehicle control, process control), quantum chemistry,[2] game-playing and decision making (backgammon, chess, racing), pattern recognition (radar systems, face identification, object recognition and more), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications (automated trading systems), data mining (or knowledge discovery in databases, "KDD"), visualization and e-mail spam filtering.

2.3 Fuzzy Logic

2.3.1 Background

Fuzzy sets were first introduced by Zadeh in 1965 to represent data and information which had non-statistical uncertainties. They were designed to mathematically represent uncertainty and vagueness and to provide formalized tools for dealing with the imprecision common to many problems.
The theories behind fuzzy logic started as early as Aristotle when he suggested his "Laws of Thought". One of these laws states that every statement must either be True or False, however even at this time there was doubt and numerous others proposed the possibility that certain statements could be both True and Not True. General opinion then changed to incorporate the possibility that there was a third possibility which was neither True nor False but was an "in between" value. This was first proposed by Lukasiewicz around 1920, when he described a three valued logic, along with the mathematics to accompany it. The third value he proposed can best be translated as the term "possible," and he assigned it a numeric value between **T** and **F**. Later, he explored four valued logics, five-valued logics, and then declared that in principle there was nothing to prevent the derivation of an infinite-valued logic. The notion of an infinite-valued logic was introduced in Zadeh's work "Fuzzy Sets" where he described the mathematics of fuzzy set theory, which lead on by extension to fuzzy logic. This theory proposed making the values True and False operate over the range of real numbers [0, 1].

2.3.2 The concept

Fuzzy logic provides a means to enable approximate human reasoning capabilities to be applied to knowledge-based systems. The theory of fuzzy logic provides a mathematical strength to represent the uncertainties associated with human cognitive processes, such as thinking and reasoning. The conventional approaches to knowledge representation lack the means for representing the meaning of fuzzy concepts. As a consequence, the approaches based on first order logic and classical probability theory do not provide an appropriate conceptual framework for dealing with the representation of common-sense knowledge, since such knowledge is by its nature imprecise. The development of fuzzy logic was motivated by the need for a conceptual framework which can address the issue of uncertainty and lexical imprecision.

Using fuzzy set theory it is easy to model the idea of fuzzy theory by using gradual memberships. In contrast to conventional set theory, in which an object or a case either is a member of a given set or not, fuzzy set theory makes it ossible that an object or a case belongs to a set only to a certain degree. Interpretations of membership degrees include similarity, preference, and uncertainty. They can state how similar an object or case is to a prototypical one, they can indicate preferences between sub optimal solutions to a problem, or they can model uncertainty about the true situation, if this situation is described in imprecise terms.

The **antecedent** of a rule consists of fuzzy descriptions of input values, and the **consequent** defines an output value for the given input. The benefits of these fuzzy systems lie in the suitable knowledge representation. But problems may arise when fuzzy concepts have to be represented by static membership degrees, which guarantee that a fuzzy system works as expected.

A fuzzy system can be used to solve a problem if knowledge about the solution is available in the form of linguistic if-then rules. By defining suitable fuzzy sets to represent linguistic terms used within the rules, a fuzzy system can be created from these rules. There is no formal model of the problem of interest and no training data required.

2.3.3 Summary

Listed below are the main features of fuzzy logic:

☆ .In fuzzy logic, exact reasoning is viewed as a limiting case of approximate reasoning.
☆. In fuzzy logic, knowledge is seen as a collection of fuzzy constraints on a set of variables.
☆. Any logical system can be fuzzified. There are two main characteristics of fuzzy systems that give them better performance for specific applications.
☆. Fuzzy systems are suitable for uncertain or approximate reasoning,especially for a system with a mathematical model that is difficult to derive.
☆. Fuzzy logic allows decision making with estimated values under incomplete or uncertain information.

2.4 Fuzzy neural networks

The basic idea of combining fuzzy systems and neural networks is to design an architecture that uses a fuzzy system to represent knowledge in an interpretable manner, in addition to possessing the learning ability of aneural network to ptimize its parameters. The drawbacks of both of the individual approaches - the black box behavior of neural networks, and the problems of finding suitable membership values for fuzzy systems could thus be avoided. A combination can constitute an interpretable model that is capable of learning and can use roblem-specific prior knowledge. Therefore, neuro-fuzzy methods are especially suited for applications, where user interaction in model design or interpretation is desired.

2.4.1 Combining fuzzy logic and neural networks

Linguistic rules and membership functions to represent linguistic values have to be determined when designing a fuzzy controller. The effectiveness of these linguistic rules depends on the knowledge of the control expert. However, the translation of these into fuzzy set theory is not straightforward and choices concerning the shape of membership functions, amongst others, have to be made. Therefore the quality of the fuzzy controller can be influenced by changing the shapes of membership functions, so methods for tuning fuzzy controllers are necessary.

Artificial neural networks are highly parallel architectures consisting of simple processing elements which communicate through weighted connections. They are able to approximate functions or to solve certain tasks by learning from examples. Neural networks can also be used for control problems and they are able to learn supervised or unsupervised from given data. The problem is that the learning process takes a lot of time and is not guaranteed to be successful. Furthermore it is not possible to integrate prior knowledge to simplify the learning process or to extract knowledge e.g. in the form of rules from the trained network.

A combination of both approaches offers the possibility to solve the tuning and design problem of fuzzy control. The combination of neural networks and fuzzy controllers assembles the advantages of both approaches and avoids the drawbacks of them. Cooperative approaches exist which use neural networks to optimize certain parameters of an ordinary fuzzy controller or to pre-process data and extract fuzzy control rules. Hybrid architectures integrate the concepts of a fuzzy controller into a neural network or understand a fuzzy controller as a neural network.

2.4.2 Neural networks to tune fuzzy parameters

The choice of a particular membership function representing a linguistic term is more or less arbitrary. For example, consider the linguistic term to be approximately zero. Obviously the corresponding fuzzy set should be a uni-modal function reaching its maximum at the value zero. Neither the shape nor the range - i.e. the support of the membership function is uniquely determined by the term 'approximately zero'. Generally the control expert has an uderstanding of the range of the membership function but may not be able to argue about small changes of the specified range. Therefore the tuning of membership functions becomes an important issue in fuzzy control. Since this tuning task can be viewed as an optimization problem, neural networks offer a possibility to solve this problem. A straightforward approach is to assume a certain shape for the membership functions which depends on different parameters that can be learned by a neural network. This can be carried out for example where the membership functions are assumed to be symmetrical triangular functions depending on two parameters, one of them determining where the function reaches its maximum, the other giving the length of the support.

Gaussian membership functions can also be used. Both approaches require a set of training data in the form correct input-output tuples and a specification of the rules including a preliminary definition of the corresponding membership functions.

Detailed below is one method which can cope with arbitrary membership functions for the input variables. The training data must be divided into r disjoint clusters $R1......Rr$. Each cluster Ri corresponds to a control rule Ri.

Elements of the clusters are tuples of input-output values of the form (x, y) where x can be a vector $x=(x1,.....xn)$ of n input variables.

This means that the rules are not specified in terms of linguistic variables but in the form of crisp input-output tuples.
A multilayer perceptron with n input units, some hidden layers, and r output units can be used to learn these clusters. The input data for this learning task are the input vectors of all clusters.
After the network has learned its weights, arbitrary values for *x* can be taken as inputs. Then the output at output unit $u_i$can be interpreted as the degree to which give input *x* matches the antecedent of rule R_i, Is the membership function for the fuzzy set representing the linguistic term on the left-hand side of rule R_i.

2.4.3 Extracting fuzzy rules with neural networks

If a set of input-output tuples for a given control problem is available, we can try to extract fuzzy control rules from this set. This can either be done by fuzzy clustering methods, or by using neural networks.
The input vectors of the input-output tuples can be taken as inputs for a *Kohonen self-organizing map1*, which can be interpreted in terms of linguistic variables. The main idea for this interpretation is to refrain from the winner take all principle after the weights for the self-organizing map are learned.

Thus each output unit corresponds to an antecedent of a fuzzy control rule. Depending on how far output unit *ui* is from being the 'winner' given input vector *x*, a matching degree (*x*) *i m* can be specified, yielding the degree to which *x* satisfies the antecedent of the corresponding rule.

2.4.4 Neural networks with fuzzy control

Another approach to combine fuzzy control and neural networks is to create a special architecture out of standard feed-forward nets that can be interpreted as a fuzzy controller. It is a neural network model of a fuzzy 1 Kohonen's self organising map is an example of a neural net which clusters rather than classifies. It uses unsupervised learning rather than the supervised learning of feed forward neural neworks.
controller which learns by updating its prediction of the physical system's behaviour and fine tunes a predefined control knowledge base.
This kind of architecture means you are able to combine the advantages of neural networks and fuzzy controllers. The system is able to learn, and the knowledge used within the system has the form of fuzzy if-then rules.
By predefining these rules with the help of an experienced operator the system has not to learn from scratch, so it learns faster than a standard neural control system.
This architecture consists of two special feed-forward neural networks, the action-state evaluation network (AEN) and the action selection network (ASN). The ASN is a multilayer neural network representation of a fuzzy controller. In fact, it consists of two separated nets, where the first one is the fuzzy inference part and the second one is a neural network that calculates $p[t, t+1]$, a measure of confidence associated with the fuzzy inference value $u(t+1)$ using the weights of time t and the system state of time $t+1$. This value can be seen as the 'probability'. The recommended control value $u(t)$ of the fuzzy inference part and the probability value p are combined to determine the final output value $u'(t) = o(U(t), p[t, t+1])$ of the ASN. The hidden units i z of the fuzzy inference network represent the fuzzy rules, the input units j x the rule antecedents, and the output unit u represents the control action, that is the de-fuzzified combination of the conclusions of all rules (output of hidden units).

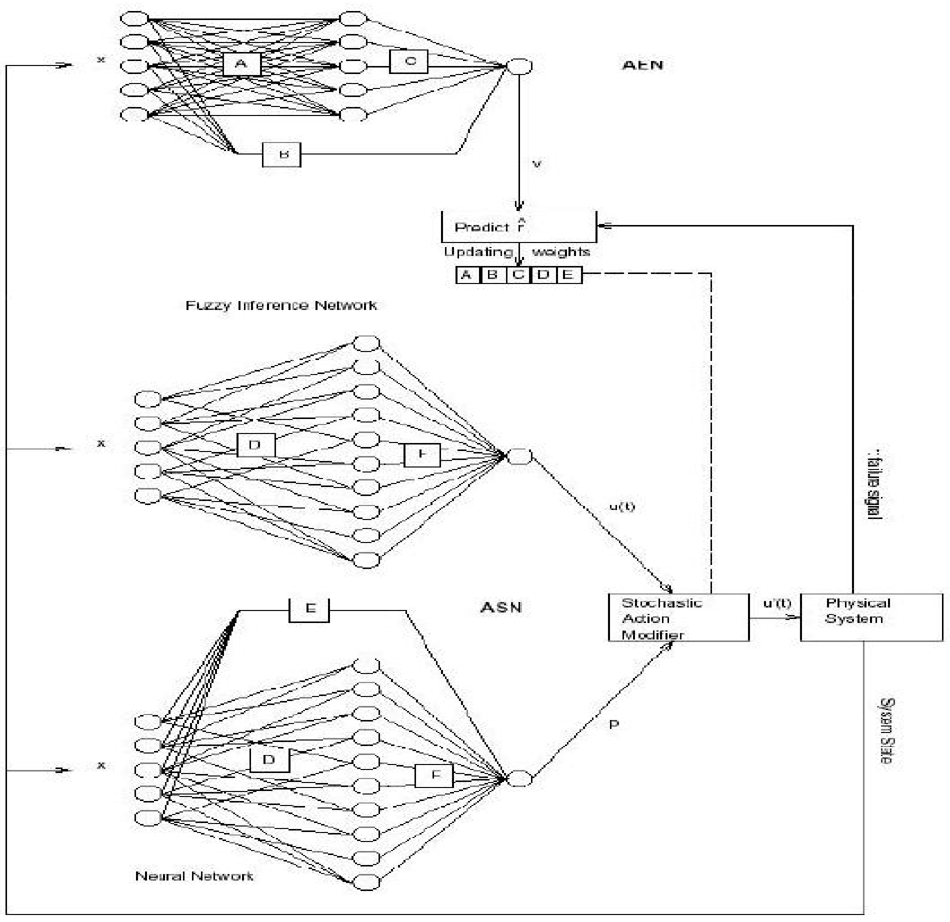

Figure 4: Fuzzy controlled neural network (source : Wikipedia)

The AEN is a feed-forward neural network with one hidden layer, which receives the system state as its input and an error signal *r* from the physical system as additional information.

It evaluates the current state and outputs a reinforcement signal used to calculate the weight changes in the system.
This is generally known as a type of fuzzy control oriented neural network, it is a system that consists of more or less standard neural networks, where one network employs a fuzzy aspect in using membership functions within its units and a fuzzy inference scheme.

SECTION 3: Introduction to Adaptive algorithms

3.1 Adaptive algorithms Introduction

Many computationally efficient algorithms for adaptive filtering have been developed within the past twenty years. They are based on either a statistical approach, such as the least-mean square (LMS) algorithm, or a deterministic approach, such as the recursive least-squares (RLS) algorithm. The major advantage of the LMS algorithm is its computational simplicity. The RLS algorithm, conversely, offers faster convergence, but with a higher degree of computational complexity.

3.2 Applications

Adaptive filters are widely used in telecommunications, control systems, radar systems, and in other systems where minimal information is available about the incoming signal.

Due to the variety of implementation options for adaptive filters, many aspects of adaptive filter design, as well as the development of some of the adaptive algorithms, are governed by the applications themselves.
Several applications of adaptive filters based on FIR filter structures are described below.

3.2.1 System Identification

One can design controls for a dynamic system if you have a model that describes the system as it changes, although this modelling is not easy with complex physical phenomena. Information about the system to be controlled can be gained by collecting experimental data of system responses to given events. This process of constructing models and estimating the best values of unknown parameters from experimental data is called system identification.

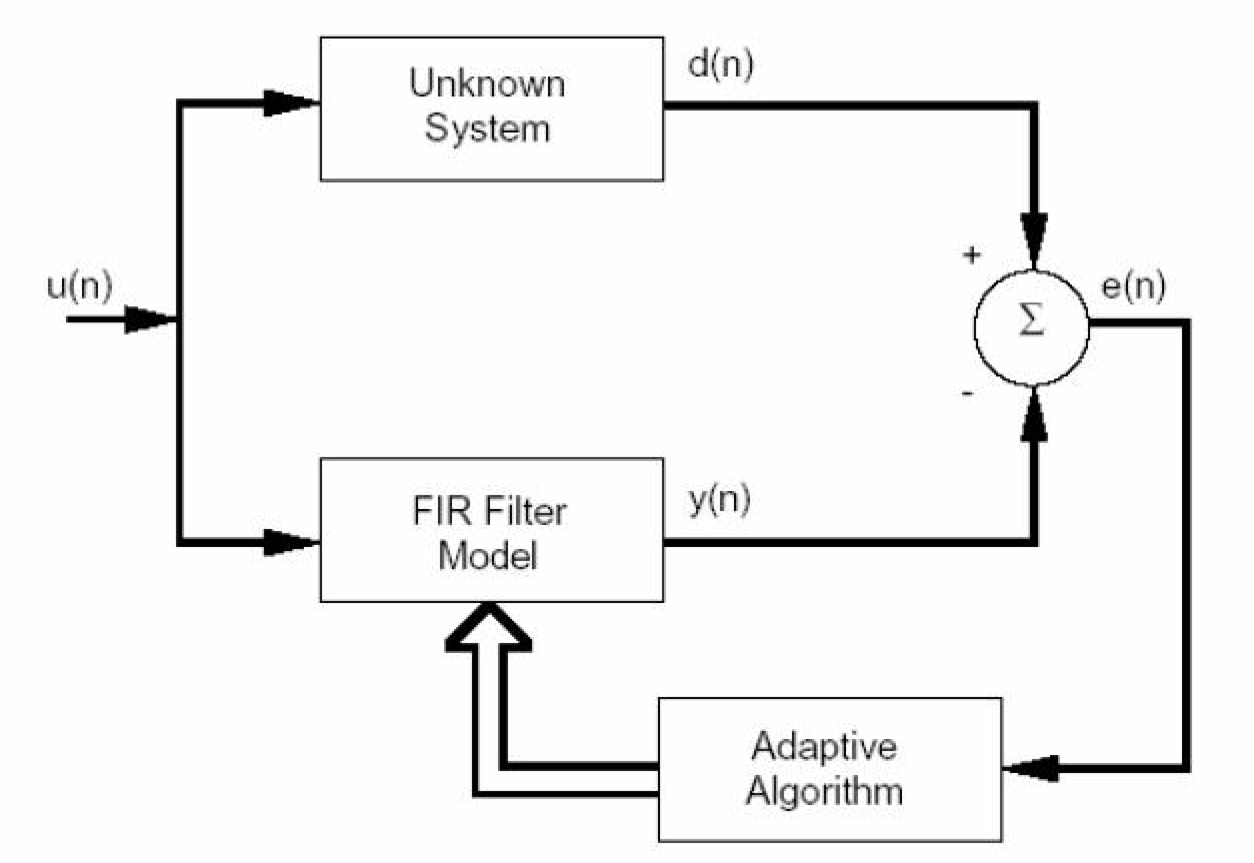

Figure 5: Adaptive filter setup (source : Wikipedia)

Figure shows a block diagram of the system identification model. The unknown system is modelled by an FIR filter with adjustable coefficients.
Both the unknown time-variant system and FIR filter model are altered by an input sequence u(n). The adaptive FIR filter output y(n) is compared with the unknown system output d(n) to produce an estimation error e(n).

The estimation error represents the difference between the unknown system output and the model (estimated) output. The estimation error e(n) is then used as the input to an adaptive control algorithm which corrects the individual tap weights of the filter. This process is repeated through several iterations until the estimation error e(n) becomes sufficiently small in some statistical sense. The resultant FIR filter response now represents that of the previously unknown system.

3.2.2 Equalization for data transmission

Adaptive filters are used widely to provide equalization in modems that transmit data over numerous bandwidth channels. An adaptive equalizer is used to compensate for the distortion caused by the transmission medium. Its operation involves a training mode followed by a tracking mode.
The equalizer is trained by transmitting a known test data sequence u(n).
A synchronized version of the test signal is generated in the receiver, meaning the adaptive equalizer is now supplied with a desired response d(n). The equalizer output y(n) is subtracted from this desired response to give an estimation error. This estimation error is used to adaptively adjust the coefficients of the equalizer to their optimum values. When the training is completed, the adaptive equalizer tracks possible time variations in channel characteristics during transmission. It does this by using a receiver estimate of the transmitted sequence as a desired response. The receiver estimate is obtained by applying the equalizer output y(n) to a decision device.

3.2.3 Echo Cancellation

Dial-up switched telephone networks are used for low-volume infrequent data transmission, these are what current 56K connections utilise. It is a device called a "hybrid" which provides full-duplex operation, transmit and receive channels, from a two-wire telephone line. As there is an impedance mismatch between the hybrid and the telephone channel, an echo is generated which can be suppressed by adaptive echo cancellers installed in the network in pairs.

This cancellation is carried out by creating an estimate of the echo signal components by using the transmitted sequence u(n) as input data, and then subtracting the estimate y(n) from the sampled received signal d(n).

The error signal can be minimized, in the least-squares sense, to adjust optimally the weights of the echo canceller. Similar applications include the suppression of narrowband interference in a wideband signal, adaptive line enhancement, and adaptive noise cancellation.

3.2.4 Noise Cancellation

Adaptive filtering can be extremely useful in cases where a speech signal is distorted in a very noisy environment with many periodic components lying in the same bandwidth as the speech. The adaptive noise canceller for speech signals needs two inputs. The main input containing the voice which is corrupted by noise. The other input (noise reference input) contains noise related in some way to that of the main input (background noise).

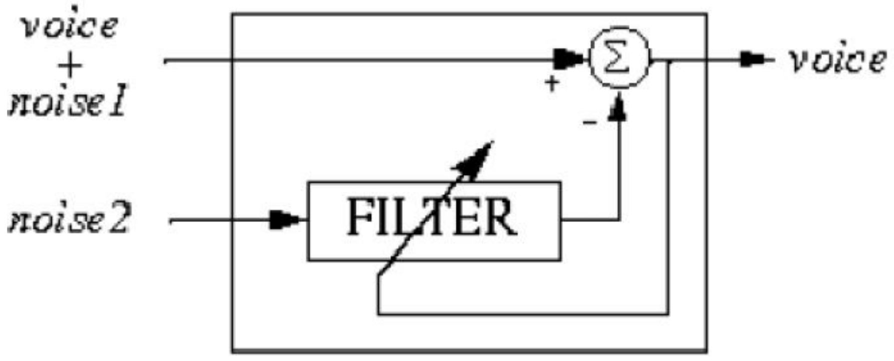

Figure 6: Adaptive noise filter (source : wikipedia)

The system filters the noise reference signal making it similar to the original signal and that filtered version is subtracted from the original signal. In an ideal situation all of the noise will be removed leaving the original signal in tact.

A good example to consider is a pilot in a fast jet talking on a radio as this speech signal will undoubtedly have a significant noise component. The microphone situated at the pilot's mouth will pick up the pilot's voice along with a high level of noise with strong periodic components, which could be substantially reduced by using an adaptive noise canceller. The main input to that system would be the pilot's microphone with a secondary noise reference input needed. This can be obtained from a second microphone located somewhere else in the cockpit, far enough the from the pilot's mouth so that it only picks up noise. The noise cancellation system would then try to minimize the error signal power. As the noise reference input does not contain any information about the speech signal, the speech will not be removed but the noise accompanying the speech will be. The output of the noise canceller (the error signal) is then the
pilot's voice with a much lower level of noise.

3.3 Adaptive Filter Algorithms

In this section the Least Means Squares algorithm and the Recursive Least Squares algorithms are discussed. LMS algorithms are based on a gradient-type search for tracking time-varying signal characteristics, while RLS algorithms provide faster convergence and better tracking of time variant signal statistics than LMS Algorithms, but are more complex computationally.
An adaptive system uses the gradient to optimize its parameters, however the gradient is usually not known so had to be estimated. A good estimate requires many small perturbations to the operating point to obtain a robust estimation

through averaging. Although this method is straightforward it is not very practical.

An algorithm to estimate the gradient was then proposed by Widrow in the 1960s that revolutionized the application of gradient descent procedures, as this algorithm used the instantaneous value as the estimator for the true quantity. This algorithm is now known as the Least Mean Squares Algorithm or LMS algorithm for short.

The LMS algorithm as mentioned is concerned with optimising the mean square error (i.e. the mean square value of the error signal). This is achieved by adjusting the values of the weights of the FIR filter which is detailed below.

3.3.1 The FIR Filter

The N tap FIR filter consists of (N-1) delayers, N multipliers, each with its corresponding weight, and N-1 adders (or a global adder):

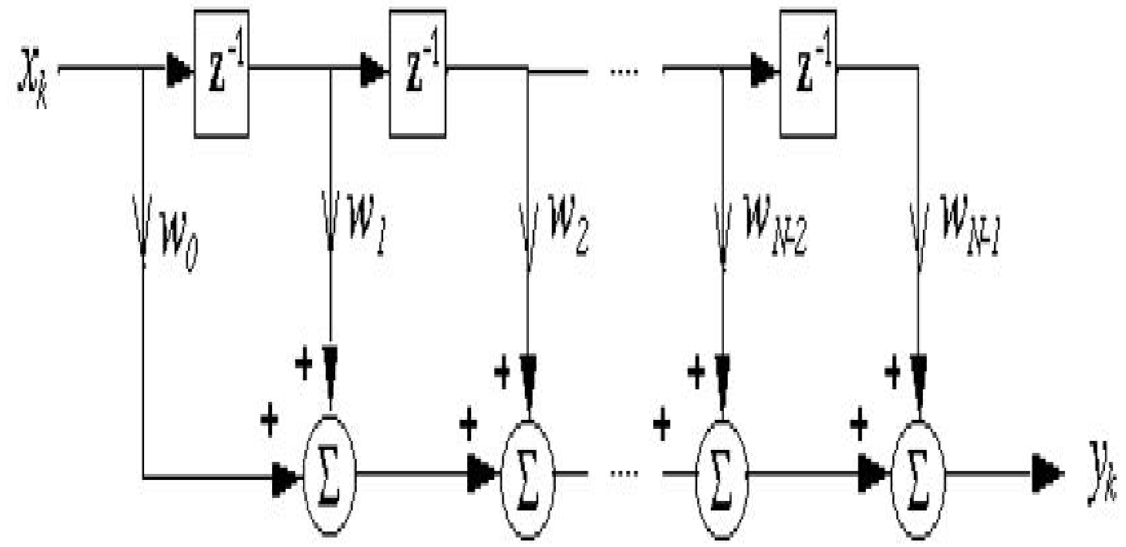

Figure 7: The FIR Filter (source : Wikipedia)

The values of these weights determine the frequency response of the filter. Algebraically, the expression for the output is:

Equation 1: Output of the FIR Filter

$$Y_k = \sum_{k=1}^{\infty} (x_k - jw_j)$$

This equation can also be expressed in vectorial notation with *k X* and *W* as vectors.

$$Y_k = [\mathrm{W}]^T \; x_k$$

FIR stands for Finite Impulse Response as the impulse response of the filter is as the name suggests finite in duration. The filter has a duration of N, where N is the number of taps of the filter, and the weights are the values of the N samples of the impulse response.

The LMS algorithm is an approximation of the Steepest Descent Method using an estimator of the gradient instead of the actual value of the gradient. This considerably reduces the number of calculations to be performed and this allows the LMS algorithm to be implemented in real time applications.

3.3.2 Steepest Decent Method

The gradient of a surface is a vector in the domain that points to the direction of maximum increase in the value of the surface. Therefore going in the opposite direction (given by ␣␣␣) would mean moving towards the minimum.
Ensuring that the weight vector changes in relation to the gradient in each iteration is what is known as "the Steepest Descent Method" :

Equation 2: Steepest decent method

$$W_{k+1} = W_k + \mu (-\nabla_k \quad)$$

The step size (µ) is important in the speed of convergence and in the stability of the LMS algorithm.

3.3.3 The LMS Algorithm

The LMS algorithm is initialized by setting all the weights to zero at time n=0. Tap weights are updated using the relationship:

Equation 3: Least Mean Squares Algorithm

$$W_{n+1} = W_n + \mu\, e(n)\, \mathrm{u(n)}$$

Here, w(n) represents the tap weights of the transversal filter, e(n) is the error signal, u(n) represents the tap inputs, and the factor μ is the adaptation parameter or step-size. To ensure convergence, μ must satisfy the condition

$$0 < \mu < (2 / tp)$$

In the above equation the value tp is the total input power which refers to the sum of the mean-square values of the tap inputs and the convergence time depends on the ratio of the maximum to minimum eigenvalues taken from the autocorrelation matrix R of the input signal. To ensure that μ does not become sufficiently large to cause filter instability, a Normalized LMS algorithm can be employed. The normalized LMS employs a timevarying
μ defined as:

$$\mu = \frac{x}{uT(n).u(n)}$$

Here, x is the normalized step-size between 0 and 2 and tap weights are updated according to the relationship:

$$w(n+1) = w(n) + \frac{x\, e(n)u(n)}{r + uT(n).u(n)}$$

The term x is the new normalized adaptation constant, while r is a small positive term included to ensure that the update term does not become excessively large when $uT(n)u(n)$ temporarily becomes small.

A problem can occur when the autocorrelation matrix associated with the input process has one or more zero eigenvalues. In this case, the adaptive filter will not converge to a unique solution. In addition, some uncoupled coefficients (weights) may grow without bound until hardware overflow or underflow occurs. This problem can be remedied by using coefficient leakage.

This "leaky" LMS algorithm can be written as:

$$W_{n+1} = (1 - \mu r)W_n + \mu\, e(n)\, u(n)$$

Here the adaptation constant μ and the leakage coefficient r are all small positive values.

3.3.4 Other LMS Variants

3.3.4.1 Block LMS

The weight vector of the FIR filter is held constant for a few iterations while an improved estimate of the performance surface gradient is obtained.

3.3.4.2 Variable Step Size

The value of μ is chosen large at the beginning and then is progressively reduced to a smaller size to iterate closer to the optimum value.

3.3.4.3 Sign Error LMS

The computation needed by the adaptive algorithm can be reduced to zero multiplications and N additions using only the sign of the error signal (and making μ be a power of two):

3.3.5 RLS Algorithm

The LMS algorithm has many advantages (due to its computational simplicity), but its convergence rate is slow. The LMS algorithm has only one adjustable parameter that affects convergence rate, the step-size parameter μ , which has a limited range of adjustment in order to insure stability.

For faster rates of convergence, more complex algorithms with additional parameters must be used. The RLS algorithm uses a least-squares method to estimate correlation directly from the input data. The LMS algorithm uses the statistical mean-squared-error method, which is slower. The RLS algorithm uses a transversal FIR filter implementation.

The order of operations the algorithm takes is:

1)COMPUTE THE FILTER OUTPUT (TAP WEIGHTS INITIALIZED TO ZERO)

2)FIND THE ERROR SIGNAL

3)COMPUTE THE KALMAN GAIN VECTOR

4)UPDATE THE INVERSE OF THE CORRELATION MATRIX

5)UPDATE THE TAP WEIGHTS

The Kalman Gain Vector is computed based on a number of are as :
*Input-data autocorrelation results
**The input data itself
***The 'forgetting' factor.
This forgetting factor is a value which ranges between 0 and 1 and provides a time-weighting of the input data such that the most recent data points are weighted more heavily than past data. This allows the filter coefficients to adapt to the characteristics of the input data. The tap weight update is based on the error signal and the Kalman Gain Vector and is expressed as :

$$W_n = W_{n-1} + \mathrm{k}\, e(n)$$

Here K is the Kalman Gain Vector and e(n) represents the error signal.

SECTION 4: MATLAB Basics

In the previous Section numerous application of adaptive filters were touched upon and two of the main algorithms that do this were looked at.
This Section presents the MATLAB as whole process will be carried out in the MATLAB software environment as this is the most effective tool for displaying the data being handled and for executing the noise cancellation algorithm itself. Details on different parts of the program as they were being written are also included to aid understanding of the process from design to completion.

4.1 MATLAB

MATLAB is a high-performance language for technical computing and it has excellent computation, visualization, and programming capabilities. It has an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows the solving of many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran.

The name MATLAB stands for **MA**trix **LAB**oratory. MATLAB has changed over the years by means of feedback from users suggesting changes and MATLAB is the standard instructional tool for courses in mathematics, engineering, and science and in industry.MATLAB is generally the tool of choice for research, development, and analysis. It also has a set of application-specific solutions called toolboxes. Toolboxes are comprehensive collections of MATLAB functions that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, neural networks and fuzzy logic.

4.1.1 MATLAB Components

There are five main parts to the MATLAB system:

Programming language: This is a high-level language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It is suitable for small scale and large scale programs alike.

Working environment: This is a set of tools which are available when you first load up MATLAB that allow you to manipulate all features of the software. It includes facilities for managing the variables in your workspace (workspace browser) and importing and exporting data. It also includes tools for developing, managing, debugging, and profiling M-files which are the file types which you produce when writing a program in MATLAB.

Graphics: The MATLAB graphics system contains high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also allows more experienced users to utilise more low level commands and have more control over the interface. This also allows the creation of graphical user interfaces using such features as the GUI Development Environment of GUIDE for short.

Mathematical Function Library: This includes a large number of computational algorithms ranging from the common ones like sum, sine, cosine, and complex arithmetic, to more advanced features like matrix inverse, matrix eigenvalues and fast Fourier transforms.

Application Program Interface (API): This is a function that allows you to write C and Fortran programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

4.1.2 Creating a program

To create a program in MATLAB the most effective way is to just type "edit" at the command prompt. This will bring up the default editor and will allow you to save the program as an 'm-file' which is the default program type in MATLAB.
Once you have saved the program you can execute it by pressing the F5 button, any errors which occur will be displayed in the command window along with the line on which they occurred and the data involved.

Section 5 : Extreme learning machine “ Theory and applications “

It is clear that the learning speed of feedforward neural networks is in general far slower than required and it has been a major bottleneck in their applications for past decades.
Two key reasons behind may be : (1) the slow gradient-based learning algorithms are extensively used to train neural networks, and (2) all the parameters of the networks are tuned iteratively by using such learning algorithms. Unlike these conventional implementations, this paper proposes a new learning algorithm called “extreme learning machine (ELM)” for single-hidden layer feedforward neural networks (SLFNs) which randomly chooses hidden nodes and analytically determines the output weights of SLFNs. In theory, this algorithm tends to provide good generalization performance at extremely fast learning speed. The experimental results based on a few artificial and real benchmark function approximation and classification problems including very large complex applications show that the new algorithm can produce good generalization performance in most cases and can learn thousands of times faster than conventional popular learning algorithms for feedforward neural networks.

1. Basic background

Radial basis function : A radial basis function (RBF) is a real-valued function whose value depends only on the distance from the origin, so that $\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)$; or alternatively on the distance from some other point *c*, called a *center*, so that $\phi(\mathbf{x}, \mathbf{c}) = \phi(\|\mathbf{x} - \mathbf{c}\|)$. Any function ϕ that satisfies the property $\phi(\mathbf{x}) = \phi(\|\mathbf{x}\|)$ is a radial function. The norm is usually Euclidean distance.Commonly used types of radial basis functions include (writing $r = \|\mathbf{x} - \mathbf{x}_i\|$), eg. Gaussian: $\phi(r) = e^{-(\varepsilon r)^2}$
Approximation: Radial basis functions are typically used to build up function approximations of the form

$$y(\mathbf{x}) = \sum_{i-1}^{N} w_i \, \phi(\|\mathbf{x} - \mathbf{x}_i\|),$$

Where, the approximating function *y*(*x*) is represented as a sum of *N* radial basis functions, each associated with a different center x_i, and weighted by an appropriate coefficient w_i.

The weights w_i can be estimated using the matrix methods of linear least squares, because the approximating function is *linear* in the weights.Approximation schemes of this kind have been particularly used in time series prediction and control of nonlinear systems

Single hidden layer feedforward networks (SLFNs) with random hidden nodes

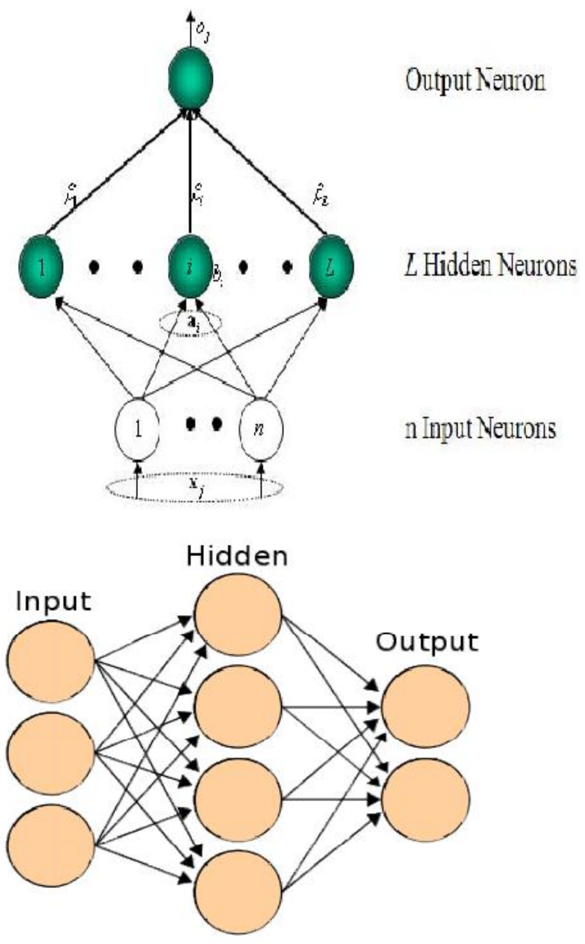

Figure 6 : SLFN, Single Laye feedforward network (source : wikipedia)

Feedforward neural networks have been extensively used in many fields due to their ability: (1) to approximate complex nonlinear mappings directly from the input samples; and (2) to provide models for a large class of natural and artificial phenomena that are difficult to handle using classical parametric techniques. On the other hand, there lack faster learning algorithms for neural networks. The traditional learning algorithms are usually far slower than required. It is not surprising to see that it may take several hours, several days, and even more time to train neural networks by using traditional methods.

From a mathematical point of view, research on the approximation capabilities of feedforward neural networks has focused on two aspects: universal approximation on compact input sets and approximation in a finite set of training samples. Many researchers have explored the universal approximation capabilities of standard multilayer feedforward neural networks. Hornik [7] proved that if the activation function is continuous, bounded and nonconstant, then continuous mappings can be approximated in measure by neural networks over compact input sets. Leshno [17] improved the results of Hornik [7] and proved that feedforward networks with a nonpolynomial activation function can approximate (in measure) continuous functions. In real applications, the neural networks are trained in finite training set. For function approximation in a finite training set, Huang and Babri [11] shows that a single-hidden layer feedforward neural network (SLFN) with at most N hidden nodes and with almost any nonlinear activation function can exactly learn N distinct observations. It should be noted that the input weights (linking the input layer to the first hidden layer) and hidden layer biases need to be adjusted in all these previous theoretical research works as well as in almost all practical learning algorithms of feedforward neural networks.

Traditionally, all the parameters of the feedforward networks need to be tuned and thus there exists the dependency between different layers of parameters (weights and biases). For past decades, gradient descent-based methods have mainly been used in various learning algorithms of feedforward neural networks. However, it is clear that gradient descent-based learning methods are generally very slow due to improper learning steps or may easily converge to local minima. And many iterative learning steps may be

required by such learning algorithms in order to obtain better learning performance.
It has been shown [23,10] that SLFNs (with N hidden nodes) with randomly chosen input weights and hidden layer biases (and such hidden nodes can thus be called random hidden nodes) can exactly learn N distinct observations. Unlike the popular thinking and most practical implementations that all the parameters of the feedforward networks need to be tuned, one may not necessarily adjust the input weights and first hidden layer.

SLFNs can be randomly assigned if the activation functions in the hidden layer are infinitely differentiable. Further for new ELM learning algorithm, single-hidden layer feedforward neural networks (SLFNs) Performance evaluation is presented in next section . Discussions and conclusions are given in later section.

For N arbitrary distinct samples (x_i , t_i) where x_i and t_i, standard SLFNs with N hidden nodes and activation function g(x) are mathematically modeled as

$$\sum_{i=0}^{N} \beta_i \ g(x_j) = \sum_{i=0}^{N} \beta_i \ g(w_i \cdot x_j + b_i) = O_j \quad , \qquad j = 1,2,3,\ldots..,N.$$

where w_i is the weight vector connecting the ith hidden node and the input nodes, β_i is the weight vector connecting the ith hidden node and the output nodes, and b_i is the threshold of the ith hidden node.

$w_i \cdot x_j$ denotes the inner product of w_i and x_j . The output nodes are chosen linear.

Standard SLFNs with N hidden nodes with N activation function g(x) can approximate these N samples with zero error means that

$$\sum_{j=1}^{N} || o_j - t_j || = 0,$$

there exist β_i, w_i and b_i such that

$$\sum_{\mathrm{i=0}}^{\mathrm{N}} \beta_i \ g(w_i \cdot x_j + b_i) = t_j \quad , \qquad \mathrm{j} = 1,2,3,\ldots\ldots,\mathrm{N}.$$

The abve N equations can be compactly written as

$$\hat{\mathrm{H}}\,\beta = \mathrm{T}$$

Mathematical Model

$\sum_{i=1}^{L} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{t}_j, j = 1, \cdots, N$, is equivalent to $\mathbf{H}\beta = \mathbf{T}$, where

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_N) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_N) \end{bmatrix}_{N \times L}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m} \text{ and } \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix}_{N \times m}$$

Where H is called as hidden layer output matrix of neural network.

Three Step Learning Model for ELM

Given a training set ℵ = {(xi,ti) hidden node output function G(a, b, x), and the number of hidden nodes L.

1) ASSIGN RANDOMLY HIDDEN NODE PARAMETERS (AI, BI), I = 1, · · · , L.

2)CALCULATE THE HIDDEN LAYER OUTPUT MATRIX H.

3) CALCULATE THE OUTPUT WEIGHT β: β = H† T

where H† is the Moore-Penrose generalized inverse of hidden layer output Matrix H.

The Moore–Penrose generalized inverse and the minimum norm least-squares solution of a general linear system which play an important role in developing our new ELM learning algorithm are briefed in the Appendix.

In OS-ELM , first it is rigorously proved that the input weights and hidden layer biases of SLFNs can be randomly assigned if the activation functions in the hidden layer are infinitely differentiable. After the input weights and the hidden layer biases are chosen randomly, SLFNs can be simply considered as a linear system and the output weights (linking the hidden layer to the output layer) of SLFNs can be analytically determined through simple generalized inverse operation of the hidden layer output matrices. The output nodes are chosen linear in this paper.

Based on this concept, OS-ELM paper proposes a simple learning algorithm for SLFNs called extreme learning machine (ELM) whose learning speed can be thousands of times faster than traditional feedforward network learning algorithms like back-propagation (BP) algorithm while obtaining better generalization performance. Different from traditional learning algorithms the proposed learning algorithm not only tends to reach the smallest training error but also the smallest norm of weights. Bartlett's [1] theory on the generalization performance of feedforward neural networks states for feedforward neural networks reaching smaller training error.

Salient Features of ELM

"Simple Math is Enough." ELM is a simple tuning-free three-step algorithm.
The learning speed of ELM is extremely fast

The hidden node parameters a_i and b_i are not only independent of the training data but also of each other.

Unlike conventional learning methods which MUST see the training data before generating the hidden node parameters, ELM could generate the hidden node parameters before seeing the training data.

Unlike traditional gradient-based learning algorithms which only work for differentiable activation functions,

ELM works for all bounded nonconstant piecewise continuous activation functions.

Unlike traditional gradient-based learning algorithms facing several issues like local minima, improper learning rate and overfitting, etc, ELM tends to reach the solutions straightforward without such trivial issues.

The ELM learning algorithm looks much simpler than other popular learning algorithms: neural networks and support vector machines.

Learning Features

1) The training observations are sequentially (one-by-one or chunk-by-chunk with varying or fixed chunk length) presented to the learning algorithm.
2) At any time, only the newly arrived single or chunk of observations (instead of the entire past data) are seen and learned.
3) A single or a chunk of training observations is discarded as soon as the learning procedure for that particular (single or chunk of) observation(s) is completed
4) The learning algorithm has no prior knowledge as to how many training observations will be presented.

OS-ELM

Two Step Learning Model

1) INITIALIZATION PHASE: WHERE BATCH ELM IS USED TO INITIALIZE THE LEARNING SYSTEM.

2) SEQUENTIAL LEARNING PHASE: WHERE RECURSIVE LEAST SQUARE (RLS) METHOD IS ADOPTED TO UPDATE THE LEARNING SYSTEM SEQUENTIALLY.

Summary

For generalized SLFNs, learning can be done without iterative tuning.
ELM is efficient for batch mode learning, sequential learning, incremental learning.
ELM provides a unified learning model for regression, binary/multi-class classification.
ELM works with different hidden nodes including kernels.
Real-time learning capabilities.
ELM always provides better generalization performance than SVM and LS-SVM
ELM always has faster learning speed than LS-SVM
ELM is the simplest learning technique for generalized SLFNs

In [28] by Hai-Jun Rong, Guang-Bin Huang, N. Sundararajan, and P. Saratchandran an online sequential fuzzy extreme learning machine (OS-Fuzzy-ELM) has been developed for function ap- proximation and classification problems. This results in a FIS that can handle any bounded nonconstant piecewise continuous membership func- tion. Furthermore, the learning in OS-

Fuzzy-ELM can be done with the input data coming in a one-by-one mode or a chunk-by-chunk (a block of data) mode with fixed or varying chunk size.

ELM is a unified framework for generalized Single-hidden Layer Feedforward Networks (SLFNs):

$$f(x) = \sum_{i=1}^{N} \beta_i \; g_i\,(\mathrm{x}) = \sum_{i=1}^{N} \beta_i \,\mathrm{G}(\mathrm{w},\ \mathrm{x}, \mathrm{b}) = \sum_{i=1}^{N} \beta_i \,\mathrm{G}(w_i \cdot x_j + \beta_i) \quad ,$$

i = 1,2,3,…..,N.

where g_i or G(w, x, b) denotes the hidden node output function (with the hidden node parameters (w_i , b_i) and β_i is the weight of the connection between the *i*-th hidden node and the output node.
In ELM, all the hidden node parameters are totally randomly generated and completely independent from the training data.
The essence of ELM is that from a function approximation point of view the hidden nodes of ELMs are not much relevant to the target functions or the training data. All the hidden node parameters (a_i and b_i) of ELM could randomly be generated according to any given continuous probability distribution without any prior knowledge (even before the data are presented and the ELM learning starts). All the hidden node parameters (a_i and b_i) of ELM are not only independent from each other but also independent from the training data, which makes ELM learning scheme more efficiently and much simpler.

In this paper first it is rigorously proves that the input weights and hidden layer biases of as named in Huang et al. [11,10], H is called the hidden layer output matrix of the neural network; the ith column of H is the ith hidden node output with respect to inputs x_1 ; x_2 ; … ; x_n .

If the activation function g is infinitely differentiable one can prove that the required number of hidden nodes Ň < N . Strictly speaking,
we have 2 Theorem 2.1. Given a standard SLFN with N hidden nodes and activation function g : R $\Longrightarrow$ R, which is infinitely differentiable in any interval, for N arbitrary distinct samples { x_i , t_i }, for any w_i and b_i randomly chosen from any intervals of Rn and R, respectively, according to any continuous probability distribution, then with probability one, the hidden layer output matrix H of the SLFN is invertible distribution and $|| H\beta - T || = 0$, then with probability one, the column vectors of H can be made full-rank and Such activation functions include the sigmoidal functions as well as the radial basis, sine, cosine, exponential, and many other nonregular functions as shown in Huang and Babri [11].

There are several issues on BP learning algorithms:

(1) When the learning rate Z is too small, the learning algorithm converges very slowly. However, when Z is too large, the algorithm becomes unstable and diverges.

(2) Another peculiarity of the error surface that impacts the performance of the BP learning algorithm is the presence of local minima [6]. It is undesirable that the learning algorithm stops at a local minima if it is located far above a global minima.

(3) Neural network may be over-trained by using BP algorithms and obtain worse generalization performance. Thus, validation and suitable stopping methods are required in the cost function minimization procedure.

(4) Gradient-based learning is very time-consuming in most applications.

The aim of above mentioned paper is to resolve the above issues related with gradient-based algorithms and propose an efficient learning algorithm for feedforward neural networks.

3.2. **Proposed minimum norm least-squares (LS) solution of SLFNs**

As rigorously proved in Theorems, unlike the traditional function approximation theories which require to adjust input weights and hidden layer biases, input weights and hidden layer biases can be randomly assigned if only the activation function is infinitely differentiable. It is very interesting and surprising that unlike the most common understanding that all the parameters of SLFNs need to be adjusted, the input weights w_i and the hidden layer biases b_i are in fact not necessarily tuned and the hidden layer output matrix H can actually remain un- changed once random values have been assigned to these parameters in the beginning of learning. For fixed input weights w_i and the hidden layer biases b_i , seen from Equation, to train an SLFN is simply equivalent to finding a least- squares solution β of the linear system

$$\mathbf{H\beta = T}$$

$$||\mathrm{H}(w_1, \ldots w_N, b_1, \ldots, b_N)\beta - \mathrm{T}|| = \min_{wi,bi,\beta} ||\mathrm{H}(w_1 \ldots, w_N, b_1, \ldots b_N)\beta - \mathrm{T}||$$

If the number Ň of hidden nodes is equal to the number N of distinct training samples, Ň = N , matrix H is square and invertible when the input weight vectors w_i and the hidden biases b_i are randomly chosen, and SLFNs can approximate these training samples with zero error.
However, in most cases the number of hidden nodes is much less than the number of distinct training samples, Ň << N ,

H is a nonsquare matrix and there may not exist w_i ; b_i ; β_i (i = 1, 2, 3,... , N~) such that **Hβ = T**. According to Theorem , the smallest norm least-squares solution of the above linear system is

$$\beta = H^{\dagger} T$$

where $H^{\dagger}$ is the Moore–Penrose generalized inverse of matrix H [22,19].

The minimum norm least-squares solution of Hβ = T is unique,

which is $\beta = H^{\dagger} T$.

Proposed learning algorithm for SLFNs :

Thus, a simple learning method for SLFNs called extreme learning machine (ELM) can be summarized as follows:

1) ALGORITHM ELM: GIVEN CERTAIN MEMBER- SHIP FUNCTION G AND RULE NUMBER L FOR A SPECIFIC APPLICATION, THE DATA

$$\mathbb{N} = \{(XI , TI)|XI \;\square RN , \; TI \;\square\; RM ,I = 1,\ldots .\}.$$

STEP 1: RANDOMLY ASSIGN INPUT WEIGHT WI AND BIAS BI ,

I = 1, … , Ň.

STEP 2: CALCULATE THE HIDDEN LAYER OUTPUT MATRIX H.

STEP 3: CALCULATE THE OUTPUT WEIGHT β

$$\beta = \hat{H}^{\dagger} T$$

WHERE $\boldsymbol{T} = [\boldsymbol{T_1}, \boldsymbol{T_2}, \boldsymbol{T_3}, \ldots \ldots, \boldsymbol{T_N}]^T$ I.E. TRANSPOSE OF $\boldsymbol{T}$ VECTOR .

Section 5.1 : **SRAN Learning Algorithm** :

The SRAN classifier uses radial function network as a basic building block. The control parameters in the proposed sequential algorithm are self-regulated, so, they are fixed, and are mostly independent of the problem considered. The control parameters alter the sequence in which the SRAN classifier approximates the decision function, based on the difference between the information contained in each sample and the knowledge acquired by the network. The higher the difference, the earlier a sample participates in learning. A few samples with lesser differences are pushed to the rear end of the sample data stack. These samples are later used to fine-tune the network parameters. Also, a few samples with redundant information are discarded from the training data set, thus avoiding over-training. Thus, the finally realized network is compact and provides better generalization performance.

In the setting of standard online/sequential learning, the training sample arrives one at a time and the network adapts its parameters based on the diff erence in knowledge between the network and the current sample. Figure 1 gives a bird's eye view of the SRAN algorithm. As each new sample (xt) is presented to the network, based on the sample error (e), the sample is either

• used for network training (growing/learning) immediately, or

• pushed to the rear end of the stack for learning in future, or

• deleted from the data set.

In ideal conditions, training stops when there are no more samples to be presented to the network. However, in real-time, training is stopped when the samples get stacked repeatedly and do not participate in learning.
Similar to other sequential learning algorithms, the SRAN learning algorithm begins with zero hidden neurons and adds new hidden neuron based on the information present in the current sample.
In standard online/sequential learning, the samples are presented only once, and all the samples are learnt. In such a network, there is no control over the sample sequence which in uences the learning ability of the network. This also means that arrival of similar samples leads to over-training of particular pattern. This will, therefore, influence the generalization ability of the sequential learning algorithm. In SRAN approach, the sequence of the training sample is controlled internally using self-regulated control parameters as explained below.

Section 6.0 : **Modified Online Sequential Fuzzy Extreme Learning Machine for Function Approximation and Classification Problem**

I. Introduction : In OS ELM, an online sequential fuzzy extreme learning machine (OS-Fuzzy-ELM) has been developed for function approximation and classification problems. There equivalence of a Takagi– Sugeno–Kang (TSK) fuzzy inference system (FIS) to a generalized single hidden-layer feedforward network is shown first, which is then used to develop the OS-Fuzzy-ELM algorithm. This results in a FIS that can handle any bounded nonconstant piecewise continuous membership function. Furthermore, the learning in OS-Fuzzy-ELM can be done with the input data coming in a one-by-one mode or a chunk-by-chunk (a block of data) mode with fixed or varying chunk size. In OS-Fuzzy-ELM, all the antecedent parameters of membership functions are randomly assigned first, and then, the corresponding consequent parameters are determined analytically. Performance comparisons of OS-Fuzzy-ELM with other existing algorithms are presented using real-world benchmark problems in the areas of nonlinear system identification, regression, and classification. The results show that the proposed OS-Fuzzy-ELM produces similar or better accuracies with at least an order-of-magnitude reduction in the training time.

Fuzzy inference Systems (FISs) have been increasingly used in the areas of function approximation and classification problems [1]. In a FIS, the methods used to update the parameters of membership functions can be broadly divided into batch learning schemes and sequential learning schemes. In batch learning, it is assumed that the complete training data are available before the training commences. The training usually involves cycling the data over a number of epochs. In sequential learning, the data arrive one by one or chunk by chunk, and the data will be discarded after the learning of the data is completed; the notion of epoch does not exist. In practical applications, new training data may arrive sequentially. In order to handle this using batch learning algorithms, one has to retrain the network all over again, resulting in a large training time. Hence, in these cases, sequential learning algorithms are generally preferred over batch learning algorithms, as they do not require retraining whenever new data are received. An example of batch learning FIS is the adaptive-network-based fuzzy inference system (ANFIS) of Jang [2] and Chiu [3]. These learning algorithms require cycling the whole training data over a number of learning cycles (epochs).

Liang et al. [8] have recently developed an online sequential learning version of the batch extreme learning machine (ELM) [9]–[12] called (OS-ELM). In OS-ELM with additive nodes, the input weights (of the connections linking the input nodes to the hidden nodes) and hidden-node biases are randomly generated. Similarly, in OS-ELM with radial basis function (RBF) nodes, the centers and widths of the nodes are randomly generated and fixed. Based on this, the output weights are analytically determined. In both ELM and OS-ELM, all the hidden-node parameters are independent not only of the training data but also of each other. OS-ELM can learn the training data not only one by one but also chunk by chunk (with fixed or varying length) and discard the data for which the training has already been done.

Combining the advantages of the neural network (learning) and the fuzzy inference system (approximate reasoning), one can develop a neuro-fuzzy system which exhibits the characteristics of both. Many researchers have developed such a neuro-fuzzy system for solving real-world problems effectively. The functional equivalence between a Gaussian RBF neural network and a FIS with Gaussian membership functions has been shown under some mild conditions by Jang and Sun [13]. Expanding this concept further, in this correspondence, huang shows the functional equivalence between a generalized single hidden- layer feedforward network (SLFN) and a FIS with any membership function. Thus, the FIS with Gaussian membership functions becomes a special case of SLFNs. Using such a functional equivalence together with the recently developed OS-ELM, in this correspondence, huang had developed an online sequential fuzzy ELM (OS-Fuzzy-ELM) that can handle any bounded nonconstant piecewise continuous membership function.

This correspondence extends the OS-ELM developed in the domain of neural networks to FISs. In the proposed OS-Fuzzy-ELM, all the parameters of membership functions are randomly generated independent of the training data, and then, the consequent parameters are analytically determined. This significantly increases the learning speed by avoiding the iterative learning steps (and, perhaps, the human intervention) in traditional FISs (and neural networks). There, they present a comprehensive evaluation of OS-Fuzzy-ELM by comparing it with other well-known learning algorithms such as ANFIS, DENFIS, SAFIS, eTS, and Simpl_eTS. We have also extended the algorithm to a chunk-by-chunk mode. Study results based on the benchmark problems from the function approximation and classification areas indicate that the proposed OS- Fuzzy-ELM produces similar or better generalization performance with at least an order-of-magnitude reduction in the training time.

Recently, Sun [14] have proposed an ELM-based fuzzy inference system called E-TSK using K -means clustering to preprocess the data. ELM is used to obtain the membership of each fuzzy rule, and the consequent part is then determined using multiple ELMs. It should be noted that E-TSK is a batch processing method, as it requires all the training data to be available a priori.

II. Initial OS-FUZZY -ELM

We know that FIS is equivalent to a generalized SLFN (in a generalized formula), as presented in [12], where G(·) represents the output function of the hidden node and β represents the output weight vector. The output functions for the hidden nodes in the SLFN are based on the membership functions of FIS. According to Huang and Chen [12], it is reasonable to infer that the SLFNs with activation function G(·) [cf. (3)] could approximate any continuous target function as long as the parameters of the membership function g are randomly generated and the membership function g is bounded, nonconstant, and piecewise continuous. For brevity, the detailed proof of the universal approximation capability of such an equivalent SLFN for the TSK models is not addressed in this correspondence.

Modified OS-Fuzzy-ELM Algorithm

Since FIS is equivalent to an SLFN, ELM can be directly applied to a FIS. In such a scheme, the parameters of membership functions (c and a) are randomly generated, and based on this, the consequent parameters (β) are analytically determined. In many real applications, the training data arrive chunk by chunk or one by one (a special case of chunk), and hence, a FIS algorithm which can handle such sequential applications is a necessity.

According to the functional equivalence between SLFNs and FISs, the online sequential ELM (OS-ELM) for SLFNs with additive or RBF hidden nodes [8] can be linearly extended to FISs, and the resulted algorithm is called OS-Fuzzy-ELM. All the parameters of the membership function c_1, c_2 , ,...., c_L, a_1 , ,......, a_L will be randomly generated in OS-Fuzzy-ELM.

Expanding this concept further, in this correspondence, we modify the OS ELM algorithm in terms of checking error of incoming data sample in order to avoid mimic's and overtraining and it is observed that it helps to improve efficiency of OS ELM and its performance evaluation using Data Sets of VC, GI and IS from UCI.

We have the following: Initial OS-Fuzzy-ELM Algorithm :

Given certain membership function g and rule number L for a specific application, the data

ℕ = {(x_i, t_i) | x_i □ R_n , t_i□ R_m, i = 1,.. . n} arrive sequentially.

STEP 1) INITIALIZATION PHASE: INITIALIZE THE LEARNING USING A SMALL CHUNK OF INITIAL TRAINING DATA ℕ0 = {(XI , TI) , WHERE I =0 TO N0 FROM THE GIVEN TRAINING SET

ℕ = {(XI , TI) | XI □ RN , TI □ RM , I = 1,.. .}.

A) ASSIGN RANDOM MEMBERSHIP FUNCTION PARAMETERS (CI , AI), I = 1,..., L.

B) CALCULATE THE INITIAL MATRIX H0 FOR THE TSK MODELS
H0 = H(C1 ,..., CL , A1 ,..., AL ; X1 ,..., XN0)
WHERE H IS HIDDEN MATRIX.

C) ESTIMATE THE INITIAL PARAMETER MATRIX Q(0) = P0$[H\ 0]^T$T0 ,
WHERE P0 =$([H]^T H)^{-1}$ AND T0 = $(t1 , . . . , tN)^T$

SUMMARIZED WORKING OF MODIFIED OS-ELM ALGORITHM IS AS FOLLOWS
STEP 1 : INITIALIZATION PHASE

1) LEARNING IS INITIALIZED USING A SMALL GROUP OF TRAINING DATA.
2) ASSIGNGN RANDOM MEMBERSHIP FUNCTION PARAMETERS.
3) CALCULATE INITIAL MATRIX H0 FOR TSK MODEL.
4) ESTIMATE INITIAL PARAMETER MATRIX Q0.

STEP 2 : SEQUENTIAL LEARNING PHASE

1) PRESENT NEXT GROUP OF DATA TO NETWORK
2) CALCULATE INITIAL MATRIX $\boldsymbol{H_{K+1}}$ FOR TSK MODEL
2.1) CALCULATE PARAMETER MATRIX $\boldsymbol{Q_{K+1}}$.
3) CALCULATE P AND Q USING LEAST MEAN SQUARE.
4) INTRODUCE HINGE LOSS FUNCTION.
5) CHECK IF PREDICTED VALUE IS SAME AND ERROR IS LESS THEN 0.15 (I.E. EFFICIENCY IS MORE THAN 85%) THEN GO TO STEP 2 TO AVOID MIMIC'S (VIA OVER TRAINING).
6) SET K = K+1 AND GO TO STEP 2 FOR NEXT GROUP OF DATA.
7) CALCULATE OVERALL EFFICIENCY AND COMPARE RESULTS WITH ORIGINAL OS-ELMRECENTLY DOVELOPED SRAN AND SVM ALGORITHMS.

Section 7 : Summary of the simulation results using MATLAB.

Simulation Results

Results with different number of Training Samples & CHUNK MODE (withought modification VC,GI and IS)

data Set	Training Samples	Training Time(s)	tea	teo	tra	tro	MF	Rules
IS	70	0.94	64.62%	64.62%	100%	100%	gaussmf	4
IS	140	1.87	77.76%	77.76%	100%	100%	gaussmf	4
IS	240	2.57	88.33%	**88.33%**	98.10%	98.10%	gaussmf	4
Vehicle	100	0.89	55.80%	55.45%	100%	100%	gaussmf	6
Vehicle	300	2.78	69.43%	68.48%	99.67%	99.67%	gaussmf	6
Vehicle	424	5.60	78.82%	78.44%	96.33%	96.29%	gaussmf	6
Glass	109	0.66	77.65%	76.92%	0.97902	0.985075	gaussmf	8

data Set	**Training Samples**	**Training Time(s)**	**tea**	**teo**	**tra**	**tro**	**MF**	**Rules**
IS	70	0.98	75.86%	75.86%	100%	100%	Triangmf	5
IS	140	1.61	68.57%	68.57%	100%	100%	Triangmf	5
IS	240	3.04	84.10%	84.10%	99.05%	99.05%	Triangmf	5
Vehicle	100	0.81	48.14%	48.34%	100%	100%	Triangmf	3
Vehicle	300	3.01	72.55%	71.33%	99.33%	99.33%	Triangmf	3
Vehicle	424	6.55	79.75%	79.62%	95.86%	95.81%	Triangmf	3
Glass	109	2.25	76.31%	70.19%	99.09%	99.10%	Triangmf	7

data Set	**Training Samples**	**Training Time(s)**	**tea**	**teo**	**tra**	**tro**	**MF**	**Rules**
IS	70	0.66	74.52%	74.52%	100%	100%	Cauchymf	3
IS	140	1.48	77.33%	77.33%	100%	100%	Cauchymf	3
IS	240	2.92	83.81%	83.81%	100%	100%	Cauchymf	3
Vehicle	100	0.95	57.27%	57.11%	100%	100%	Cauchymf	4
Vehicle	300	3.01	72.52%	72.04%	99.67%	99.67%	Cauchymf	4
Vehicle	424	6.58	82.79%	**82.46%**	97.09%	97.10%	Cauchymf	4
Glass	109	0.55	7741.00%	**78.85%**	98.53%	99.10%	Cauchymf	12

Results without error percentage excluding Hinge Loss for VC,GI and IS

Error	data Set	Training Time(s)	tea(%)	teo(%)	tra(%)	tro(%)	Rules	MF
N/A	**IS**	2.91	83.80%	83.80%	100%	100%	3	Cauchymf
N/A	**IS**	3.04	84.09%	84.09%	99.04%	99.04%	5	Triangmf
N/A	**IS**	2.57	88.33%	88.33%	98.09%	98.09%	4	gaussmf
N/A	**Vehicle**	6.58	82.79.3%	82.46%	97.08%	97.09%	4	Cauchymf
N/A	**Vehicle**	6.55	79.74%	79.62%	95.86%	95.81%	3	Triangmf
N/A	**Vehicle**	5.6	78.81%	78.43%	96.32%	96.29%	6	gaussmf
N/A	**Glass**	0.54	77.41%	78.84%	98.53%	99.11%	12	Cauchymf
N/A	**Glass**	2.24	76.31%	70.19%	99.09%	99.11%	7	Triangmf
N/A	**Glass**	0.65	77.65%	76.92%	97.90%	98.51%	8	gaussmf

Results with Different error percentage including Hinge Loss for VC,GI and IS

Error	data Set	Training Time(s)	tea(%)	teo(%)	tra(%)	tro(%)	Rules	MF
10	**IS**	0.45	86.14%	86.14%	92.38%	92.38%	3	Cauchymf
10	**IS**	0.49	88.52%	88.52%	95.24%	95.24%	5	Triangmf
10	**IS**	0.24	88.90%	**88.90%**	94.29%	94.29%	4	gaussmf
10	**Vehicle**	0.07	55.81%	54.98%	95.00%	95.00%	4	Cauchymf
10	**Vehicle**	0.32	69.07%	**69.67%**	94.00%	94.00%	3	Triangmf
10	**Vehicle**	0.4	55.62%	55.45%	97.00%	97.00%	6	gaussmf
10	**Glass**	0.29	80.78%	75.24%	99.15%	99.10%	12	Cauchymf
10	**Glass**	0.23	83.14%	**76.20%**	96.39%	93.73%	7	Triangmf
10	**Glass**	0.42	80.12%	76.19%	96.75%	96.72%	8	gaussmf

Error	data Set	Training Time(s)	tea(%)	teo(%)	tra(%)	tro(%)	Rules	MF
15	**IS**	0.24	90.52%	90.52%	93.81%	93.81%	3	Cauchymf
15	**IS**	0.23	90.81%	90.82%	95.71%	95.14%	5	Triangmf
15	**IS**	0.16	90.33%	90.33%	94.29%	94.29%	4	gaussmf
15	**Vehicle**	0.59	81.14%	81.04%	92.36%	92.42%	4	Cauchymf
15	**Vehicle**	0.47	80.13%	80.09%	92.24%	92.23%	3	Triangmf
15	**Vehicle**	1.40	79.66%	79.38%	91.44%	91.48%	6	gaussmf
15	**Glass**	0.45	78.26%	75.24%	98.26%	98.21%	12	Cauchymf
15	**Glass**	0.58	77.62%	71.43%	99.13%	99.10%	7	Triangmf
15	**Glass**	0.56	78.23%	77.14%	98.49%	98.51%	8	gaussmf

Error	data Set	Training Time(s)	tea(%)	teo(%)	tra(%)	tro(%)	Rules	MF
20	**IS**	0.09	88.71%	88.71%	92.86%	92.86%	3	Cauchymf
20	**IS**	0.45	91.24%	91.24%	97.62%	97.62%	5	Triangmf
20	**IS**	0.17	89.14%	89.14%	95.24%	95.24%	4	gaussmf
20	**Vehicle**	0.08	58.76%	58.29%	98.00%	98.00%	4	Cauchymf
20	**Vehicle**	0.08	70.68%	70.85%	95.00%	95.00%	3	Triangmf
20	**Vehicle**	0.20	50.43%	50.00%	99.00%	99.00%	6	gaussmf
20	**Glass**	0.95	81.66%	75.24%	98.56%	98.51%	12	Cauchymf
20	**Glass**	0.20	72.55%	71.43%	98.49%	97.76%	7	Triangmf
20	**Glass**	0.30	81.75%	80.01%	93.42%	93.43%	8	gaussmf

Error	data Set	Training Time(s)	tea(%)	teo(%)	tra(%)	tro(%)	Rules	MF
50	**IS**	0.46	89.57%	89.57%	94.76%	94.76%	3	Cauchymf
50	**IS**	0.40	86.76%	86.76%	95.71%	95.71%	5	Triangmf
50	**IS**	0.39	90.42%	90.42%	95.71%	95.71%	4	gaussmf
50	**Vehicle**	0.06	56.56%	56.16%	97.21%	97.10%	4	Cauchymf
50	**Vehicle**	0.09	66.37%	65.87%	96.21%	96.12%	3	Triangmf
50	**Vehicle**	0.18	64.99%	64.73%	96.21%	96.01%	6	gaussmf
50	**Glass**	0.82	77.24%	76.19%	97.31%	97.31%	12	Cauchymf
50	**Glass**	0.15	77.86%	75.23%	95.53%	95.52%	7	Triangmf
50	**Glass**	0.29	77.96%	72.38%	94.39%	94.32%	8	gaussmf

Results for Binary Data Sets for BC, Liver, ION and PIMA

data Set	Training Time(s)	tea	teo	tra	tro	MF	Rules
BC	0.16	95.10%	95.30%	95%	96%	Cauchymf	3
BC	0.41	95.29%	95.30%	98%	98%	Triangmf	3
BC	0.22	95.69%	95.56%	98.57%	98.67%	gaussmf	3

Liver	0.06	70.39%	72.41%	75%	77%	Cauchymf	3
Liver	0.08	70.22%	70.34%	70.89%	72.50%	Triangmf	3
Liver	0.08	68.59%	70.34%	73.56%	75.00%	gaussmf	3
ION	0.27	64.57%	63.35%	0.93316	0.93	Cauchymf	3
ION	0.12	61.35%	60.16%	91.58%	90.00%	Triangmf	3
ION	0.09	66.58%	62.15%	88.98%	89.00%	gaussmf	3

PIMA	0.12	75.69%	78.26%	75.64%	78.00%	Cauchymf	3
PIMA	0.14	72.15%	76.09%	75.96%	78.25%	Triangmf	3
PIMA	0.12	76.23%	79.35%	74.52%	77.50%	gaussmf	3

Section 7.1 : **Conclusion:**

Performance of modified OS ELM is compared with SVM and recently developed SRAN and it is observed that modification done on OS ELM algorithm in terms of checking error of incoming data sample in order to avoid mimic's and overtraining helps to improve efficiency of OS ELM.

Section 8 : **possible ways to improve further this work within the area and any other methods which could be investigated**.

1)Changing parameter of algorithm like number of rules can improve further the speed of learning and hence working of OS ELM can be improved.
2) Changing crieteria to find mininmum error also be tried to find improvent in efficiency of algorithm.

REFERENCES

[1] J.-S. R. Jang, C.-T. Sun, and E. Mizutani, Neuro-Fuzzy and Soft Comput- ing: A Computational Approach to Learning and Machine Intelligence. Englewood Cliffs, NJ: Prentice–Hall, 1997.
[2] J.-S. R. Jang, "ANFIS: Adaptive-network-based fuzzy inference system,"
IEEE Trans. Syst., Man, Cybern., vol. 23, no. 3, pp. 665–685, May/Jun. 1993. [3] S. L. Chiu, "Selecting input variables for fuzzy models," J. Intell. Fuzzy
Syst., vol. 4, pp. 243–256, 1996.
[4] N. K. Kasabov and Q. Song, "DENFIS: Dynamic evolving neural-fuzzy inference system and its application for time-series prediction," IEEE Trans. Fuzzy Syst., vol. 10, no. 2, pp. 144–154, Apr. 2002.
[5] P. P. Angelov and D. P. Filev, "An approach to online identification of Takagi–Sugeno fuzzy models," IEEE Trans. Syst., Man, Cybern. B,
Cybern., vol. 34, no. 1, pp. 484–498, Feb. 2004.

[6] P. Angelov and D. Filev, “Simpl_eTS: A simplified method for learning evolving Takagi–Sugeno fuzzy models,” in Proc. 14th IEEE Int. Conf. Fuzzy Syst., 2005, pp. 1068–1073.

[7] H.-J. Rong, N. Sundararajan, G.-B. Huang, and P. Saratchandran, “Sequential Adaptive Fuzzy Inference System (SAFIS) for nonlinear system identification and prediction,” Fuzzy Sets Syst., vol. 157, no. 9, pp. 1260–1275, May 2006.

[8] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, “A fast and accurate online sequential learning algorithm for feedforward networks,” IEEE Trans. Neural Netw., vol. 17, no. 6, pp. 1411–1423, Nov. 2006.

[9] G.-B. Huang, Q.-Y. Zhu, K. Z. Mao, C.-K. Siew, P. Saratchandran, and N. Sundararajan, “Can threshold networks be trained directly?” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 53, no. 3, pp. 187–191, Mar. 2006.

[10] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine:
Theory and applications,” Neurocomputing, vol. 70, no. 1–3, pp. 489–501, Dec. 2006.

[11] G.-B. Huang, L. Chen, and C.-K. Siew, “Universal approximation us- ing incremental constructive feedforward networks with random hidden nodes,” IEEE Trans. Neural Netw., vol. 17, no. 4, pp. 879–892, Jul. 2006.

[12] G.-B. Huang and L. Chen, “Convex incremental extreme learning ma- chine,” Neurocomputing, vol. 70, no. 16–18, pp. 3056–3062, Oct. 2007.

[13] J.-S. R. Jang and C.-T. Sun, “Functional equivalence between radial basis function networks and fuzzy inference systems,” IEEE Trans. Neural
Netw., vol. 4, no. 1, pp. 156–159, Jan. 1993.

[14] Z.-L. Sun, K.-F. Au, and T.-M. Choi, “A neuro-fuzzy inference system through integration of fuzzy logic and extreme learning machines,” IEEE Trans. Syst., Man, Cybern. B, Cybern., vol. 37, no. 5, pp. 1321–1331, Oct. 2007.

[15] G.-B. Huang, N.-Y. Liang, H.-J. Rong, P. Saratchandran, and
N. Sundararajan, “On-line sequential extreme learning machine,” in Proc. IASTED Int. Conf. CI, Calgary, AB, Canada, Jul. 4–6, 2005.

[16] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications for modeling and control," IEEE Trans. Syst., Man, Cybern., vol. SMC-15, no. 1, pp. 116–132, Feb. 1985.
[17] H. Ying, "Sufficient conditions on uniform approximation of multivariate functions by general Takagi–Sugeno fuzzy systems with linear rule conse- quent," IEEE Trans. Syst., Man, Cybern. A, Syst., Humans, vol. 28, no. 4, pp. 515–520, Jul. 1998.
[18] X.-J. Zeng and M. G. Singh, "Approximation theory of fuzzy systems—MIMO case," IEEE Trans. Fuzzy Syst., vol. 3, no. 2, pp. 219–235, May 1995.
[19] P. Angelov, J. Victor, A. Dourado, and D. Filev, "On-line evolution of Takagi–Sugeno fuzzy models," in Proc. 2nd IFAC Workshop Adv. Fuzzy/Neural Control, Oulu, Finland, 2004, pp. 67–72.
[20] C. Blake and C. Merz, "UCI repository of machine learning databases," Dept. Inf. Comput. Sci., Univ. California, Irvine, CA, 1998. [Online].
Available: http://www.ics.uci.edu/~mlearn/MLRepository.html
[21] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," IEEE Trans. Neural Netw., vol. 1, no. 1, pp. 4–27, Mar. 1990.
[22] C.-T. Lin, C.-M. Yeh, S.-F. Liang, J.-F. Chung, and N. Kumar, "Support-vector-based fuzzy neural network for pattern classification," IEEE Trans. Fuzzy Syst., vol. 14, no. 1, pp. 31–41, Feb. 2006.
[23] Q.-L. Tran, K.-A. Toh, D. Srinivasan, K.-L. Wong, and S. Q.-C. Low, "An empirical comparison of nine pattern classifiers," IEEE Trans. Syst., Man, Cybern. B, Cybern., vol. 35, no. 5, pp. 1079–1091, Oct. 2005.
[24] K.-A. Toh, Q.-L. Tran, and D. Srinivasan, "Benchmarking a reduced multivariate polynomial pattern classifier," IEEE Trans. Pattern Anal. Mach. Intell., vol. 26, no. 6, pp. 740–755, Jun. 2006.

[25] Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks

[26] S. Tamura, M. Tateishi, Capabilities of a four-layered feedforward neural network: four layers versus three, IEEE Trans. Neural Networks 8 (2) (1997) 251–255.

[27] G.-B. Huang, Learning capability and storage capacity of two-hidden-layer feedforward networks, IEEE Trans. Neural Networks
[28] Online Sequential Fuzzy Extreme Learning Machine for
Function Approximation and Classification Problems