2.1. Introduction

Recent research in Adaptive Systems has resulted in a variety of adaptive automation. An adaptive automation is a system whose structure is alterable or adjustable in such a way that its behaviour and performance improves through contact with its environment. A simple example of an adaptive system is the automatic gain control used in radio and television receiver. The most important factor in adaptive system is its time-varying and self-adjusting performance. Their characteristic depends upon the input signal. If a signal is applied to the input of the adaptive system to test its response characteristic, the system adapts to this specific input and thereby changes its own form. The adaptation procedure is carried out using different algorithms such as the Least Mean Square (LMS), Recursive Least Square (RLS), etc. In many problems these algorithms do not work satisfactorily. Hence based on the neural architecture of humane brain the different Artificial Neural Algorithms are developed [20, 24, 61, 62]. These are capable of mapping the input and output non-linearly. But, they can not provide proper results when uncertainty in the input space arises. To overcome such problems the algorithms based on Fuzzy Logic based filtering techniques are developed [25, 72, 123, 124].

In this chapter different types of fuzzy and neural algorithms has been studied, which are used as tools to solve different problems dealt in the subsequent chapters. The present chapter is organised as follows: Section 2.2. deals with the adaptive filtering technique under which the LMS and RLS algorithms are explained in
different subsections. The Artificial Neural Network and the associated training algorithms are discussed in Section 2.3. and its subsections. The Fuzzy Adaptive Filters and the Least Mean Square Fuzzy Adaptive Filter has been explained in Section 2.4 and 2.4.1. respectively.

2.2. Adaptive Filtering Techniques

Filter is an important subsystem in any signal processing system. Filters are used to remove undesirable signal components from the desired signal. In Adaptive Filters the coefficients can be changed from time to time depending on the situation. Here the filter updates its coefficients from the knowledge of the past input and the present error. The error is generated from the reference input and actual output. The update procedure depends upon the different algorithms used.

2.2.1. Least Mean Square Algorithm (LMS)

LMS algorithm was developed by Widrow and Hoff [117]. In this algorithm the weights are updated according to estimate the mean square surface gradient. For the ith weight it is given by:

\[
    w_i(n) = w_i(n-1) + 2\lambda \Delta w_i e(n)x(n-1)
\]  

(2.1)

where \( \lambda \) is the convergence factor. A \( N \) point tapped delay filter based on LMS algorithm is shown in Fig. 2.1. Here \( x(n) \) is the input signal fed to the tap delay and the signal samples are multiplied with the filter coefficients \( w_0, w_1, ..., w_N \) where \( N \) is the filter order. \( y(n) \) is the estimated output where as \( d(n) \) is the desired signal and the difference between these two is the error signal \( e(n) \). Using this error and the estimated and desired signal the update eqn.(2.1) is evaluated to produce the minimum Mean Square Error (MSE).
2.2.2. Recursive Least Square Algorithm (RLS)

The RLS algorithm basically minimises the sum of squared errors up to last signal sample. The recursive weight update equation is given by:

$$w(n + 1) = w(n) + \Gamma_{xx}^{-1}(n + 1)x(n + 1)e(n + 1)$$

(2.2)

where,

$$\Gamma_{xx}^{-1}(n + 1) = \Gamma_{xx}^{-1}(n) - \frac{\Gamma_{xx}^{-1}(n)x^T(n + 1)\Gamma_{xx}^{-1}(n)}{1 + x^T(n + 1)\Gamma_{xx}^{-1}(n)x(n + 1)}$$

(2.3)

and,

$$\Gamma_{xx}(n) = \sum_{k=0}^{n}x(k)x^T(k)$$

(2.4)

Under low noise conditions the convergence is guaranteed within $2N$ iterations where $N$ is the filter order.
2.3. Artificial Neural Network (ANN)

Artificial neural network takes their name from the network of nerve cells in the brain. It provides an unique computing architecture, introduced by Von Neumann. Recently, ANN has been found to be an important technique for classification and optimisation problem [20, 24, 61, 62, 115]. There are extensive application of ANN in the field of channel equalisation [14, 33, 84, 97], estimation of parameters of non-linear systems [68], pattern recognition [28, 34, 35, 49, 65, 80, 83, 104, 121], etc. ANN is capable of performing non-linear mapping between the input and output space due to its large parallel interconnection between different layers and the non-linear processing characteristics. An artificial neuron basically consists of a computing element that performs the weighted sum of the input signal and the connecting weight. The sum is added with the bias or threshold and the resultant signal is then passed through a non-linear element of tanh(.) type. Each neuron is associated with three parameters that can be adjusted during learning: these are the connecting weights, the bias and the slope of the non-linear function. For the structural point of view a NN may be single layer or it may be multi-layer. In multi-layer structure, there are many artificial neurons in one layer and for a practical case there may be a number of layers. Each neuron of the one layer is connected to each and every neuron of the next layer. The Functional link ANN is another type of single layer NN. In this type of network the input data is allowed to pass through a functional expansion block where the input data are non-linearly mapped to more number of points. This is achieved by using trigonometric functions, product or power terms of the input. The output of the functional expansion is then passed through a single neuron.

The learning of the ANN may be supervised in the presence of the desired signal or it may be unsupervised when the desired signal is not accessible. Rumelhart developed the Back propagation algorithm [93], which is central to much work on supervised learning in multi-layer NN. A feed forward structure with input, output, hidden layers and non-linear sigmoid functions are used in this type of network. In recent years many different types of learning algorithm using the incremental
backpropagation algorithm [27], evolutionary learning using the nearest neighbor MLP [125] and a fast learning algorithm based on the layer-by-layer optimization procedure [115] are suggested in literature. In case of unsupervised learning the input vectors are classified into different clusters such that elements of a cluster are similar to each other in some sense. The method is called competitive learning [10], because during learning a set of hidden units compete with each other to become active and perform the weight change. The winning unit increases its weights on those links with high input values and decreases them on those with low input values. This process allows the winning unit to be selective to some input values. The details of different types of NNs and their learning algorithms are discussed below.

2.3.1. Supervised Learning

In supervised learning, for each input pattern the correct answer or the desired output is known. When one input is applied to the network, the difference between the output estimated by the network and the known desired output gives rise to the error signal which is then used to change the connecting weights to minimise the difference. This is done incrementally making small adjustments with respect to each input to minimise the error. After certain time, the error will attain a steady state value and will not decrease further. It is then said that the learning is over.

2.3.1.1 Single Neuron Structure

![Fig. 2.2. Structure of a Single Neuron](image)
The basic structure of an artificial neuron is presented in Fig. 2.2. The operation in an artificial neuron involves the computation of the weighted sum of inputs and threshold. The resultant signal is then passed through a non-linear activation function. The output of the neuron may be represented as,

\[ y(n) = g \left[ \sum_{i=1}^{N} w_i(n)x_i(n) + \theta(n) \right] \]  

(2.5)

where \( \theta(n) \) is the threshold to the neurons at the first layer, \( w_i(n) \) is the weight associated with the \( j^{th} \) input, \( N \) is the no. of inputs to the neuron and \( g(\cdot) \) is the non-linear activation function. Different types of non-linear function are shown in Fig. 2.3.

Fig. 2.3. Different Types of Non-Linear Activation Function, (a) Hard Limiter, (b) Threshold Logic, (c) Sigmoid and (d) Piecewise Linear

If the non-linear activation function is a sigmoid type function, then

\[ g(s) = \frac{1 - e^{-\phi s}}{1 + e^{-\phi s}} \]  

(2.6)

where \( s \) is the input to the sigmoid function and \( \phi \) is the slope of the sigmoid function. For the steady convergence a proper choice of \( \phi \) is required. The parameters such as the weights and the thresholds are chosen arbitrarily and are updated during the training procedure to minimise the difference between the desired and the estimated signal. The update equations are,
\[
\Delta w_j(n+1) = \lambda e(n) x_j(n) + \alpha \Delta w_j(n)
\]
for weights \((2.7)\)

\[
\Delta \theta(n+1) = \lambda e(n) + \alpha \Delta \theta(n)
\]
for thresholds \((2.8)\)

where

\[
e(n) = (d(n) - y(n)) \left(1 - y(n)^2\right)/2
\]

where \(d(n)\) is the desired output and \(\lambda\) is the positive step size.

### 2.3.1.2. Multi layer Structure

The scheme of multi-layer neural network using three layers and the learning mechanism is depicted in Fig.2.4.

![Structure of Multi-Layer Neural Network](image)

Fig. 2.4. Structure of Multi-Layer Neural Network

\(x_i(n)\) represent the input to the network, \(f_i^{(1)}\) and \(f_i^{(2)}\) represent the output of the input and hidden layers respectively and \(y(n)\) represents the output of the final layer of the neural network. The connecting weights between the input to the first layer, first to second layer and the second layer to the output layers are represented by \(w_{ij}^{(1)}, w_{jk}^{(2)},\) and \(w_{kl}^{(3)}\) respectively.
If $P_1$ is the number of neurone in the first layer, each element of the output vector may be calculated as

$$f_j^{(1)} = g_j \left[ \sum_{i=1}^{N} w_{ji}^{(1)} x_i(n) + \theta_j^{(1)} \right]$$

for $j = 1, 2, 3, \ldots, P_1$ (2.10)

where $\theta_j^{(1)}$ is the threshold to the neurons at the first layer, $N$ is the no. of inputs and $g(\cdot)$ is the non-linear activation function of eqn.(2.6). The time index $n$ has been dropped to make the equations more simpler. Let $P_2$ be the number of neurons in the second layer. The output of this layer is represented as

$$f^{(2)} = [f_1^{(2)}, \ldots, f_k^{(2)}, \ldots, f_{P_2}^{(2)}]$$

(2.11)

Each element of this output vector, $f_k^{(2)}$, may be written as

$$f_k^{(2)} = g_k \left[ \sum_{j=1}^{P_2} w_{jk}^{(2)} f_j^{(1)} + \theta_k^{(2)} \right]$$

(2.12)

where $\theta_k^{(2)}$ is the threshold to the neurons at the second layer. The output of the final layer can be calculated as

$$y(n) = g_1 \left[ \sum_{k=1}^{P_2} w_{k1}^{(3)} f_k^{(2)} + \theta_1^{(3)} \right]$$

(2.13)

where $\theta_1^{(3)}$ is the threshold to the neuron at the final layer. The output of the equaliser in terms of the network input and the connecting weights may be expressed as
Using the Back Propagation (BP) Neural Algorithm, a multi-layer neural network having 3 neurons in input layer, 2 neurons in the hidden layer and 1 neuron in the output layer is shown in Fig. 2.5., the parameters of the neural network are updated in a batching mode. In case of conventional BP algorithm, initially the weights and the thresholds are random values. With the help of eqn. 2.10., eqn. 2.12., and eqn. 2.13. the final and intermediate outputs are calculated. The final output is compared with the desired output and the resulting error signal is obtained. This error signal is used to update the weights and thresholds of the hidden layers as well as the output layer. The reflected error components at each of the hidden layers is computed using the errors of the last layer and the connecting weights between the hidden and the last layer and error obtained at this stage is used to update the weights between the input and the hidden layer. The thresholds are also updated in a similar manner as that of the corresponding connecting weights. The weights and the thresholds are updated in an iterative method until the difference between the desired and the output becomes minimum. For measuring the degree of matching, the Mean Square Error (MSE) is taken as a performance measurement.

\[ y(n) = g_{w} \left[ \sum_{k=1}^{N} w^{(2)}_{k} g_{l} \left( \sum_{j=1}^{N} w^{(2)}_{j} g_{n} \left( \sum_{i=1}^{N} w^{(1)}_{i} x_{i}(n) + \theta_{l}^{(1)} \right) + \theta_{j}^{(2)} \right) + \theta_{k}^{(3)} \right] \]

Learning Algorithm

**Fig. 2.5. Neural Network Using BP Algorithm**
The key equations describing the BP algorithm for multi-layered ANN can be written as,

\[ \Delta w_j^l(n+1) = \lambda e_j^l(n) x_j^{l-1}(n) + \alpha \Delta w_j^l(n) \]  

(2.15)

for weights

\[ \Delta \theta_j^l(n+1) = \lambda e_j^l(n) + \alpha \Delta \theta_j^l(n) \]  

(2.16)

for thresholds

where

\[ e_j^l(n) = \left( d(n) - y(n) \right) \left( 1 - y(n)^2 \right)/2 \]  

(2.17)

for output layer

\[ e_j^l(n) = \left[ 1 - (f_j^l)^2 \right] \left[ \sum_i e_i^{l+1}(n)w_{ik}^{l+1}(n) \right]/2 \]  

(2.18)

for hidden layers

where \( l \) is an index for different layers, \( d(n) \) is the desired output and \( \lambda \) is the positive step size and \( n \) is the index for all the neurons above \( j \) th layer.

\[ e_k^l = \left[ 1 - (f_k^l)^2 \right] \left[ e_1^{l+1}w_{k1}^{l+1} + e_2^{l+1}w_{k2}^{l+1} + e_3^{l+1}w_{k3}^{l+1} \right]/2 \]  

(2.19)

2.3.1.3. Functional Link Artificial Neural Network (FLANN)

The functional link in a (FLANN) of Pao – network originally proposed by Pao is a novel single layer ANN structure capable of forming arbitrarily complex decision regions by generating non-linear decision boundaries. Here, the initial representation of a pattern is enhanced by using non-linear function and thus the pattern dimension space is increased. The functional link acts on an element of a pattern or entire pattern itself by generating a set of linearly independent function and then evaluates these functions with the pattern as the argument. Hence separability of the patterns becomes possible in the enhanced space. The use of FLANN non only increases the
learning race but also has less computational complexity. Pao et. al [82] have investigated the learning and generalisation characteristics of a random vector FLANN and compared with those attainable with multilayer perceptor structure trained with back propagation algorithm by taking few functional approximation problems.

A FLANN structure is shown in Fig. 2.6. Let us consider a two-dimensional input pattern $X = [x_1, x_2]^T$. This pattern has been enhanced by functional expansion using trigonometric functions such as:

$$
\begin{bmatrix}
  x_1 \cos(x_1) & \sin(x_1) & \cdots & \cos(2\pi x_1) & \sin(2\pi x_1) & \cdots \\
  x_2 \cos(x_2) & \sin(x_2) & \cdots & \cos(2\pi x_2) & \sin(2\pi x_2) & \cdots \\
  x_1 & x_2 & \\
\end{bmatrix}^T.
$$

The weighted sum of the components of the enhanced input is passed through a hyperbolic tangent non-linear function to produce an output $y(n)$ by using eqn.(2.5) and eqn.(2.6). The error resulted out of comparison between output of the FLANN and desired response is used to update the weights of the FLANN by the BP algorithm stated in eqn.(2.7) to eqn.(2.9).
The non-linear mapping may be of trigonometric type as stated above or of power series type. The purpose of the FLANN structure is to introduce initial non-linearity into the input signal so that the overall non-linearity as demanded by the problem may be met with a fewer neurons and layers.

2.3.2. Unsupervised Learning

In case of unsupervised competitive learning only one output unit or only one per group wins at a time. The output units compete to win or to fire, hence this type of network is often called as winner-takes-all units. The purpose of such type of network is to categorise or cluster the input data. Similar inputs are classified as being in the same category and so fires the same output unit. Multi-layer back propagation algorithm is extremely slow because the optimum weight of one layer depends upon all the weights in all other layers. This can be avoided to some extent by the unsupervised learning approach.

In case of competitive learning only one neuron at the output is one and all others are zero. A three layered competitive learning network has been depicted in Fig.2.7.

![Fig.2.7. A Three Layer Competitive Learning Network](image)

In the above figure \( P_1, P_2 \) and \( P_3 \) represents the no. of neurons in the first layer, hidden layer and the output layer respectively and \( w_{ij} \) is the weight between the
input layer to the hidden layer and \( w_{jk} \) is the weight between the hidden layer to the output layer. The input vector is,

\[
X = \left( x_1, x_2, \ldots, x_n \right)
\]

(2.21)

The output vector of the first layer is,

\[
Z = \left( z_1, z_2, \ldots, z_{p_1} \right)
\]

The input to the hidden layer is,

\[
z_j = \sum_{i=1}^{p_1} w_{ji} x_i \]

(2.22)

where \( j = 1, 2, \ldots, p_2 \)

then competition takes place in the hidden layer. Competition assigns a single unit in the hidden layer to be the winner. The unit with highest sum is denoted as \( z_c \), where

\[
z_c = \max_{j=1}^{p_2} \left( z_j \right)
\]

After competition the hidden layer activations are,

\[
z_c = 1.0 \quad \text{for } j = c
\]

\[
z_j = 0.0 \quad \text{for } j \neq c
\]

(2.23)

The network output is,

\[
y_k = \sum_{j=1}^{p_2} w_{jk} z_j \quad \text{for } j = c
\]

\[
0 \quad \text{for } j \neq c
\]

(2.24)

where \( k = 1, 2, \ldots, P_3 \). Only one neuron in the output layer has the non-zero activation where as all others have zero activation.
2.4. Fuzzy Adaptive Filter (FAF)

Fuzzy logic introduced by Zadeh [123, 124] in 1965 and Dubois and Prade [25] in 1980, is logic of approximate mode of reasoning and is finding applications from process control to medical diagnosis. This logic has the ability to model the uncertain data encountered in our real life. Existing filters can process numerical data only and the existing expert system can process linguistic information in the form of IF-THEN rules. Hence, their applications are limited to problems where either the numerical or the linguistic data plays an important role. There are a large number of problems in Engineering, seismology, etc. where both numerical and linguistic information are critical [114, 113]. Such type of problems can not be solved using the existing technique. At present when we face such problems, we use the linguistic information, in the choice among different filters, the evaluation of filter performance, the choice of filter orders, the interpretation of filtering results, etc. The linguistic information can not be used in this way, because in many practical cases the linguistic information is not about which kind of filter should be used and what should be the filter order, but is in the form of IF-THEN rules concerning the fuzzy concepts. Wang [114] developed a fuzzy adaptive filter that can use both linguistic as well as numerical data in their natural form i.e. as fuzzy IF-THEN rules and input-output data pairs. The fuzzy logic involves more computation that can be reduced by Fuzzy NN. It is recently shown that Neuro fuzzy approach provides better approach in control problems and in communication. A recent work on fuzzy adaptive filter minimises some criterion function, as new information becomes available. The filter is constructed from a set of fuzzy IF-THEN rules using an adaptive algorithm.

A fuzzy adaptive filter performs a two stage operation on the input vector \( x \): first, it performs a non-linear transformation \( p(*) \) on \( x \); then the filter output is obtained as a linear combination of these transformed signals. In this sense, the fuzzy adaptive filter is similar to the radial basis function and potential function approach. The unique feature of the fuzzy adaptive filter is that the linguistic rules can be incorporated into the filter.
2.4.1. LMS Fuzzy Adaptive Filter (LMSFAF)

![Diagram of LMSFAF](image)

(A Denotes The Gaussian Membership Function)

Fig. 2.7. Structure of LMSFAF

The learning algorithm for LMSFAF is derived based on the four steps mentioned below.

**Step I.**

In Fig. 2.7, a set of $M$ fuzzy sets are constructed using the following Gaussian membership function,

$$
\mu_{F_i}^{m}(x) = \exp \left[ -(1/2) \left( \frac{x - c_i^m}{\sigma_i^m} \right)^2 \right]
$$

where $m = 1, 2, \ldots, M$, $i = 1, 2, \ldots, n$. $c_i^m$ and $\sigma_i^m$ are the free parameters which are to be updated during the learning procedure and $n$ is the time index.
Step II.

A set of $M$ fuzzy IF-THEN rules are calculated in the following method:

$$R^m: \text{IF } x_1 \text{ is } F_{i1}^m \text{ and } \cdots \text{ and } x_n \text{ is } F_{in}^m, \text{ THEN } d \text{ is } G^m$$

where $d$ is the desired output. If there are linguistic rules in the form of the above equation then $F_{im}$ and $G^m$ are set to be the level of these linguistic rules, otherwise $c^m_i$ and $\sigma_i^m$ parameters are chosen arbitrarily. The membership function of these rules are updated during the learning procedure.

Step III.

The output of the fuzzifier can be expressed as,

$$b^m(n-1) = \prod_{i=1}^{M} \exp \left[ \frac{1}{2} \left( \frac{x_i(n) - c_i^m(n-1)}{\sigma_i^m(n-1)} \right)^2 \right]$$

$$b(n-1) = \sum_{m=1}^{M} b^m(n-1)$$

At each time point $n = 1, 2, \ldots$, the input to the filter is given by,

$$a^m(n-1) = \frac{b^m(n-1)}{b(n-1)}$$

The filter output is

$$y(n) = \sum_{m=1}^{M} w^m(n)a^m(n)$$
the error signal is

\[ e(n) = d(n) - y(n) \]  \hspace{1cm} (2.34)

**Step IV.**

The weight update eqn. is

\[ w''(n) = w''(n-1) + \lambda e(n-1)a''(n-1) \]  \hspace{1cm} (2.35)

The free parameters used in the Gaussian membership function can be updated using the following equations;

\[ \Delta c_i^{m}(n) = \lambda e(n) q(n) \left[ a^{m}(n-1) \right] \left[ w^{m}(n-1) - y(n) \right] \]  \hspace{1cm} (2.36)

\[ \Delta \sigma_i^{m}(n) = \lambda e(n) q(n) \left[ \frac{x_i(n) - c_i^{m}(n-1)}{\sigma_i^{m}(n-1)} \right] \left[ a^{m}(n-1) \right] \left[ w^{m}(n-1) - y(n) \right] \]  \hspace{1cm} (2.37)

where \[ q(n) = \frac{(x_i(n) - c_i^{m}(n-1))}{\left[ \sigma_i^{m}(n-1) \right]^2} \] , and \( \lambda \) is the small positive step size.