TIME SERIES PREDICTION AND CHANNEL EQUALIZER USING ARTIFICIAL NEURAL NETWORKS WITH VLSI IMPLEMENTATION

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

in

VLSI DESIGN and EMBEDDED SYSTEM

By
JIJU MV
Roll No:20607001

Department of Electronics and Communication Engineering
National Institute Of Technology
Rourkela
2008

TIME SERIES PREDICTION AND CHANNEL EQUALIZATION USING ARTIFICIAL NEURAL NETWORKS WITH VLSI IMPLEMENTATION

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

in

VLSI DESIGN and EMBEDDED SYSTEM

By
JIJU MV
Roll No:20607001

Under the Guidance of
Prof. G. Panda

Department of Electronics and Communication Engineering
National Institute Of Technology
Rourkela
2008

National Institute Of Technology
Rourkela

CERTIFICATE

This is to certify that the thesis entitled, **"Time series Prediction and channel Equalization Using Artificial Neural Networks with VLSI implementation"** submitted by Sri **JIJU MV** in partial fulfillment of the requirements for the award of Master of Technology Degree in **Electronics & communication Engineering** with specialization in **"VLSI DESIGN & EMBEDDED SYSTEM "** at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University / Institute for the award of any Degree or Diploma.

Prof. G. Panda (*FNAE , FNASc*)
Dept. of Electronics & Communication Engg.
National Institute of Technology
Rourkela-769008

Date:

ACKNOWLEDGEMENTS

This project is by far the most significant accomplishment in my life and it would be impossible without people who supported me and believed in me.

I would like to extend my gratitude and my sincere thanks to my honorable, esteemed supervisor **Prof. G. Panda,** Head, Department of Electronics and Communication Engineering. He is not only a great lecturer with deep vision but also and most importantly a kind person. I sincerely thank for his exemplary guidance and encouragement. His trust and support inspired me in the most important moments of making right decisions and I am glad to work with him.

I want to thank all my teachers **Prof. K. K. Mahapatra, Prof. G.S. Rath**, **Prof. S.K. Patra** and **Prof. S.Meher** for providing a solid background for my studies and research thereafter. They have been great sources of inspiration to me and I thank them from the bottom of my heart.

I would like to thank all my friends and especially my classmates for all the thoughtful and mind stimulating discussions we had, which prompted us to think beyond the obvious. I've enjoyed their companionship so much during my stay at NIT, Rourkela.

I would like to thank all those who made my stay in Rourkela an unforgettable and rewarding experience.

Last but not least I would like to thank my parents, who taught me the value of hard work by their own example. They rendered me enormous support during the whole tenure of my stay in NIT Rourkela.

JIJU MV

Roll No 20607001

CONTENTS

Abstract

The architecture and training procedure of a novel recurrent neural network (RNN), referred to as the multifeedbacklayer neural network (MFLNN), is described in this paper. The main difference of the proposed network compared to the available RNNs is that the temporal relations are provided by means of neurons arranged in three feedback layers, not by simple feedback elements, in order to enrich the representation capabilities of the recurrent networks. The feedback layers provide local and global recurrences via nonlinear processing elements. In these feedback layers, weighted sums of the delayed outputs of the hidden and of the output layers are passed through certain activation functions and applied to the feedforward neurons via adjustable weights. Both online and offline training procedures based on the backpropagation through time (BPTT) algorithm are developed. The adjoint model of the MFLNN is built to compute the derivatives with respect to the MFLNN weights which are then used in the training procedures. The Levenberg–Marquardt (LM) method with a trust region approach is used to update the MFLNN weights. The performance of the MFLNN is demonstrated by applying to several illustrative temporal problems including chaotic time series prediction and nonlinear dynamic system identification, and it performed better than several networks available in the literature.

List Of Figures

List Of Tables

Table No. **Table Title** **Page No.**

Abbreviations Used

RNN	Recurrent Neural Network
BP	Back propagation
LMS	Least Mean Square
RLS	Recursive Least Square
ANN	Artificial Neural Network
MFLNN	Multifeedback-layer Neural Network
LM	Levenberg-Marquardt
BPTT	Back Propagation Through Time
MLP	Multi layer perceptron
FLANN	Functional Link Artificial Neural Network
LIN	Linear least square-based equalizer
FIR	Finite Impulse Response
MSE	Mean Square Error
BER	Bit Error Rate
FPGA	Field programmable Gate Array
HDL	Hardardware Description Language

Chapter 1

INTRODUCTION

In recent years, with the growth of internet technologies, high speed and efficient data transmission over communication channels has gained significant importance. The rapidly increasing computer communication has necessitated higher speed data transmission over wide spread network of voice bandwidth channels. In digital communications the symbols are sent through linearly dispersive mediums such as telephone, cable and wireless. In band width efficient data transmission systems, the effect of each symbol transmitted over such time-dispersive channel extends to the neighboring symbol intervals. This distortion caused by the resulting overlap of received data is called inter symbol interference (ISI) [4.5].

The pursuit to build intelligent human like machines led to the birth of artificial neural network (ANN).Much work based on computer simulations has proved capability of ANNs to map, model, and classify nonlinear systems. The special features of ANNs such as capability to learn from examples , adaptation, parallelism, robustness to noise, and fault tolerance have opened their application fields of engineering, science economics. Real time application are feasible only if low cost high speed neural computation is made viable.

1.1 Motivation

The majority of physical systems contain complex nonlinear relations, which are difficult to model with conventional techniques. Neural networks (NNs) have learning, adaptation, and powerful nonlinear mapping capabilities. Therefore, they have been studied to deal with predicting, modeling, and control of complex, nonlinear, and uncertain systems, in which the conventional methods fail to give satisfactory results Recurrent neural networks (RNNs) naturally involve dynamic elements in the form of feedback connections providing powerful dynamic mapping and representational capabilities. The main difference of the proposed network (MFLNN) compared to the available RNNs is that the temporal relations are provided by means of neurons arranged in three feedback layers, not by simple feedback elements, in order to enrich the representation capabilities of the recurrent networks.

Another motivation of thesis is to design and implement a neural-network-based nonlinear channel equalizer under the consideration of tradeoffs between the hardware chip size, the processing speed, and the cost.

1.2 Thesis Layout

Chapter2 introduces artificial neural network and illustrates its learning process.

Chapter 3 discussed the architecture and Training procedure of a novel recurrent network,referred as MFLNN.The performance of the MFLNN demonstrated by applying to several illustrative temporal problems including chatic time series prediction and nonlinear system identification.

Chapter 4 introduces basic theory of channel equalizer

Chapter 5 explains the applications of neural network techniques on digital communication systems. We compare the performance of two different structures of equalizer, namely, the linear least-mean-square-based equalizer (LIN) and the functional link artificial neural networks (FLANN).

In chapter 6 discussed the hardware implementation of equalizers for transmissions through nonlinear communication channels based on artificial neural networks structure. After the designing procedure is finished, the circuit implemented using hardware description languages (HDLs). We choose field-programmable-gate-array (FPGA) devices for the hardware realization of our channel equalizer. And compared performance of two different structures of equalizer

Chapter 7 summarizes the work done in this thesis work

Chapter 2

CONCEPTS OF NEURAL NETWORK

2.1 Single Neuron Structure

A neuron is an information processing unit that is fundamental to the operation of a neural network. The three basic elements of the neuronal model:

1. A set of synapses or connecting links, each of which is characterized by a weight or streght of its own.
2. An adder for summing the input signals, weighted by the respective synapses of the neuron
3. An activation function for limiting the amplitude of the output of a neuron. The activation function is also referred to As a squashing function in that it squashes the permissible amplitude range of the output signal to some finite value.

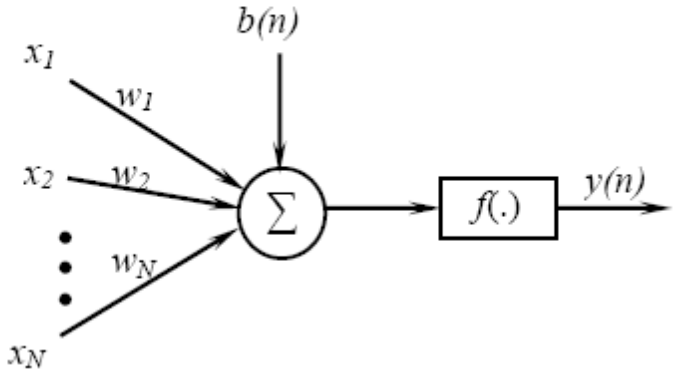


Figure 2.1 Single Neuron structure

The structure of a single neuron is presented in Fig. 2.1.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,

$$y(n) = f\left[\sum_{j=1}^{N} w_j(n)x_j(n) + b(n)\right]$$

Where x_1, x_2, x_n are the input signals ; $w_j(n)$ = weight associated with the j^{th} input,

$b(n)$ = threshold to the neuron is called as bias.

and N = no. of inputs to the neuron.

2.2 Activation Functions and Bias.

The perceptron internal sum of the inputs is passed through an activation function, which can be any monotonic function. Linear functions can be used but these will not contribute to a non-linear transformation within a layered structure, which defeats the purpose of using a neural filter implementation. A function that limits the amplitude range and limits the output strength of each perceptron of a layered network to a defined range in a non-linear manner will contribute to a nonlinear transformation. There are many forms of activation functions, which are selected according to the specific problem. All the neural network architectures employ the activation function which defines as the output of a neuron in terms of the activity level at its input (ranges from -1 to 1 or 0 to 1). Table 2.1 summarizes the basic types of activation functions. The most practical activation functions are the sigmoid and the hyperbolic tangent functions. This is because they are differentiable.

The bias gives the network an extra variable and the networks with bias are more powerful than those of without bias. The neuron without a bias always gives a net input of zero to the activation function when the network inputs are zero. This may not be desirable and can be avoided by the use of a bias.

Name	Definition
Linear	$f(x) = kx$
Step	$f(x) = \beta, \quad if\ x \geq k$ $= \delta, \quad if\ x < k$
Sigmoid	$f(x) = \frac{1}{1+e^{-\alpha x}}, \alpha > 0$
Hyperbolic Tangent	$f(x) = \tanh(\gamma x) = \frac{1-e^{-\gamma x}}{1+e^{-\gamma x}}, \gamma > 0$
Gaussian	$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$

Table 2.1:Common activation function

2.3 Learning Processes

The property that is of primary significance for a neural network is that the ability of the network to learn from its environment, and to improve its performance through learning. The improvement in performance takes place over time in accordance with some prescribed measure. A neural network learns about its environment through an interactive process of adjustments applied to its synaptic weights and bias levels. Ideally, the network becomes more knowledgeable about its environment after each iteration of learning process. Hence we define learning as: "It is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded."

The processes used are classified into two categories as

(A) Supervised Learning (Learning With a Teacher)

(B) Unsupervised Learning (Learning Without a Teacher)

2.3.1 Supervised Learning:

We may think of the teacher as having knowledge of the environment, with that knowledge being represented by a set of input-output examples. The environment is, however unknown to neural network of interest. Suppose now the teacher and the neural network are both exposed to a training vector, by virtue of built-in knowledge, the teacher is able to provide the neural network with a desired response for that training vector. Hence the desired response represents the optimum action to be performed by the neural network. The network parameters such as the weights and the thresholds are chosen arbitrarily and are updated during the training procedure to minimize the difference between the desired and the estimated signal. This updation is carried out iteratively in a step-by-step procedure with the aim of eventually making the neural network emulate the teacher. In this way knowledge of the environment available to the teacher is transferred to the neural network. When this condition is reached, we may then dispense with the teacher and let the neural network deal with the environment completely by itself. This is the form of supervised learning.

The update equations for weights are derived as LMS :

$$w_j(n+1) = w_j(n) + \mu \Delta w_j(n)$$

$\Delta w_j(n)$ is the change in w_j in nth iteration.

2.3.2 Unsupervised Learning

In unsupervised learning or self-supervised learning there is no teacher to over-see the learning process, rather provision is made for a task independent measure of the quantity of representation that the network is required to learn, and the free parameters of the network are optimized with respect to that measure. Once the network has become turned to the statistical regularities of the input data, it develops the ability to form the internal representations for encoding features of the input and thereby to create new classes automatically. In this learning the weights and biases are updated in response to network input only. There are no desired outputs available. Most of these algorithms perform some kind of clustering operation. They learn to categorize the input patterns into some classes.

2.4 Recurrent Neural Network

A strict feedforward architecture does not maintain a short-term memory. Any memory effects are due to the way past inputs are re-presented to the network (as for the tapped delay line). A simple recurrent network (SRN; (Elman, 1990)) has activation feedback which embodies short-term memory. A state layer is updated not only with the external input of the network but also with activation from the previous forward propagation. The feedback is modified by a set of weights as to enable automatic adaptation through learning (e.g. backpropagation). Recurrent network are the neural network with one or more feedback loop. The feedback can be of a local or global kind. Recurrent neural networks (RNNs) naturally involve dynamic elements in the form of feedback connections providing powerful dynamic mapping and representational capabilities. Figure below The application of feedback enables recurrent network to acquire state representations, which make them suitable devices for such diverse applications as nonlinear prediction and modeling, adaptive equalization speech processing, plant control, automobile engine diagnostics

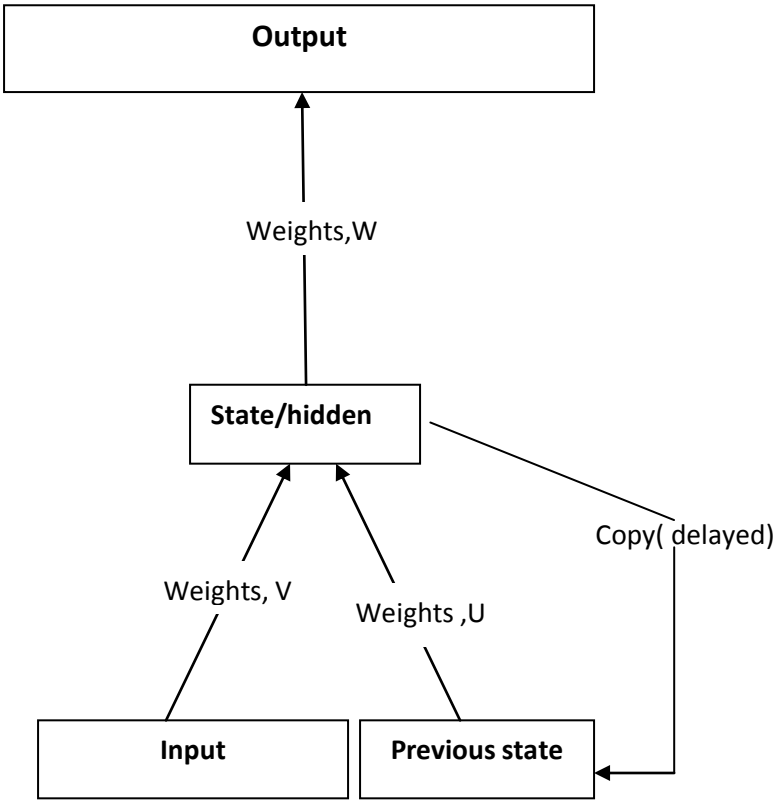


Figure 2. 2 A simple Recurrent network

2.5 The Back-Propagation Algorithm

In order to train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (EW). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The back propagation algorithm is the most widely used method for determining the error derivatives of the weights.

The back-propagation algorithm is easiest to understand if all the units in the network are linear. The algorithm computes each EW by first computing the EA, the rate at which the error changes as the activity level of a unit is changed. For output units, the EA is simply the difference between the actual and the desired output. To compute the EA for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those weights by the EAs of those output units and add the products. This sum equals the EA for the chosen hidden unit. After calculating all the EAs in the hidden layer just before the output layer, we can compute in like fashion the EAs for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the EA has been computed for a unit, it is straight forward to compute the EW for each incoming connection of the unit. The EW is the product of the EA and the activity through the incoming connection.Note that for non-linear units, the back-propagation algorithm includes an extra step. Before back-propagating, the EA must be converted into the EI**,** the rate at which the error changes as the total input received by a unit is changed.

The steps involve in applying back propagation algorithm is as described below :

First the input is propagated through the ANN to the output. After this the error e_k on a single output neuron k can be calculated as:

$$e_k = d_k - y_k \tag{2.1}$$

Where y_k is the calculated output and d_k is the desired output of neuron *k*. This error value is used to calculate a δ_k value, which is again used for adjusting the weights. The δ_k value is calculated by:

$$\delta_k = e_k g'(y_k) \tag{2.2}$$

$$\delta_j = \eta g'(y_j) \sum_{k=0}^{K} \delta_k w_{jk} \tag{2.3}$$

Where *K* is the number of neurons in this layer and η is the learning rate parameter, which determines how much the weight should be adjusted. The more advanced gradient descent algorithms does not use a learning rate, but a set of more advanced parameters that makes a more qualified guess to how much the weight should be adjusted.

Using these δ values, the Δw values that the weights should be adjusted by, can be calculated by:

$$\Delta w_{jk} = \delta_k y_k \tag{2.4}$$

The Δw_{jk} value is used to adjust the weight w_{jk}, by $w_{jk} = w_{jk} + \Delta w_{jk}$ and the backpropagation algorithm moves on to the next input and adjusts the weights according to the output. This process goes on until a certain stop criteria is reached. The stop criteria is typically determined by measuring the mean square error of the training data while training with the data, when this mean square error reaches a certain limit, the training is stopped. More advanced stopping criteria involving both training and testing data are also used.

Chapter 3

MULTIFEED-BACK LAYER NEURAL NETWORK AND ITS APPLICATION

3.1 Introduction

The majority of physical systems contain complex nonlinear relations, which are difficult to model with conventional techniques. Neural networks (NNs) have learning, adaptation, and powerful nonlinear mapping capabilities. Therefore, they have been studied to deal with predicting, modeling, and control of complex, nonlinear, and uncertain systems, in which the conventional methods fail to give satisfactory results .The NNs can be classified as static (feedforward) and dynamic (recurrent). Because of their inherent feedforward structure, the role of the static NNs are limited to realize static mappings. However, the output of a dynamic system is a function of past outputs and past inputs. In order to use them for identification of nonlinear dynamical systems, all necessary past inputs and past outputs of the dynamic system have to be fed to the static NN, explicitly; so, the number of delayed inputs and outputs should be known in advance. The use of the long tapped delay input increases the input dimensions resulting in curse of dimensionality problem [16].

Recurrent neural networks (RNNs) naturally involve dynamic elements in the form of feedback onnections providing powerful dynamic mapping and representational apabilities. They are able to learn the system dynamics without assuming much knowledge about the structure of the system under consideration such as the number of delayed inputs and outputs. Furthermore, the recurrent systems can inherently produce multistep ahead predictions; so, the multistep ahead prediction models, which are required in some process control applications, such as predictive control, can efficiently be built by RNNs . Thus, the RNNs have attracted great interest. The Hopfield , the Elman , the Jordan , the fully recurrent , the locally-recurrent , the memory neuron , the recurrent radial basis function, and the block-structured recurrent networks are some of the examples of RNNs. The Hopfield network is a simple recurrent network which has a fully connected single-layer structure. It is capable of restoring previously learned static patterns from their corrupted realizations. Elman and Jordan proposed specific recurrent networks which have an extra set of context nodes that copy the delayed states of the hidden or output nodes back to the hidden layer neurons. In these structures, the feedback weights, assumed to be unity, are not trainable. The fully recurrent neural network allows any neuron to be connected to any other neuron in the network. While being more general, it lacks stability. In , the local feedback has been taken before the entry into the nonlinearity activation function, while

in , it has been taken after the nonlinearity. In the memory neuron network , each feedforward neuron is associated with a memory neuron the single scalar output of which summarizes the history of past activation of that unit. In the past output values of a radial basis function network are fed back to both the network input and output nodes. In a systematic way to build networks of high complexity using a block notation was given.

Recently, a number of recurrent fuzzy neural network (RFNN) structures appeared in the literature. Dynamic fuzzy logic systems (DFLSs) and their nonsingleton generalizations were investigated in [17]. In [11], a recurrent neurofuzzy network was proposed to build long-term prediction models for nonlinear processes. In [18], a recurrent self-organizing neural fuzzy inference network was constructed by realizing dynamic fuzzy reasoning. In [19], an RFNN structure was proposed by realizing fuzzy inference using dynamic fuzzy rules. In [20], a Takagi–Sugeno–Kang (TSK)-type RFNN was developed from a series of recurrent fuzzy if-then rules with TSK-type consequent parts. In [21], a type of RFNN called additive delay feedback neural-fuzzy networks trained with the backpropagation approach was proposed. In [22], a DFNN consisting of the recurrent TSK rules was developed. Its premise and defuzzification parts are static while its consequent part rules are recurrent neural networks with internal feedback and time delay synapses. In [23], a wavelet-based RFNN was developed by combining the traditional TSK fuzzy model and the wavelet neural networks with some feedback connections.

In this paper, the architecture and training procedure of a new RNN, called the multifeedback-layer neural network (MFLNN), are presented. The structure of the proposed MFLNN differs from the other RNNs in the literature. The main difference of the proposed network compared to the available RNNs is that the temporal relations are provided by means of neurons arranged in three feedback layers, not by simple feedback elements, in order to enrich the representation capabilities of the recurrent networks. The feedback signals are processed in three feedback layers which contain nonlinear pocessing elements (neurons) as in feedforward layers. In these feedback layers, the weighted sums of the delayed outputs of the hidden and output layers are passed through activation functions and applied to the feedforward neurons via some adjustable weights.

Both online and offline training procedures based on the backpropagation through time BPTT) algorithm have been investigated [11]. The adjoint model of the MFLNN is built to

compute the derivatives with respect to the network weights, which are used for training purpose. It is shown that the offline training fails to adapt to changes in system dynamics. Hence, an online training procedure is derived. In this procedure, the online adjustment of the weights is performed over a certain history of the input–output data stored in a stack. The stack discards the oldest pattern and accepts a new pattern from the system at each time step. Therefore, the stack contains enough data to represent plant dynamics, and eliminates the too old data to adapt to the changes in system dynamics at each time step. The derivatives for the MFLNN weights are computed with a type of truncated BPTT algorithm in a manner which gives the same result as the unfolding of the MFLNN in time through the stack. The Levenberg–Marquardt (LM) method with a trust region approach is used to adjust the MFLNN weights .The learning, adaptation, and generalization performances of the developed MFLNN are tested by applying several temporal problems including chaotic time series prediction and nonlinear dynamic system identification. Performance comparisons are made against several networks suggested in the literature.

3.2 LM algorithm

Standard Levenberg-Marquardt algorithm, a variation on the error back-propagation algorithm, provides us with a good switching capability between the Gauss-Newton algorithm and the Steepest Descent method. The quadratic performance index F(w) to be minimized is sum of the squares of the error between desired output and actual output for all patterns as given by

$$F(w) = \underline{e}^T \underline{e} \tag{3.1}$$

In Eq.(3.1), e is the error vector defined by

$$\underline{e} = \left[e_{1,1} \cdots e_{R,1}\, e_{1,2} \cdots e_{R,2} \cdots e_{1,Q} \cdots e_{R,Q} \right]^T$$

where $e_{r,q}$ is error between $d_{r,q}$ (desired value for the r^{th} output and q^{th} pattern) and $a_{r,q}$ (actual value of the r^{th} output for q^{th} pattern), Q is the number of patterns, and R is the number of outputs. Moreover, w is the parameter vector given by

$$\underline{w} = [w_1 w_2 ..... w_M]^T$$

where M is the number of adjustable parameters. Eq.(3.1) can also be written as

$$F(\underline{w}) = \sum_{q=1}^{Q}\sum_{r=1}^{R}\left(d_{r,q} - a_{r,q}\right)^2$$

$$= \sum_{q=1}^{Q}\sum_{r=1}^{R}\left(e_{r,q}\right)^2 \qquad (3.2)$$

In general, the update rule is

$$\Delta\underline{w}_k = \underline{w}_k + \Delta\underline{w}_k \qquad (3.3)$$

In the Newton's method, adjustable parameters are updated by

$$\Delta\underline{w}_k = \left[H(w_k)\right]^{-1} g(w_k) \qquad (3.4)$$

where H(wk) is the Hessian matrix given by

$$\underline{\underline{H}} = \begin{bmatrix} \frac{\partial^2 F(\underline{w}_k)}{\partial w_1^2} & \frac{\partial^2 F(\underline{w}_k)}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 F(\underline{w}_k)}{\partial w_1 \partial w_M} \\ \frac{\partial^2 F(\underline{w}_k)}{\partial w_2 \partial w_1} & \frac{\partial^2 F(\underline{w}_k)}{\partial w_2^2} & \cdots & \frac{\partial^2 F(\underline{w}_k)}{\partial w_2 \partial w_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F(\underline{w}_k)}{\partial w_M \partial w_1} & \frac{\partial^2 F(\underline{w}_k)}{\partial w_M \partial w_2} & \cdots & \frac{\partial^2 F(\underline{w}_k)}{\partial w_M^2} \end{bmatrix} \qquad (3.5)$$

and g(wk) is the gradient vector given by

$$\underline{g}(\underline{w}_k) = \left[\frac{\partial F(\underline{w}_k)}{\partial w_1} \quad \frac{\partial F(\underline{w}_k)}{\partial w_2} \quad \cdots \quad \frac{\partial F(\underline{w}_k)}{\partial w_M}\right]^T \qquad (3.6)$$

The gradient vector and the Hessian matrix can be written in terms of the Jacobian matrix as

$$\underline{g}(\underline{w}_k) = 2\underline{\underline{J}}^T(\underline{w}_k)\underline{e}_k \qquad (3.7)$$

And

$$\underline{\underline{H}}(\underline{w}_k) = 2\underline{\underline{J}}^T(\underline{w}_k)\underline{\underline{J}}(\underline{w}_k) + neglected\ terms \tag{3.8}$$

Where J(wk) is the RQxM Jacobian matrix given by

$$\underline{\underline{\mathbf{J}}}(\underline{w}_k) = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_1} & \frac{\partial e_{1,1}}{\partial w_2} & \cdots & \frac{\partial e_{1,1}}{\partial w_M} \\ \frac{\partial e_{2,1}}{\partial w_1} & \frac{\partial e_{2,1}}{\partial w_2} & \cdots & \frac{\partial e_{2,1}}{\partial w_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{R,1}}{\partial w_1} & \frac{\partial e_{R,1}}{\partial w_2} & \cdots & \frac{\partial e_{R,1}}{\partial w_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{1,Q}}{\partial w_1} & \frac{\partial e_{1,Q}}{\partial w_2} & \cdots & \frac{\partial e_{1,Q}}{\partial w_M} \\ \frac{\partial e_{2,P}}{\partial w_1} & \frac{\partial e_{2,P}}{\partial w_2} & \cdots & \frac{\partial e_{2,P}}{\partial w_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_{R,Q}}{\partial w_1} & \frac{\partial e_{R,Q}}{\partial w_2} & \cdots & \frac{\partial e_{R,Q}}{\partial w_M} \end{bmatrix} \tag{3.9}$$

Afterwards, substitution of (3.7) and (3.8) into (3.4) yields the update rule for Gauss-Newton method, and the weight updates are calculated by

$$\Delta w_k = -[\underline{J}^T(w_k)\,\underline{J}(w_k)]^{-1}\,\underline{J}^T(w_k)e_k \tag{3.10}$$

Whereas the neglected terms in Eq.(3.8) may cause some accuracy errors in the calculation of the Hessian, the most significant advantage of the Gauss-Newton method over Newton's method is the elimination of the necessity of calculation of the second derivatives. Furthermore, the fact that the matrix [JT(wk)J(wk)] may be singular constitutes the main disadvantage of the Gauss- Newton method. In the Levenberg-Marquardt algorithm, the singularity problem in the Gauss-Newton method is overcome by introducing an additional term, which as well provides a

good switching between the Steepest Descent and the Gauss-Newton method. For standard Levenberg-Marquardt algorithm, adjustable parameters are updated by

$$\Delta w_k = -[J^T(w_k)\,J(w_k) + \mu_k I]^{-1}\,J^T(w_k)\,e_k \qquad (3.11)$$

where μ is the learning rate, I is identity matrix . During training process the learning rate μ is incremented or decremented by a scale at weight updates. As the learning rate draws closer to zero, the Levenberg-Marquardt algorithm approaches the Gauss-Newton method, while it approaches the Steepest Descent algorithm as the learning rate takes large value.

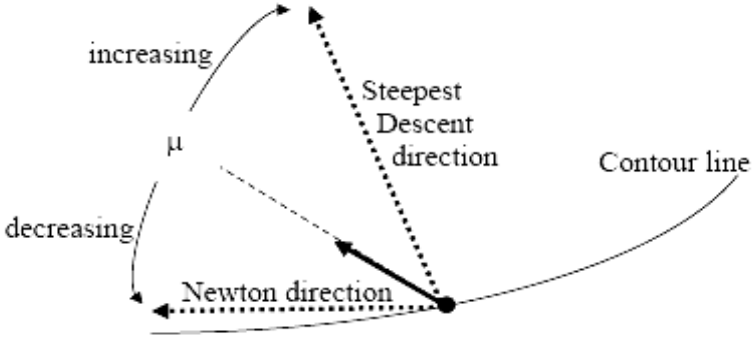


Figure 3.1 Transition between the Steepest Descent and the Gauss-Newton

Although the Levenberg-Marquardt algorithm gives a good compromise between those methods, its main disadvantage, as can be seen from Eq.(3.11), is the necessity of computation of $[J^T(wk)J(wk) + \mu kI]^{-1}$ square matrix at every weight updates, the dimension of which is MxM. In [18], one modification on the performance index is proposed in order to reduce the abovementioned computational complexity, where the performance index given by Eq.(3.2) is replaced with the performance index given by Eq.(3.12),

$$F(\underline{w}) = \sum_{r=1}^{R}\left[\sum_{q=1}^{Q}\left(d_{r,q} - a_{r,q}\right)^2\right]^2 \qquad (3.12)$$

The new performance index can also be written in a quadratic form :

$$F(\underline{w}) = \hat{\underline{e}}^T\hat{\underline{e}}$$

where e is the new error vector $\hat{\underline{e}} = \left[\hat{e}_1\ \hat{e}_2 \cdots \hat{e}_R\right]^T$ and $\hat{\underline{e}}_r = \sum_{q=1}^{Q} (d_{r,q} - a_{r,q})^2$, for r = 1,…,R.

It can be observed that the continuity requirements are still preserved and that the new measure can well be considered as a measure of similarity between the desired and the produced patterns. By this modification, proposed in [18], the new update rule is then written by,

$$\Delta \underline{w}_k = -\left[\frac{1}{\mu_k}\underline{\underline{I}} - \frac{1}{\mu_k^2}\hat{\underline{\underline{J}}}^T(\underline{w}_k)\left(\underline{\underline{I}} + \frac{1}{\mu_k}\hat{\underline{\underline{J}}}(\underline{w}_k)\hat{\underline{\underline{J}}}^T(\underline{w}_k)\right)^{-1}\hat{\underline{\underline{J}}}(\underline{w}_k)\right]\hat{\underline{\underline{J}}}^T(\underline{w}_k)\hat{\underline{e}} \tag{3.13}$$

where ˆJ is the new Jacobian matrix, the size of which is now RxM. Consequently, in the new update rule the size of the matrix to be inverted becomes RxR. In most neural network applications R is less than M. Another modification investigated during the study is on the gradient computation of the sigmoidal activation function, which is proposed in .This modification aims at improving the slow asymptotic convergence rate of the error-back propagation algorithm by using the slope of the line connecting the output value and the desired value instead of using derivative of the activation function as the gradient information. In the limit case that the output value approaches the desired value, the calculated slope becomes very near to the calculated derivative of the activation function, and then both algorithms become identical.

3.3 Multifeedback layer Neural Network (MFLNN)

input and output of the MFLNN, respectively, and k is the time index. The MFLNN has three feedforward and feedback layers. In the feedforward layers, W_1 and W_2 represent the weights between the input and hidden layers, and the hidden and output layers, respectively. In addition to the feedforward layers, the MFLNN has two local and one global feedback layers. In these feedback layers, the weighted sums of the delayed outputs of the hidden and output layers are applied to certain activation functions as in the feedforward layer neurons. $W^b{}_1$, $W^b{}_2$ and $W^b{}_3$ represent the weights connected to the inputs of the feedback layer neurons and represents the time delay operators. The outputs of the feedback layers neurons ($h^c(k), y^c(k), and\ Z^c(k)$) are applied to the hidden and output layers neurons via the adjustable weights ($W^c{}_1, W^c{}_2 and W^c{}_3$). The bias connections to the neurons are omitted to simplify the resentation in Fig. 1.

As a rule of thumb, the number of plant states should be a good starting value for the number of neurons in the hidden layer. The number of neurons in the feedback layer from the hidden-to-hidden layer is set equal to the number of the hidden layer neurons. The number of neurons in the feedback layer from the output-to-hidden layer is set equal to the number of the output layer neurons. The number of neurons in the feedback layer from the output-to-output layer is set equal to the number of the output layer neurons. However, their numbers can be incremented to perhaps improve the accuracy. One uses trial–error or previous data about the system to come up with a proper number.

Fig. 2 depicts the details of the MFLNN where each layer is simply represented by only one of its neurons boxed in dashed lines. In the figure, the neurons of each feedback layer are labeled by their connection to their corresponding inputs and outputs.

To train the recurrent systems, the BPTT-like derivative calculation is required. However, the calculation of the derivatives by using the chain rule or by the unfolding in time is very complicated, so we built the adjoint model of the MFLNN, which is depicted in Fig. 3, to simplify the computations. It is constructed by reversing the branch directions, replacing summing junctions with branching points and vice versa, and replacing the time delay operators

with time advance operators. The Jacobian matrix or the gradient vector is easily computed by means of the adjoint model of the MFLNN. Since the weights are updated by the LM method

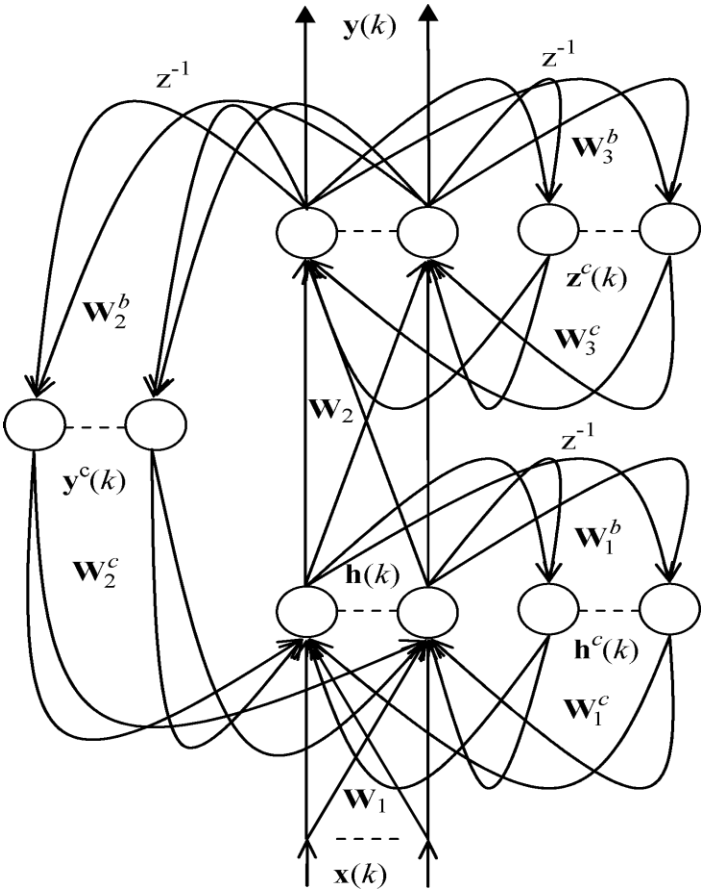


Figure 3.2 Structure of MFLANN

the calculation of the Jacobian matrix is required. The elements of the Jacobian matrix for an output of the MFLNN are computed by feeding 1 instead of the corresponding error value in the adjoint model and 0 for others. The backward phase computations from k=T to k=1 are performed by means of the adjoint model of the MFLNN. When the forward and ackward phases of the computations are completed, the sensitivities for each weight, which form the Jacobian matrix, are obtained as in the BPTT algorithm.

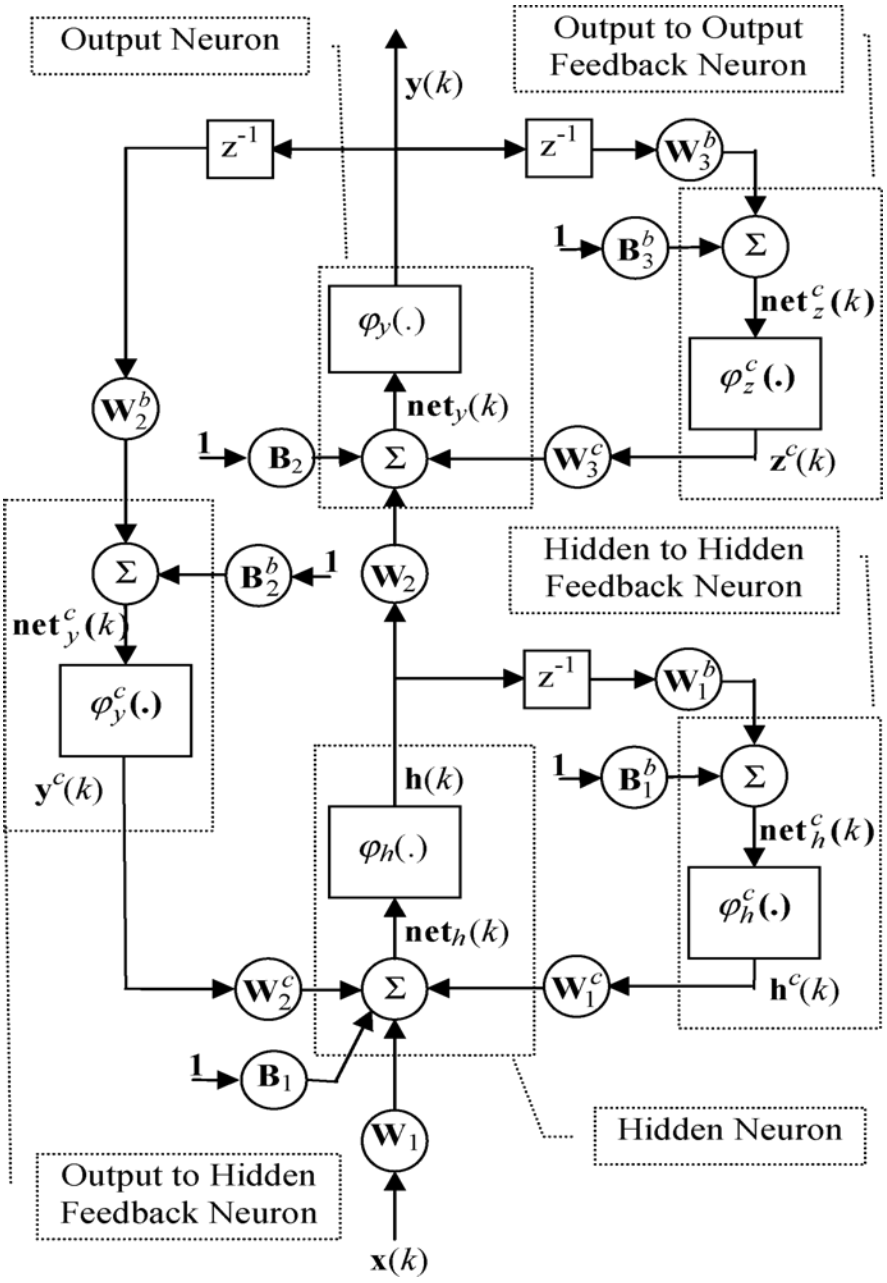


Figure 3.3 Layers of the MFLNN

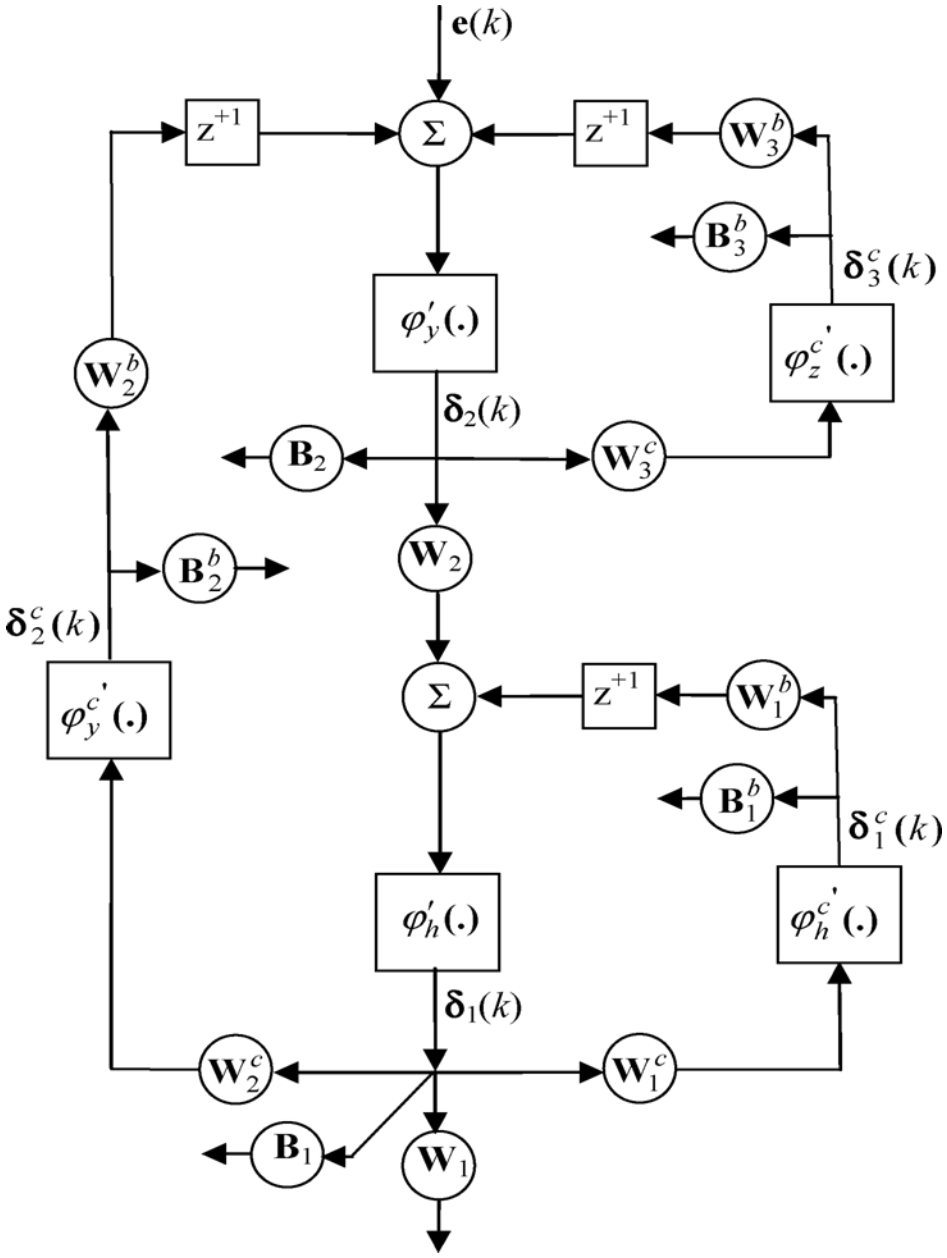


Figure 3.4 Adjoint model of the MFLANN

As it was expressed previously, the elements of the Jacobian matrix are computed in two stages which are eferred to as the forward and backward phases. In the forward phase, the MFLNN actions are computed and stored from k=1 to k=T through the trajectory. The errors at every are determined as the differences between the desired outputs and the MFLNN outputs. The initial

values for the output of the hidden layer(h) and of the output layer are (y) set to 0
h(0)=0 y(0)=0

The induced local fields (net quantities) produced at the input of the activation functions of the feedback neurons are

$$net^{c}{}_{h}(k) = [W^{b}{}_{1} h(k-1)] + B^{h}{}_{1}$$

$$net^{c}{}_{y}(k) = [W^{b}{}_{1} y(k-1)] + B^{h}{}_{2} \qquad (3.14)$$

$$net^{c}{}_{z}(k) = [W^{b}{}_{1} y(k-1)] + B^{h}{}_{3}$$

Where $W^{b}{}_{1}, W^{b}{}_{2}$ and $W^{b}{}_{3}$ are the input weights of the feed back layers. $B^{b}{}_{1}$, $B^{b}{}_{2}$,and $B^{b}{}_{3}$ are the biases of the feedback layer neurons. Then, the outputs of the feedback layer neurons(h^{c} , y^{c} and Z^{c}) are computed by,

$$h^{c}(k) = \varphi^{c}{}_{h}(net^{c}{}_{h}(k))$$

$$y^{c}(k) = \varphi^{c}{}_{h}(net^{c}{}_{y}(k)) \qquad (3.15)$$

$$z^{c}(k) = \varphi^{c}{}_{z}(net^{c}{}_{z}(k))$$

Where $\varphi^{c}{}_{h}$, $\varphi^{c}{}_{y}$, and $\varphi^{c}{}_{z}$ represents the activation function of the feedback layer neuron.The net quantities (net_{h}) of the hidden layer nerons and their output(h) are computed by

$$net_{h}(k) = [W_{1} x(k)] + [W^{c}{}_{1} h^{c}(k)] + [W^{c}{}_{2} y^{c}(k)] + B1 \qquad (3.16)$$

$$h(k) = \varphi_{h}(net_{h}(k))$$

Where W1 represents the weights between the input and hidden layers, and B1 the biases applied to the hidden layer neurons. $W^{c}{}_{1}$ and $W^{c}{}_{2}$ are the output weights of the feedback layers. φ_{h} represents the hidden layer activation functions. Similarly, the net quantities of the output layer neurons and their outputs are computed by

$$net_h(k) = [W_1 x(k)] + [W^c{}_1 h^c(k)] + [W^c{}_2 y^c(k)] + B1$$

$$h(k) = \varphi_h(net_h(k))$$

Where W_2, B_2 and φ_y represent the weights between the .hidden and output layers, neurons, and the output layer activation function ,respectively. $W^c{}_3$ represents the output weights of the feedback layer.

The error (e) signal is defined as the difference between the MFLNN output () and the desired output (y) and the desired output (y_d)

$$e(k) = y(k) - yd(k) \tag{3.17}$$

We define the instantaneous value of the error energy, which is a function of all the free parameters, as

$$E(k) = \frac{1}{2}(e(k)^T e(k)) \tag{3.18}$$

Then, the cost function defined as a measure of the learning performance is

$$E_{total} = \frac{1}{T}\sum_{k=1}^{T} E(k) \tag{3.19}$$

The weights are adjusted to minimize the cost function, so the sensitivities with respect to each weight have to be computed. At every k, the sensitivity for each weight is computed by multiplyingthe input of this weight in the MFLNN and the adjoint model, so the inputs of the weights in the adjoint model have to be computed.Therefore, after completing the forward phase computations, the backward phase computation is carried out through the adjoint model of MFLNN from k=T to k=1. The local sensitivities at k=T+1 are set to 0

$$\delta^c{}_3(T+1) = 0 \quad , \quad \delta^c{}_2(T+1) = 0 \quad , \quad \delta^c{}_1(T+1) = 0$$

The derivatives of the activation functions of each layer with respect to their inputs are computed as

$$\varphi^{c'}{}_z(net^c{}_z(k)) = \frac{\partial \varphi^c{}_z(net(k))}{\partial net(k)}\Big| net = net^c{}_z(k)$$

$$\varphi^{c'}{}_y(net^c{}_y(k)) = \frac{\partial \varphi^c{}_y(net(k))}{\partial net(k)}\Big| net = net^c{}_y(k)$$

$$\varphi^{c'}{}_h(net^c{}_h(k)) = \frac{\partial \varphi^c{}_h(net(k))}{\partial net(k)}\Big| net = net^c{}_h(k) \qquad (3.20)$$

$$\varphi^{'}{}_y(net^c{}_y(k)) = \frac{\partial \varphi{}_y(net(k))}{\partial net(k)}\Big| net = net{}_y(k)$$

$$\varphi^{'}{}_h(net^c{}_h(k)) = \frac{\partial \varphi{}_h(net(k))}{\partial net(k)}\Big| net = net{}_h(k)$$

The local sensitivities are obtained as

$$\delta_2(k) = [\varphi^{'}{}_y(net{}_y(k))][e(k) + (W^b{}_2)^T \delta^c{}_2(k+1) + ((W^b{}_3)^T \delta^c{}_3(k+1)]$$

$$\delta_1(k) = [\varphi^{'}{}_y(net{}_y(k))][((W^b{}_1)^T \delta^c{}_1(k+1)) + (W^T{}_2 \delta_2(k))]$$

$$\delta^c{}_3(k) = [\varphi^{c'}{}_z(net^c{}_z(k))][(W^c{}_3)^T \delta_2(k)]$$

$$\delta^c{}_2(k) = [\varphi^{c'}{}_y(net^c{}_y(k))][(W^c{}_2)^T \delta_1(k)]$$

$$\delta^c{}_1(k) = [\varphi^{c'}{}_y(net^c{}_y(k))][(W^c{}_3)^T \delta_2(k)]$$

In the case of the calculation of the Jacobian matrix, e(k) is set to in (11). Then, the sensitivity or each weight is computed by multiplying the values scaled by this weight in the MFLNN and the adjoint model as follows:

$$\frac{\partial e(k)}{\partial W2} = \delta_2(k)h^T(k) \qquad \frac{\partial e(k)}{\partial B_2} = \delta_2(k)$$

$$\frac{\partial e(k)}{\partial W1} = \delta_2(k)X^T(k) \qquad \frac{\partial e(k)}{\partial B1} = \delta_1(k)$$

$$\frac{\partial e(k)}{\partial W^c{}_3} = \delta_2(k)Z^{cT}(k) \qquad \frac{\partial e(k)}{\partial W^b{}_3} = \delta^3{}_2(k)y^T(k-1)$$

$$\frac{\partial e(k)}{\partial W^c{}_2} = \delta_1(k)y^{cT}(k) \qquad \frac{\partial e(k)}{\partial W^b{}_2} = \delta^c{}_2(k)y^T(k-1) \qquad (3.21)$$

$$\frac{\partial e(k)}{\partial W^c{}_1} = \delta_1(k)h^{cT}(k) \qquad \frac{\partial e(k)}{\partial W^b{}_2} = \delta^c{}_1(k)h^T(k-1)$$

$$\frac{\partial e(k)}{\partial B^b{}_3} = \delta^c{}_3(k) \qquad \frac{\partial e(k)}{\partial B^b{}_2} = \delta^c{}_2(k) \qquad \frac{\partial e(k)}{\partial B^b{}_1} = \delta^c{}_1(k)$$

The overall sensitivity for each weight is obtained by summing the related sensitivity in (3.21) over the trajectory. The Jacobian matrix which is required to train the MFLNN is

$$J = \left(\frac{\partial e}{\partial W1} \frac{\partial e}{\partial W2} \frac{\partial e}{\partial W^c{}_1} \frac{\partial e}{\partial W^b{}_1} \frac{\partial e}{\partial B^b{}_1} \frac{\partial e}{\partial W_2} \frac{\partial e}{\partial B_2} \frac{\partial e}{\partial W^c{}_2} \frac{\partial e}{\partial W^b{}_2} \frac{\partial e}{\partial B^b{}_2} \frac{\partial e}{\partial W^c{}_3} \frac{\partial e}{\partial W^c{}_3} \frac{\partial e}{\partial B^b{}_3} \right)$$

The gradient vector is computed from the Jacobian matrix is by

$$g = J^T e \qquad (3.22)$$

The network weight vector w is defined as

$$W = [W_1, B_1, W^c{}_1, W^b{}_1, B_1{}^c B_2, W^b{}_3 B^b{}_1 B^b{}_2, W^b{}_2, W^c{}_3, W^b{}_3, B^b{}_3]$$

The change in the weight vector ΔW_n at the th iteration is computed by the LM method

$$[J_n{}^T J_n + \mu_n I)\Delta W_n = -J^T{}_n e_n \qquad (3.23)$$

where $\mu_n \geq 0$ is a scalar and is the identity matrix. For a sufficiently large value of , the matrix $[J_n{}^T J_n + \mu_n I)$ is positive definite and ΔW_n is a descent direction. When $\mu_n = 0$, ΔW_n is the Gauss–Newton vector. As $\mu_n \to \infty$, $\mu_n I$ term dominates so that represents an infinitesimal step in the steepest descent direction. We used the trust region approach of Fletcher to determine μ_n

3.4 Online training Structure.

The online training procedure of the MFLNN is described in Fig. 4. In this procedure, a short history of the training patterns is stored in a first-in–first-out (FIFO) stack with a certain size . The stack discards the oldest pattern from it and accepts a new pattern from the system at each time step. Therefore, the stack should be properly sized to contain enough data to represent system dynamics and eliminate the too old data to adapt

to the changes in system dynamics at each time step. The sensitivities of the MFLNN weights are computed in the manner of unfolding the MFLNN in time through the stack. The online training is performed using the entire patterns stored in the stack at each time step by the LM method with the trust region approach.

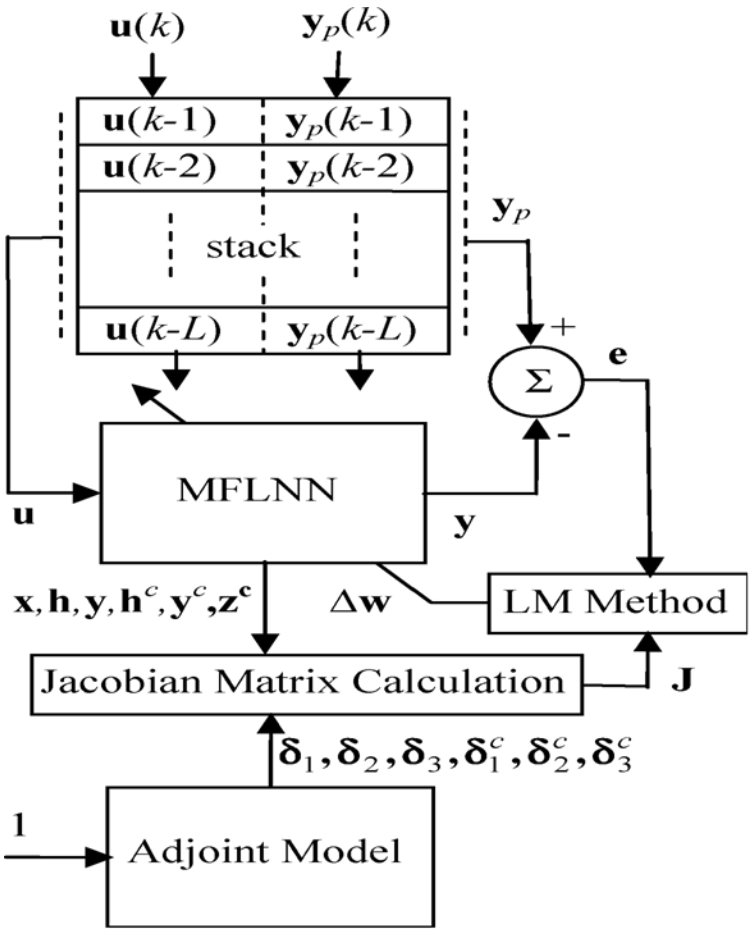


Figure 3.5 Online Training structure of the MFLANN

The training of the MFLNN is performed by adjusting the weight vector at each time step as

$$W_{k+1} = W_k + \Delta W_k$$

The cost function at each time step is given as

$$E = \frac{1}{2L}\sum_{m=0}^{L-1}[y(k-m) - y_p(k-m)]^T\,[y(k-m) - y_p(k-m)]$$

Where L is the stack size which determines the data length and m is the time delay index . y_p and y represent the desired output and the MFLNN output, respectively. The training is performed over the L patterns at each time step. The elements of the Jacobian matrix are computed in two

stages which are referred to as the forward and backward phases. The forward phase computations are performed, by starting L-1 steps back in time to the present time, from time

k-L+1 to *k* , and the values of the variables (x ,h , y,h^c , y^c, and z^c) are stored at each time step. The error values, which are obtained as the difference between the plant outputs stored in the stack and the MFLNN output, are also stored. After completing the forward phase computations, the backward phase computation is carried out through the adjoint model of MFLNN, starting from the present time , going backward by steps to time . Finally, the elements of the Jacobian matrix are computed by means of the forward and backward computations. These computations correspond to a type of BPTT algorithm by unfolding the MFLNN in time through the stack. The procedure operates online and the dynamic derivative calculation is performed over more than one pattern in the stack avoiding the shortcomings of static gradient calculation and of employing only one pattern at a time.

3.5 Application of MFLNN

3.5.1 Predicting Chaotic Time Series

In this first example, the learning and generalization performance of the MFLNNis tested through a chaotic time series prediction problem. The time series data is generated by using the Mackey–Glass equation that models the white blood cell production in leukemia patients [26]. The model is described by

$$\frac{dx}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - .1x(t)$$

The prediction of future values of this time series is studied in [27] by comparing with several other approaches. To make a comparison, we prepare the data in the same way as [27]. The equation is integrated by the fourth-order Runge–Kutta method.The time step used in the method is 0.1, initial condition (for in the integration), and delay term

. The time series values are stored at integer points. We extracted 1000 input–output data pairs of the following form:

$$x^d = \{[x(k-18) \quad x(k-12) \quad x(k-6) \quad x(k)\ ], \quad y^d = [x(k+6)]\}$$

where 118 to 1117.We use the first 500 pairs as the training data set, while the remaining 500 pairs as the testing data set. In this example, the MFLNN has five hidden-layer neurons. The hyperbolic tangent activation functions are used in the hidden and feedback layers. The linear activation function is used in the output layer. The training and prediction performances are determined by the root-mean-squared error (rmse) criteria defined as the positive square root of the mean-squared error (mse):

$$rmse = \sqrt{\frac{1}{N}\sum_{k=1}^{N} y(k) - y^d(k))^2}$$

where is the size of data pairs in the training or testing set and represents the predictions of the MFLNN. Fig. 5 shows the rmse curves in the logarithmic scale for both training and testing data sets. It indicates that the most of the learning was done in the first 20 epochs. One should notice that the testing rmse is less than the training rmse, which is clearly explained in [27], to be due to the initial conditions having been set to zero,

where the rest of the data set having well represented. The desired and predicted values for both training data and testing data are essentially the same in Fig. 3.7. Their differences can only be seen on a finer scale by plotting the prediction error in Fig. 3.6.

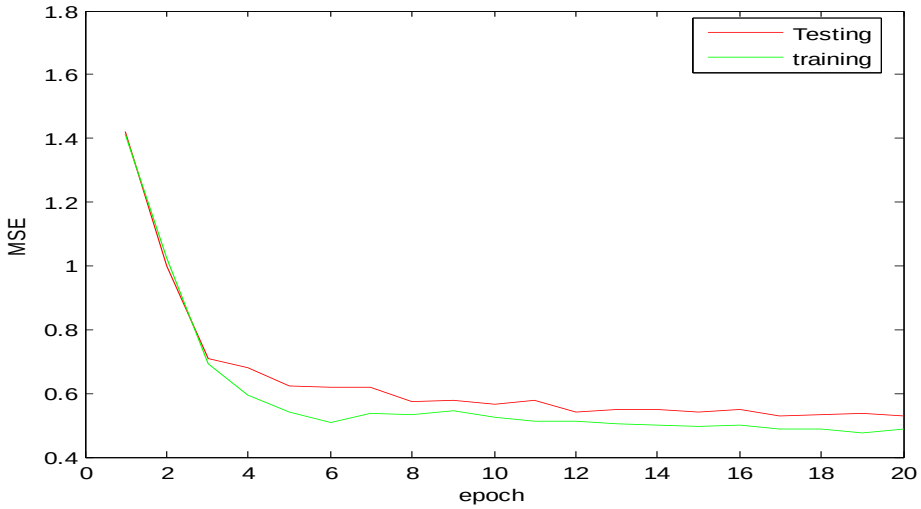


Figure 3.6 RMSE curves in the logarithmic scale

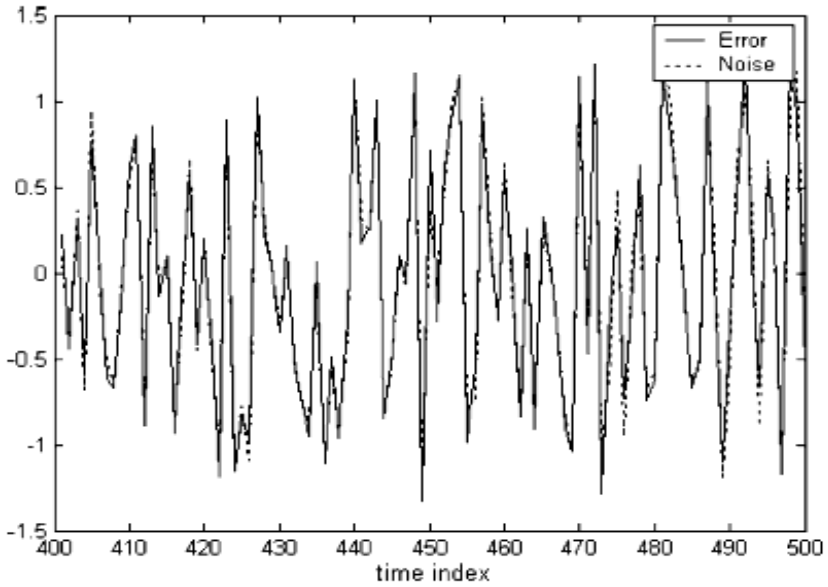


Figure 3.7 Mackey-Glass Time series values (from t=124 to 1123) and six step ahead prediction

3.5.2 Identification of a MIMO Nonlinear Plant

The method for system identification of a time invariant, causal, discrete time plant is depicted in Fig.**.** the plant is excited by a signal u(k) , and the output d(k) is measured. The plant is assumed to be stable with known parameterization but with unknown values of the parameters. The objective is to construct a suitable identification model which when subjected to the same input as the plant, produces an output which approximates in the sense described by for some desired and a suitably defined norm. The choice of the identification model and the method of adjusting its parameters based on the identification error constitute the two principal parts of the identification problem.

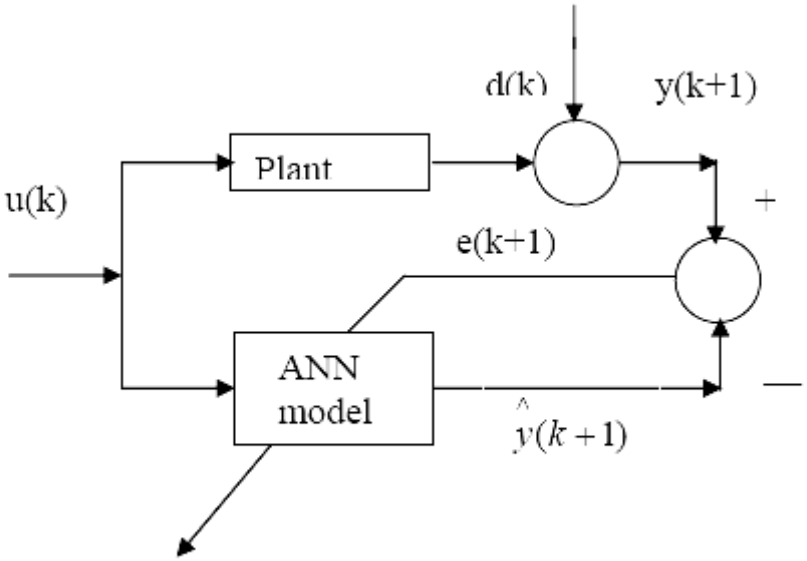


Figure 3.8 Basic Block diagram of system identification model

As a second example, the identification of a multiple input–multiple-output (MIMO) nonlinear dynamical system with two inputs and two outputs is considered to demonstrate

the structural capabilities of the MFLNN to model a certain nonlinear mapping. To make a comparison with other recurrent networks, the same plant that was used in [18] was chosen. The plant is described by the following difference equation:

$$\begin{bmatrix} y_{p1}(k) \\ y_{p2}(k) \end{bmatrix} = 0.5 \begin{bmatrix} \frac{y_{p1}(k-1)}{1+y_{p2}^2(k-1)} + u_1(k) \\ \frac{y_{p1}(k-1)\cdot y_{p2}(k-1)}{1+y_{p2}^2(k-1)} + u_2(k) \end{bmatrix}$$

Where k is the discrete time step $u_1(k)$, $u_2(k)$ and $y_{p1}(k)$, $y_{p2}(k)$ are the inputs and the outputs of the plant, respectively.In this example, the MFLNN has two inputs, two outputs, and two hidden-layer neurons. The hyperbolic tangent activation functions are used in the hidden and feedback

layers. The linear activation functions are used in the output layer. The weights are initialized by the Nguyen–Widrow method. The LM method with the trust region approach is used to update the MFLNN weights. The training data set is obtained by applying independent and identically distributed (i.i.d.) uniform sequence over [2, 2] for 500 samples and a sinusoid signal given by sin (πk/45) for the remaining 500 samples to both plant inputs. The testing data set is obtained by applying the following inputs to both plant inputs:

$$u(k)=\begin{cases}\sin(\pi k/25), & k<250\\ 1, & 250\le k<500\\ -1, & 500\le k<750\\ 0.3\sin(\pi k/25)+0.1\sin(\pi k/32) & \\ \quad +0.6\sin(\pi k/10), & 750\le k<1000\end{cases}$$

The identification performance of the MFLNN for the testing data is shown in Figures below

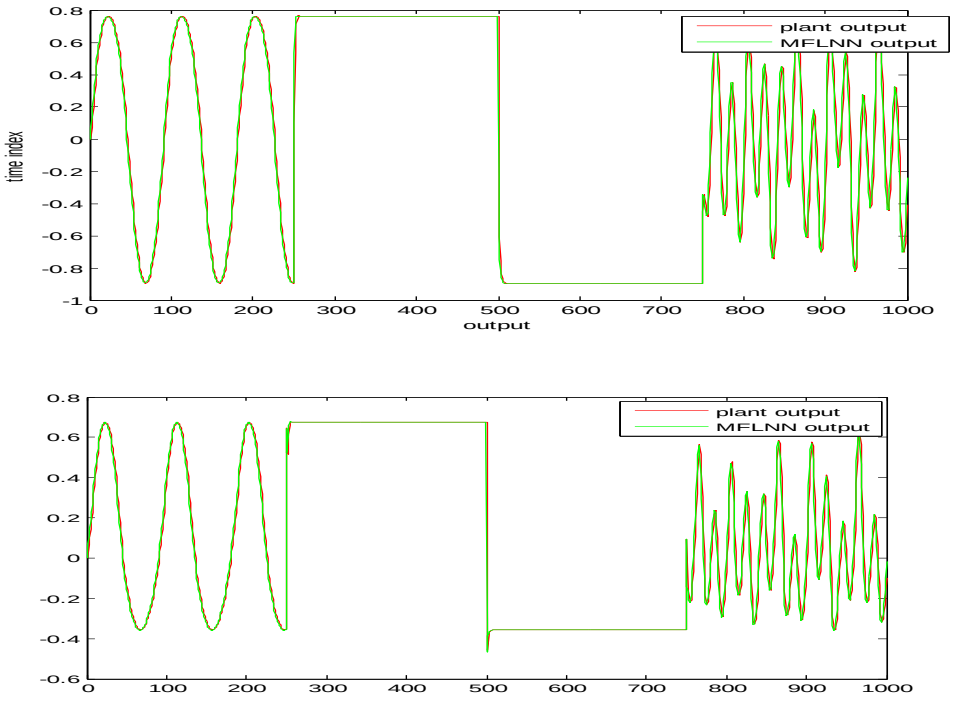


Figure 3.9 Output of plant and the MFLNN

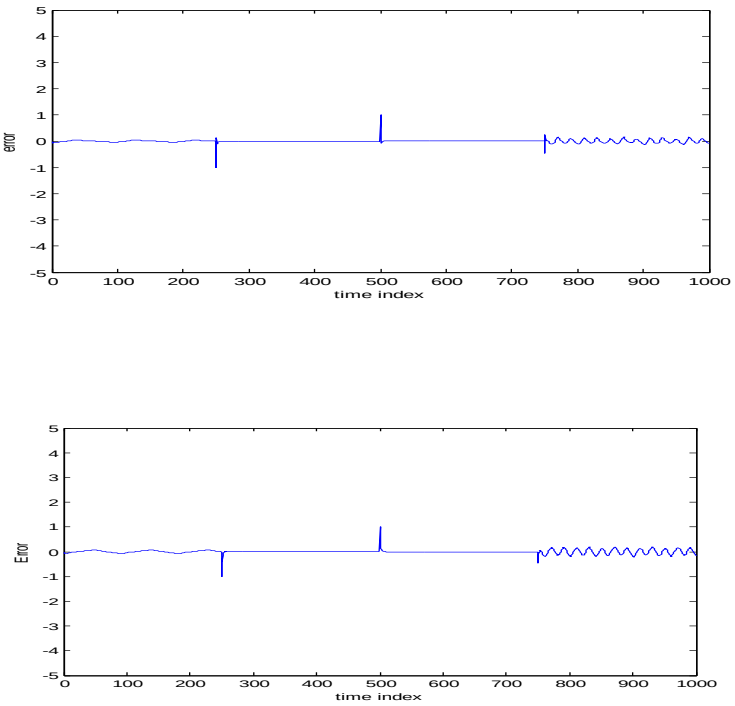


Figure 3.10 Instantaneous identification error

where we obtained that the MFLNN needs fewer training time steps and network parameters, and still achieves higher accuracy

3.5.3 Predictive Modeling of a NARMA Process

As a third example, the predictive ability of the MFLNN for a nonlinear autoregressive moving average (NARMA) process is examined and compared with the DFLS and the DFNN, reported in [17] and [23], respectively. The process is described by the following difference equation:

$$y_p(k) = v(k) + 1.2 \cdot y_p(k-1) \cdot \exp\left(-\frac{y_p^2(k-1)}{6}\right)$$

$$+0.8 \cdot v(k-1) \cdot \exp\left(-\frac{v^2(k-1)}{3}\right)$$

Where v(k) is a zero-mean uniform white noise process with standard deviation σ_V . In this example, the MFLNN has one input, one output, and two hidden-layer neurons. The hyperbolic tangent activation functions are used in all layers. The input of the MFLNN is $y_{p(k-1)}$ and its output is $y_p\hat{}(k)$. Training and testing data sets contain 1000 and 500 data pairs, respectively. Both sets are scaled into the range [1, 1]. Training lasted for 20 epochs. The whole procedure was repeated for 100 times, with the weights being initialized randomly within the interval [0.6, 0.6]. Fig. 10 shows the mse curves that correspond to σ_V = 0.7 for the last of the 100 trials. The uniform white noise v(k) and the instantaneous error (e(k)= $y_p(k)$- $y_p\hat{}(k)$) for the last 100 samples of the testing set are depicted in Fig 3. 11. One should note that v(k) and e(k) match, and the variances and the mean mse for the testing phase are almost equal, and, thus, one may conclude that the MFLNN can adequately learn the plant characteristics. In addition, the standard deviation of the error measure is smaller in the case of MFLNN, indicating the method robustness. The weight values of the MFLNN for 0.7.

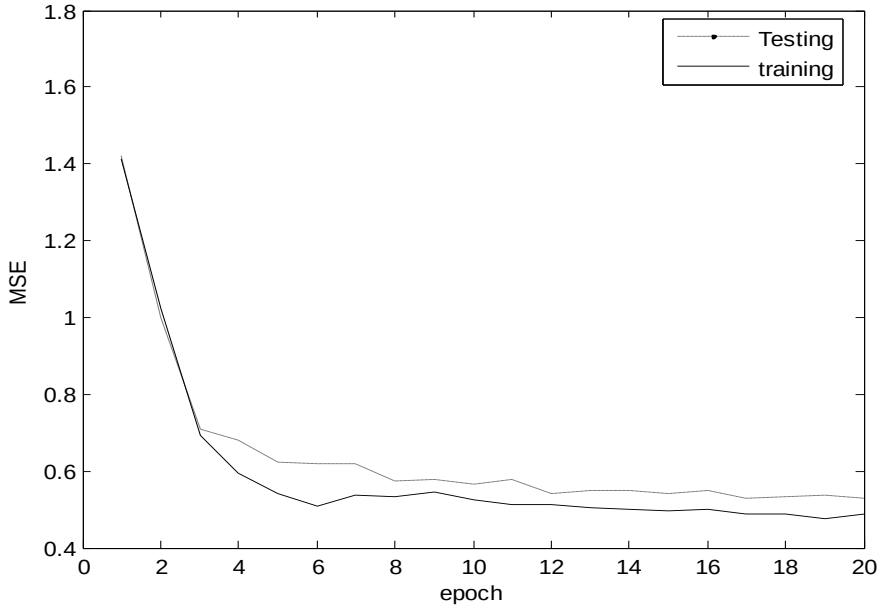


Figure 3.11 MSE curve for $\sigma_V = 0.7$

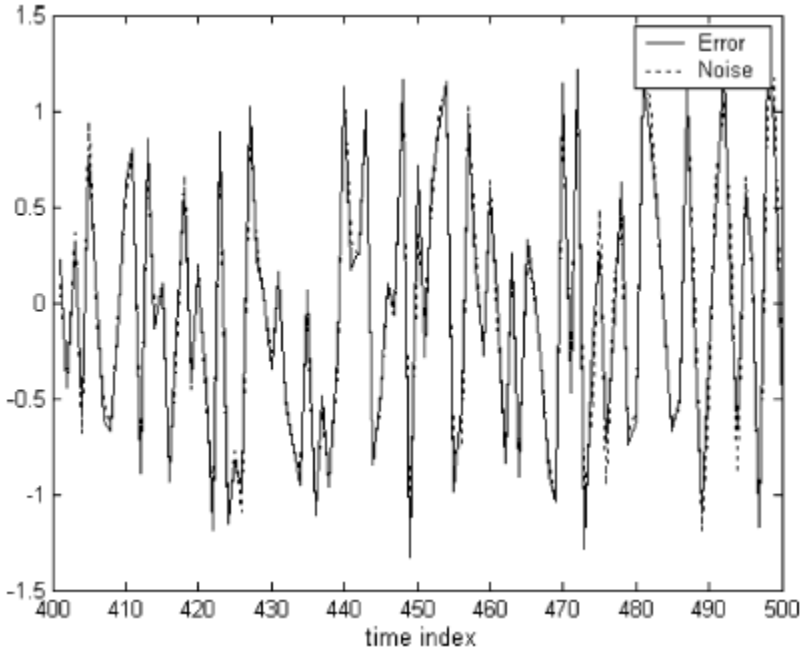


Figure 3.12 White noise V(k) $\sigma_V = 0.7$ and Instantaneous error e(k)

Chapter 4

CONCEPTS OF CHANNEL EQUALIZATION

4.1. Introduction

Recently, there has been substantial increase of demand for high speed digital data transmission effectively over physical communication channel. Communication channels are usually modeled as band-limited linear finite impulse response (FIR) filters with low pass frequency response. When the amplitude and the envelope delay response are not constant within the bandwidth of the filter, the channel distorts the transmitted signal causing intersymbol interference (ISI).B because of this linear distortion, the transmitted symbols are spread and overlapped over successive time intervals. In addition to the linear distortion, the transmitted symbols are subject to other impairments such as thermal noise, impulse noise, and nonlinear distortion arising from the modulation/demodulation process, cross-talk interference, the use of amplifiers and converters, and the nature of the channel itself. All the signal processing methods used at the receiver's end to compensate the introduced channel ver the transmitted symbols are referred as channel equalization techniques High speed communications channels are often impaired by channel inter symbol interference (ISI) and additive noise. Adaptive equalizers are required in these communication systems to obtain reliable data transmission. In adaptive equalizers the main constraint is training the equalizer. Many algorithms have been applied to train the equalizer, each having their own advantages and disadvantages. More over the importance of the channel equalization always keeps the research going on to introduce new algorithm to train the equalizer.

Adaptive channel equalization was first proposed by Lucky in 1965. One of the major drawback of the MLP structure is the long training time required for generalization and thus, this network has very poor convergence speed which is primarily due to its multilayer architecture. A single layer polynomial perceptron network(PP N) has been utilized for the purpose of channel equalization [5.3] in which the original input pattern is expanded using polynomials and cross-product terms of the pattern and then,this expanded pattern is utilized for the equalization problem. Superior performance of this network over a linear equalizer has been reported. An alternative ANN structure called functional link ANN(FLANN) originally proposed by Pao is a novel single layer ANN capable of forming arbitrarily complex decision regions. In this network,

the initial representation of pattern is enhanced by the use of nonlinear function resulting in higher dimensional pattern and hence, the separability of the patterns becomes possible. The PPN, which uses the polynomials for the expansion of the input pattern , in fact, is a subset of the broader FLANN family. Applications of the FLANN have been reported for functional approximation and for channel equalization. In the case of 2-ary PAM signal, BER and MSE performance of the FLANN-based equalizer is superior than two other ANN structures such as MLP and PPN.

4.2. Baseband Communication System

In an ideal communication channel, the received information is identical to that transmitted. However, this is not the case for real communication channels, where signal distortions take place. A channel can interfere with the transmitted data through three types of distorting effects: power degradation and fades, multi-path time dispersions and background thermal noise. Equalization is the process of recovering the data sequence from the corrupted channel samples. A typical base band transmission system is depicted in Fig.4.1., where an equalizer is incorporated within the receiver

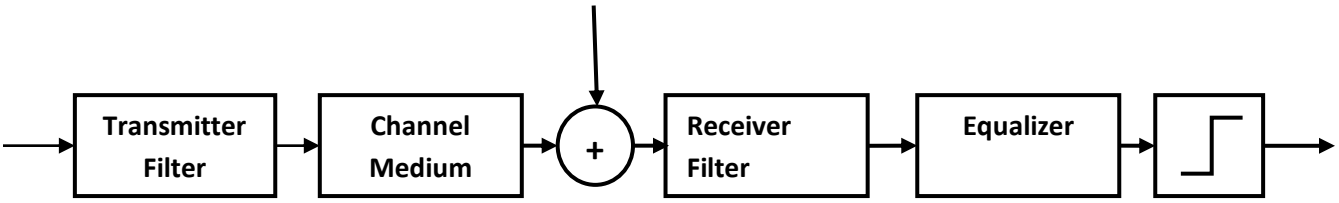


Figure 4.1 Base band communication System

4.3. Channel Interference

In a communication system data signals can either be transmitted sequentially or in parallel across a channel medium in a manner that can be recovered at the receiver. To increase the data rate within a fixed bandwidth, data compression in space and/or time is required.

4.3.1. Multipath Propagation.

Within telecommunication channels multiple paths of propagation commonly occur. In practical terms this is equivalent to transmitting the same signal through a number of separate channels, each having a different attenuation and delay. Consider an open-air radio transmission channel that has three propagation paths, as illustrated in Fig.4.2 [14].These could be direct, earth bound and sky bound. Fig.4.2 (b) describes how a receiver picks up the transmitted data. The direct signal is received first whilst the earth and sky bound are delayed. All three of the signals are attenuated with the sky path suffering the most. Multipath interference between consecutively transmitted signals will take place if one signal is received whilst the previous signal is still being detected. In Fig.4.2. this would occur if the symbol transmission rate is greater than$1/\tau$. Because bandwidth efficiency leads to high data rates, multi-path interference commonly occurs.

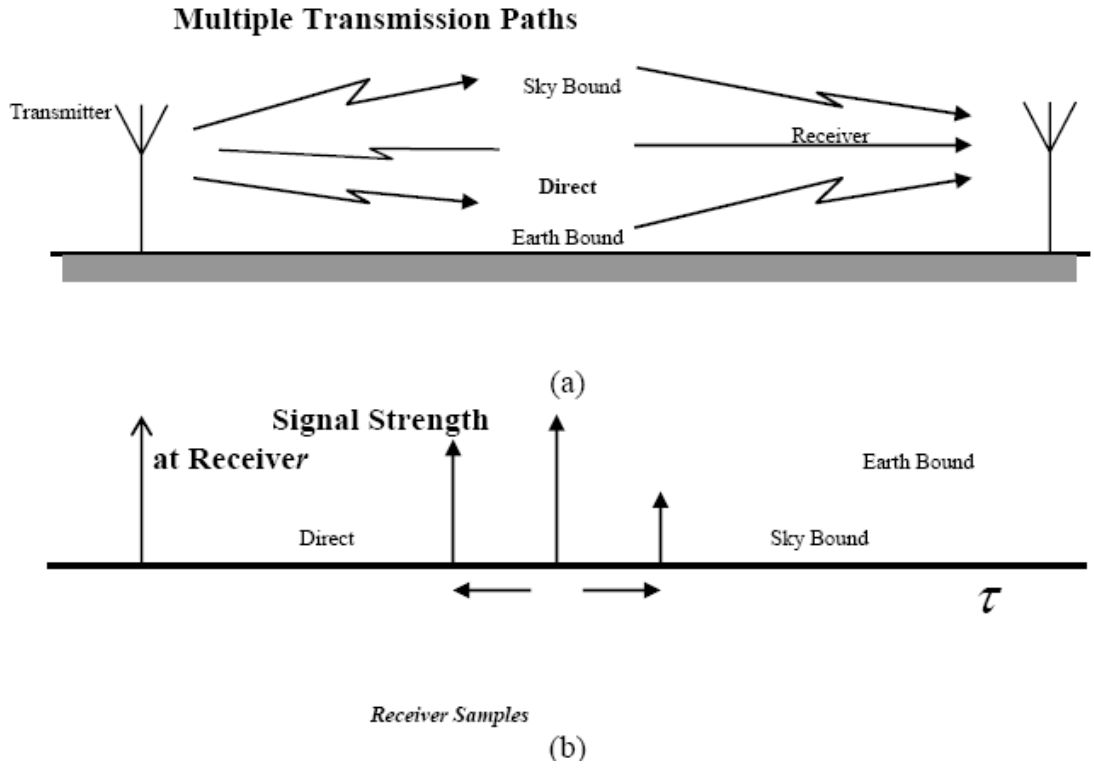


Figure 4.2 Impulse Response of a transmitted signal in a channel which has 3 modes of propagation, (a) The signal transmitted paths, (b) The received samples.

Channel models are used to describe the channel distorting effects and are given as a summation of weighted time delayed channel inputs d(n-i).

$$H(z) = \sum_{i=0}^{m} d(n-1)z^{-i} = d(n) + d(n-1)z^{-1} + d(n-2)z^{-2} + \ldots\ldots\ldots\ldots \quad (4.1)$$

The transfer function of a multi-path channel is given in Equation 5.1. The model coefficients d(n-i) describe the strength of each multipath signal.

4.4. Minimum And Nonminimum Phase Channels

When all the roots of the model z-transform lie within the unit circle, the channel is termed minimum phase . The inverse of a minimum phase channel is convergent ,illustrated by the equation

$$H(z) = 1.0 + 0.5z^{-1}$$

$$\begin{aligned} \frac{1}{H(z)} &= \frac{1}{1.0 + .5z^{-1}} \\ &= \sum_{i=0}^{\infty} \left(-\frac{1}{2}\right)^{i} z^{-i} \\ &= 1 + 0.5z^{-1} + 0.25z^{-2} - 0.125z^{-3} + \ldots\ldots\ldots \\ &= z\left[\sum_{i=0}^{\infty} \left(-\frac{1}{2}\right)^{i} z^{-i}\right] \\ &= z.[1 - 0.5z + .25z^{2} - 0.125z^{3}] \end{aligned} \quad (4.2)$$

Since equalizers are designed to invert the channel distortion process they will in effect model the channel inverse. The minimum phase channel has a linear inverse model therefore a linear equalization solution exists. However, limiting the inverse model to m-dimensions will approximate the solution and it has been shown that nonlinear solution can provide a superior inverse model in the same dimension.

A linear inverse of a non-minimum phase channel does not exist without incorporating

time delay. A time delay creates a convergent series for the non-minimum phase model, where longer delays are necessary to provide a reasonable equalizer. Equations (4.3) describe a nonminimum phase channel with a single delay inverse and a four sample delay inverse. The latter of these is the more suitable for a linear filter.

$$H(z) = 1.0 + 0.5z^{-1}$$

$$z^{-1}\frac{1}{H(z)} = \frac{1}{1.0 + .5z^{-1}} = 1 - .5z + .25z^2 - 0.125z^3 .....(noncausal) \qquad (4.3)$$

$$z^{-4}\frac{1}{H(z)} = z^{-3} - .5z^{-3} + .25z^{-1} - 0.125z + ... \text{ (truncated and causal)}$$

4.5 Intersymbol Interference

Inter-symbol interference (ISI) has already been described as the overlapping of the transmitted data. It is difficult to recover the original data from one channel sample dimension because there is no statistical information about the multipath propagation. Increasing the dimensionality of the channel output vector helps characterize the multipath propagation .this has the effect of not only increasing the number of symbol but also increase the Euclidian distance between the output classes. When additive Gaussian noise, η, is present within the channel , the input sample will form Gaussian clusters around the symbol centers. These symbol clusters can be characterized by a probability density function(pdf) with a noise variance σ_η^2 ,where the noise can cause the symbol clusters to interfere. Once this occurs, equalization filtering will become inadequate to classify all of the input samples. Error control coding schemes can be employed in such cases but these often require extra bandwidth.

4.5.1 Symbol Overlap.

The expected number of errors can be calculated by considering the amount of symbol interaction, assuming Gaussian noise . Taking any two neighboring symbols, the cumulative

distribution function(CDF) can be used to describe the overlap between the two noise characteristics.The overlap is directly related to the probability of error between the two symbols and if these two symbols belong to opposing classes,a class error will occur.

Figure 2.3 shows two Gaussian functions that could represent two symbol noise distributions. The Euclidean distance, L, between symbol canters and the noise variance σ^2, can be used in the cumulative distribution and therefore the probability of error, as in equation (5.6)

$$CDF(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi\sigma}} \exp\left[-\frac{x^2}{2\sigma^2}\right] dx \qquad (4.4)$$

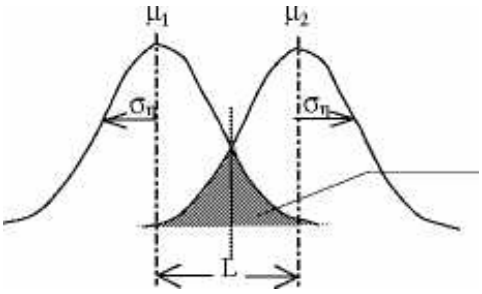


Figure4.3 Interaction between two neighboring symbols.

$$p(e) = 2CDF\left(\frac{L}{2}\right)$$

Since each channel symbol is equally likely to occure, the robability of unrecoverable errors occurring in the equalization space can be calculated using the sum of all the CDF overlap between each opposing class symbol. The probability of error is more commonly described as the BER. Equation(5.7)describes the BER based upon the Gaussian noise overlap, where N_{sp} is the number of symbols in the positive class, N_m is the distance between the i th positive symbol and its closest neighboring symbol in the negative class.

$$BER(\sigma_n) = \log\left[\frac{2}{N_{sp}+N_m}\sum_{i=1}^{Nsp} CDF\left(\frac{\Delta_i}{2\sigma_n}\right)\right]$$

4.6. Channel Equalization

High speed communications channels are often impaired by channel inter symbol interference (ISI) and additive noise. Adaptive equalizers are required in these communication systems to obtain reliable data transmission. In adaptive equalizers the mn constraint is training the equalizer. Many algorithms have been applied to train the equalizer, each having their own advantages and disadvantages. More over the importance of the channel equalizer always keeps the reaserch going on to introduce new algorithm to train the equalizer.

The optimal BER equalization performance is obtained using a maximum likelihood sequence estimator (MLSE) on the entire transmitted data sequence.A more practical MLSE would operate on smaller data sequence but these can still be computationally expensive , they also have problem tracking time-varying channels and can only produce sequence of output with a significant time delay .Another equalization approach implements a symbol-by-symbol detection procedure and is based upon adaptive filter.The symbol to symbol approachs to equalization applies the channel output samples to a decision classifier that separate the symbol into their respective classes. Traditionally these equalizers have been designed using linear filters,LTE and LDFE, with a simple FIR structure.The ideal equalizer will model the inverse of the channel model but this code doesnot take into account the effect of noise within the channel.

4.7 Summary

To compensate the ISI, Multipath channel effects on frequency response and other types of noise effects an equalizer placed at the receiver end. Since equalizer comes under inverse modeling it is difficult to design. Proper care is taken in choosing the while training the channel. LMS types equalizer performs well in case of linear channels but its performance degrades while the channel becomes nonlinear. So different nonlinear structures are being used to design nonlinear equalizer like MLP,RBF,FLANN and many more.

Chapter 5

NONLINEAR CHANNEL EQUALIZER USING ANN

5.1 Introduction

Generally speaking, message signals will inevitablysuffer from noise, interference, and power attenuation during transmission. Therefore, the receiver needs to perform some compensation for the distortion in order to get correct information . Traditional receivers usually use linear channel equalizers to solve this problem. However, when the message signals are transmitted through highly nonlinear channels, linear equalizers are no longer able to provide satisfactory results. A neural network has a fairly complicated mapping ability between input and output signals, and is therefore capable of dealing with nonlinear problems [23]. The motivation of our research is to design and implement a neural-network-based nonlinear channel equalizer under the consideration of tradeoffs between the hardware chip size, the processing speed, and the cost.The applications of neural network techniques on digital communication systems were first proposed by Siu *et al.* [24].They have made a comparison on the equalization performance between multilayer perceptron (MLP) based on the backpropagation (BP) algorithm and linear least-mean-square-based equalizer (LIN) based on least-mean-square (LMS) algorithms. According to [24], the MLP has superior performance to LIN on both bit-error-rate (BER) and mean-squared-error (MSE) characteristics, especially when message signals are transmitted through highly noisy channels. MLP, however, requires longer training time and tends to converge to undesired local minima instead of the global one. Although Zerguine has proposed a multilayer perceptron based decision feedback equalizer with lattice structure to solve the convergence problem, its high computational complexity still greatly limits the applications. Cha has used adaptive complex radial basis function (RBF) networks to deal with the channel equalization [9]. However, as the RBF network needs a large number of hidden nodes to achieve acceptable system performance, it is not quite suitable for parallel processing. The problem of huge number of hidden nodes encountered by RBF seems to be solved by using the minimum radial basis function (MRBF) neural networks proposed by Jianping [25]. However, actually in the equalization procedure of a system applying MRBF, the neural network has to first increase the number of hidden nodes, and then omits the unnecessary nodes according to rules defined in the algorithm. Since the chip size of circuit depends on the maximum number of nodes along the equalization process, the MRBF technique cannot help in simplifying the hardware design. Reference [8] indicated that the functional link artificial neural network (FLANN) presents even

better performance than MLP when techniques. After the designing procedure is finished, the circuit can be easily implemented using hardware description languages (HDLs). We choose field-programmable-gate-array (FPGA) devices for the hardware realization of our channel equalizer.

5.2 System Architecture

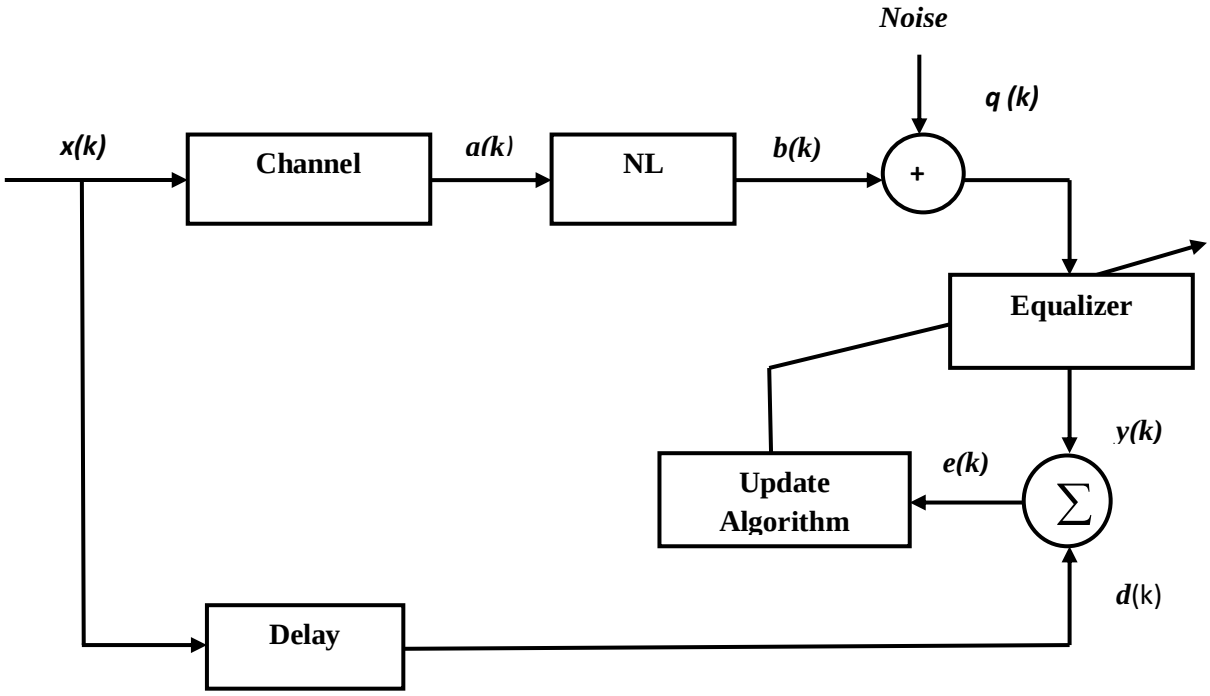
The block diagram of the digital communication system with equalizer is shown in figure below The channel is composed of the transmitting filter, the transmission medium, and other component characteristics. A commonly used linearly dispersive channel model is the so-called finite-impulse-response (FIR) model. The output *a(k)* from an FIR channel at time *kt* is

$$a(k) = \sum_{i=0}^{n_h-1} h(i)t(k-i)$$

Where *h(i)* ,*i*=0,…..n_h-1, are the tap values of the channel, and n_h is the length of the FIR chanel. The nonlinearly distorted output b(k) associated with a(k) can be written as

$$b(k) = \phi(a(k)) = \phi(t(k), t(k-1), ...t(k-n_h+1), h(0), h(1), ...h(n_h-1))$$

Where $\phi(.)$ is the nonlinear function generated by the block labeled as NL .Since the channel may also be effected by the additive white Gaussian noise(AWGN) with variance σ^2 ,the received signal at the equalizer is r(k)=b(k)+q(k), where q(k) is the white Gaussian noise(AWGN) with variance σ^2,the received signal at the equalizer is r(k)=b(k)+q(k), where q(k) is the white Gaussian noise sample at time instant *kT*. The compensated output $\hat{y}(k)$ from the equalizer is then compared with the desired signal. The error signal is defined as e(k)= y(k)-$\hat{y}(k)$, where the desired signal y(k)=t(k-d) represents the delayed version of the received signal, and D is the time delay of the signal transmitted through the physical channel. If the error e (k) is over the tolerable limit, for example, ε the parameter of the equalizer will be continued until the error function. This process will be continued until the error is under the limit value ε.



NL=Non linearity

Figure 5.1 Block diagram of channel Equalization

5.3 LIN Structure

The block diagram of an LIN structure is depicted in Fig. 2. The input signals are first passed through a bank of n delays to to form $X(k) = [x,(k), x(k-1).....x_n(k-n)]^T$ where the superscript T denotes denotes the transpose of a matrix, and the delayed signals are multiplied with a set of weights $W(k) = [w_0(k), w_1(k).....w_n(k)]$, and are then summed up with a randomly generated bias *b(k)* The result *s(k)* is the input to a linear function to obtain $\hat{y}(k)$. Without loss of generality, we will denote the linear function by purelin(.) in the followings. The error function *e(k)* is computed as the difference between $\hat{y}(k)$ and *y(k)* .

When is greater than the highest tolerable limit , the system will modify the weighting coefficients based on LMS criterion .

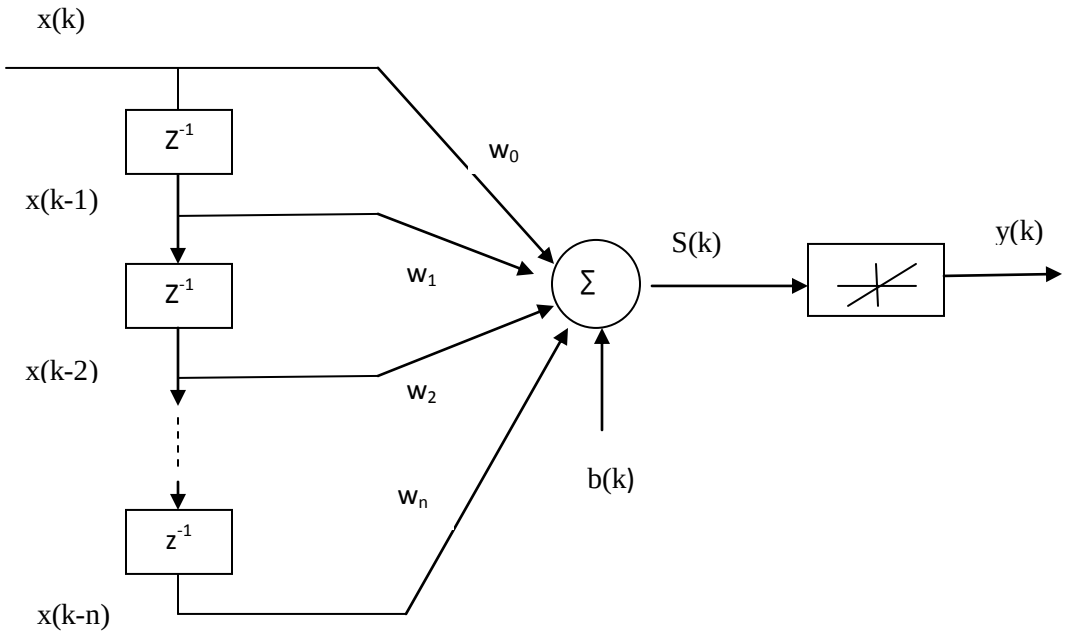


Figure5.2 LIN structure

The whole learning algorithm can ,thus, be summarized as follows:

$$S(k) = W(k)X(k) + b(k)$$

$$\hat{y}(k) = purelin(s(k))$$

$$e(k) = y(k) - \hat{y}(k)$$

$$W(k+1) = W(k) + 2\alpha e(k)X^{T}(k)$$

$$b(k+1) = b(k) + 2\alpha e(k)$$

The positive constant appearing in the above equations is the learning factor in a neural network. The numerical value of α satisfies $0<\alpha<2/\ \lambda_{max}$, where λ_{max} is the largest eigenvalue of the Hessian matrix. The initial values of *W(k)* and *b(k)* is randomly generated from an arbitrarily

selected range [-.5 ,0.5]. Although the use of LIN is generally limited on linearly separable problems, it is still quite popular due to its simplicity. For example, LIN plays a very important role in the design of adaptive filters .

5.4 FLANN

Pao originally proposed FLANN and it is a novel single layer ANN structure capable of forming arbitrarily complex decision regions by generating nonlinear decision boundaries [3.4]. Here, the initial representation of a pattern is enhanced by using nonlinear 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 separation of the patterns becomes possible in the enhanced space. The block diagram of a system with FLNN is shown in figure.

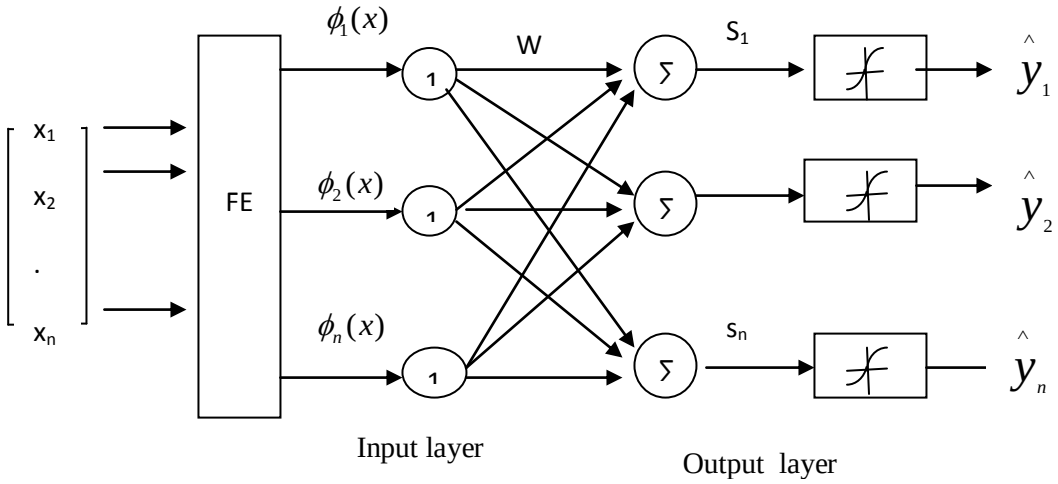


Figure 5.3 FLANN Structure

where the block labeled F.E. denotes a functional expansion. These functions map the input signal vector $X = [x_1 x_2 .....x_n]^T$ into N linearly independent functions

$\Phi = [\varphi_1(X)\,\varphi_2(X)\ \varphi_N(X)]^T$ The linear combination of these function values is presented in its matrix form, that is, $S = W\Phi$ where $S = [s_1 s_2 .....S_m]^T$, and W is the m x N dimensional weighting matrix. The matrix S is fed into a bank of identical nonlinear functions to generate the equalized output $\hat{Y} = [\hat{y}_1\ \hat{y}_2 ... \hat{y}_m]^T$, where $\hat{y}_j = \rho(s_j)$, j=1,2,……m. Here the nonlinear function is defined as $\rho(.) = \tanh(.)$.The major difference between the hardware structures of MLP and FLANN is that FLANN has only input and output layers, and the hidden layers are completely replaced by the nonlinear mappings. In fact, the task performed by the hidden layers in MLP is carried out by functional expansions in FLANN. Since the input signals are nonlinearly mapped into the output signal space, FLANN has also the ability to resolve the equalization problems for nonlinear channels. Similar to MLP, the FLANN uses the BP algorithm to train the neural networks. However,since the FLANN has much simpler structure than MLP, its speed of convergence for training process is a lot faster than MLP. The whole learning algorithm for the FLANN is summarized as follows :

$$\hat{y}_j(k) = \rho\left(\sum_{i=1}^{N} W_{ji}(k)\varphi(X_k)\right)$$

$$= \rho\left(W_j(k)\Phi(X_k)\right),$$

$$X_k = [x_1(k)x_2(k).....x_n(k)]^T,$$

$$W_j(k) = [w_{j1}(k)w_{j2}(k).....w_{jN}(k)],$$

$$\Phi(X_k) = [\varphi_1(X)\,\varphi_2(X)\ \varphi_N(X)]^T$$

$$\Delta(k) = \delta(k)(\Phi(X_k))^T$$

$$W(k+1) = W(k) + \mu\Delta(k) + \gamma\Delta(k-1)$$

$$\delta(k) = [\delta_1(k)\delta_2(k).....\delta_m(k)]^T$$

$$\delta_j(k) = \left(1 - \hat{y}_j^{\,2}(k)\right)e_j(k),$$

$$e_j(k) = y_j(k) - \hat{y}_j(k)$$

$$W(k) = [W_1(k)W_2(k).....W_m(k)]^T$$

In the above equations, μ is the learning factor and γ is the momentum factor that helps to accelerate the speed of convergence of algorithms. The values of these parameters are chosen according the inequalities $0.1 < \mu < 1.0$ and $0 < \gamma < 0.9$.

Number of operation	LIN	FLANN
Addition	$2n_0n_L+n_L$	$3(3n_0{}^+ + {}_{n0+}C_2)n_L+4n_L$
Multiplication	$3n_0n_L$	$4(3n_0{}^+ + {}_{n0+}C_2)n_L+2\ n_L+{}_{n0+}C_2$
Tanh(.)	0	n_L
Cos(.) and sin(.)	0	$2n_0{}^+$

Table 2 Number of operation for LIN and FLANN

5.5 Design Procedure

The procedure of the equalizer during one iteration can be divided into the following four steps.

1) Compute the estimated output of the network in the forward direction.

2) Evaluate the errors between the signals from the output layer and the input layer.

3) Calculate the amount of modification for the weightings between layers.

4) Update the weighting vector for each layer.

The computational complexities of MLP, LIN, and FLANN for training the neural network during each iteration are summarized in Table I, where $n_0{}^+$ represents the number of input

signals of FLANN structure, n_0 is the number of nodes at the LIN . It can be seen from the table that FLANN requires slightly more additions and multiplications than LIN.

Now, we turn our attention to the compensating performance of the algorithms over three different channel models. The normalized responses of the channel models considered in this thesis are expressed in the form of their Z transformation as follows:

CH=1:0;

CH=2:$0.209+0.995Z^{-1}+0.209Z^{-2}$

CH=3:$0.260+0.93076Z^{-1}+0.260Z^{-2}$

CH=4:$0.340+0.903Z^{-1}+0.304Z^{-2}$

CH=5:$0.341+.876Z^{-1}+.341Z^{-2}$

Where the notation CH=1 corresponds to an ideal channel with unit impulse response that has no intersymbol interference(ISI) . Model CH stands for a finite impulse response (FIR) channel with channel lenth 3. Following are the channel models we consider four kinds of nonlinearities, namely,

NL=0; b(k)=a(k);

NL=1: b(k)=tanh(a(k));

NL=2:b(k)=a(k)+.2a2(k)-.1a3(k);

NL=3:b(k)=a(k)+.2a2(k)-.1a3(k)+.5cos(πa(k));

The nonlinear model NL=0 stands for a purely linear module, that is, there is no nonlinearity in the model. The model NL=1 corresponds to systems suffering from nonlinear distortion which is possibly caused by the saturation of amplifiers used in the transceivers. In models NL =2 and

NL =3 we assume the signals have suffered from some second-order, third-order, and/or trigonometric nonlinear distortions during transmission.

5.6 Simulation study and results:

The performance of LIN and FLANN based channel equalizer is compared for linear and nonlinear channel . To study the effect of nonlinearity on the system performance four different nonlinear channel models with the following nonlinearities has been introduced. The learning parameter μ both for LIN and FLANN structure was suitably chosen to obtain best result.

Four different channels were studied with the following transfer function:

$CH=1:0.209+0.995Z^{-1}+0.209Z^{-2}$

$CH=2:0.260+0.93076Z^{-1}+0.260Z^{-2}$

$CH=3:0.340+0.903Z^{-1}+0.304Z^{-2}$

$CH=4:0.341+.876Z^{-1}+.341Z^{-2}$

To study the effect of nonlinearity on the system performance four different nonlinear channel models with the following nonlinearities has been introduced.

NL=0; b(k)=a(k);

NL=1: b(k)=tanh(a(k));

NL=2:b(k)=a(k)+.2a2(k)-.1a3(k);

NL=3:b(k)=a(k)+.2a2(k)-.1a3(k)+.5cos(πa(k));

Simulation result for channel 3 with 20 dB noise has been simulated for different linear and non linear channel has been studied. From Fig**.** it is seen that convergence characteristics of FLANN faster converges than the LIN structure. simulation result have demonstrated that FLANN presents much better error performance than LIN especially when communication system is highly nonlinear. From BER plot it is seen that FLANN structure perform better than the LIN structure for all the linear and nonlinear channel.

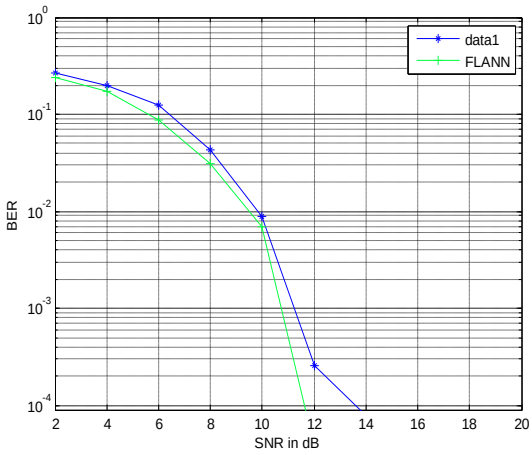


Fig 5.4(a)

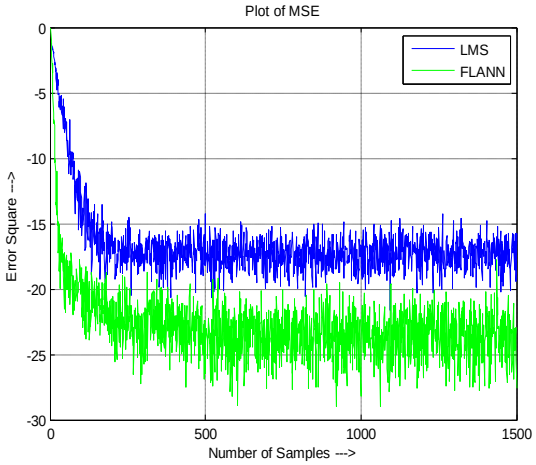


fig 5.4(b)

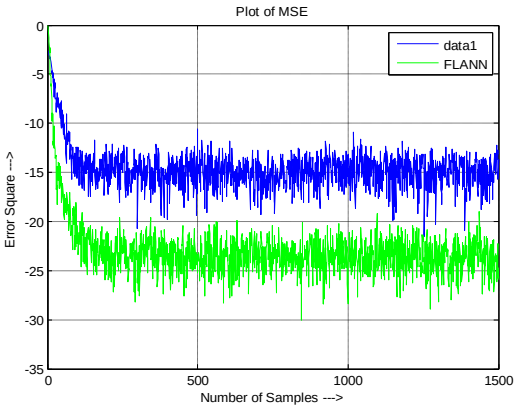


Fig 5.4(c)

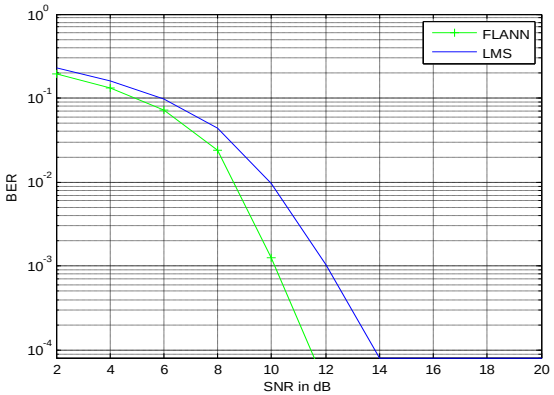


fig 5.4(d)

Fig 5.4 (a),(b) corresponds to MSE and BER plot for FLANN and LIN equalizer structure for NL=1 and fig 4.4 (c), (d) for NL=2 .Fig 4.4(e),(d) for NL=3.

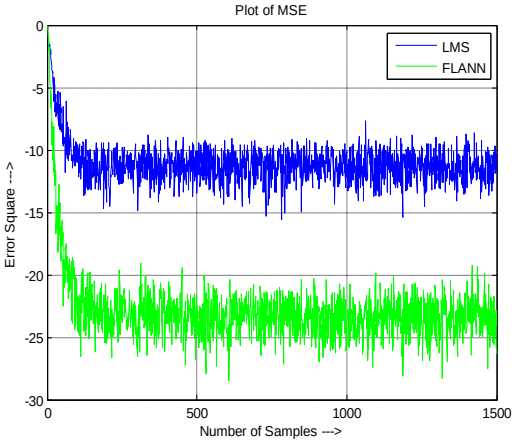


Fig 5.4(e)

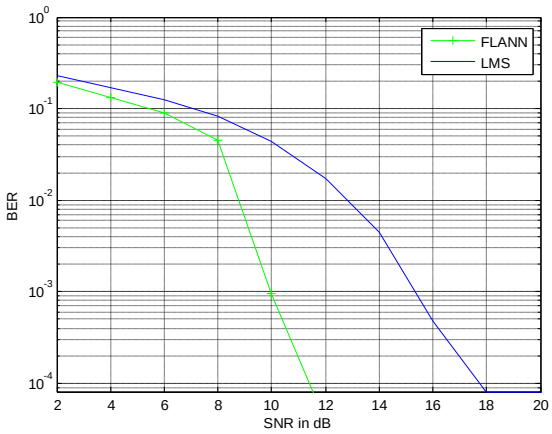


Fig 5.4(f)

5.7 Summary

Artificial neural network based nonlinear channel equalizer is described in this chapter.We compared the performance of two different structure of equalizer,namely , the linear least squared-based (LIN) and the functional link artificial neural networs(FLANN).Taking different types of nonlinearity the MSE and BER are plotted. Simulation results have demonstrated that FLANN presents much better error performance than LIN, especially when the communication channel is highly nonlinear.

Chapter 6

VLSI IMPLEMENTATION OF NONLINEAR CHANNEL EQUALIZER

6.1 Introduction

The pursuit to build intelligent human-like machines led to the birth of artificial neural networks (ANNs). Much work based on computer simulations has proved the capability of ANNs to map, model, and classify nonlinear systems. The special features of ANNs such as capability to learn from examples, adaptations, parallelism, robustness to noise, and fault tolerance have opened their application to various fields of engineering, science, economics, etc. Real-time applications are feasible only if low-cost high-speed neural computation is made viable. Implementation of neural networks (NNs) can be accomplished using either analog or digital hardware. The digital implementation is more popular as it has the advantage of higher accuracy, better repeatability, lower noise sensitivity, better testability, and higher flexibility and compatibility with other types of preprocessors. On the other hand, analog systems are more difficult to be designed and can only be feasible for large scale productions, or for very specific applications. After the designing procedure of channel equalization for nonlinear transmission environments using neural network technique is finished, the circuit is implemented using hardware description language (HDL's).We implemented using Xilinx FPGA "xcs600bg560".

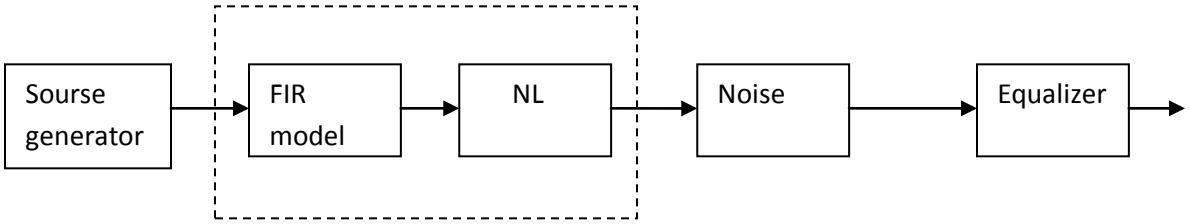


Figure6.1 Generalized module of channel with equalizer

The generalized model of communication channel with equalizer is shown in figure. In which equalizer is implemented using FLANN structure and LIN structure. The function of source generator is to generate random binary signal to the channel. The channel is modeled FIR model. In LIN structure composed of series-in-parallel-out (SIPO) shift registers and can be implemented as the tapped delay line structure depicted in Fig. 2. In addition to the simple SIPO, the FLANN structure also contains a functional expansion block, as shown in Fig. 4. The

transcendental functions and in the functional expansion are built using cordic algorithm. The mathematical operations associated with matrices, such as the computation of $S(k,)e(k)(X(k))^T$ in the LMS algorithm, and $\delta(k)(\Phi(k))^T$ in the BP algorithm, are carried out by the matrix multiplier. The output estimator for FLANN structure is a nonlinear function, which is replaced by a ROM lookup table in order to speed up the processing. The delta generator computes and according to (11). The error generator of LIN structure, however, requires only simple subtraction operations. The block diagram of the weight refresher for FLANN is given in Fig. 8.FLANN demands for two adders to refresh and Δw. While LIN requires only one adder for the purpose of refreshing W.

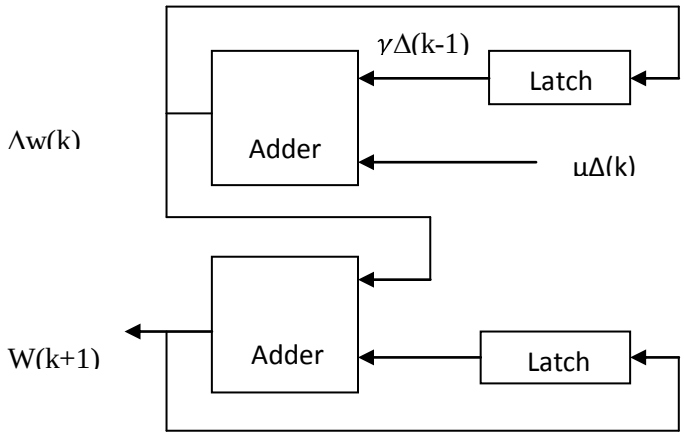


Figure 6.2 Block Diagram of FLANN weight refresher

6.2 CORDIC algorithm

In the past decade, the unprecedented advances in VLSI technology have simulated great interests in developing special purpose, parallel processor arrays to facilitate real time digital signal processing. Parallel processor arrays such systolic arrays have been extensively studied. The basic arithmetic computation of these parallel VLSI arrays has often been implemented with a multiplication and accumulation unit(MAC) ,because these operations arise frequently in DSP

applications. The reduction in hardware cost also motivated the development of more sophisticated DSP algorithms which require the evaluation of elementary functions such as trigonometric, exponential and logarithmic functions, which cannot be evaluated efficiently with MAC based arithmetic units. Consequently, when DSP algorithms incorporate these elementary functions, it is not unusual to observe significant performance degradation.

On The other hand, an alternative arithmetic computing algorithm known as CORDIC (Coordinate Rotation Digital Computer) has received renewed attention, as it offers a unified iterative formulation to efficiently evaluate each of these elementary functions. Specifically, all the evaluation tasks in CORDIC are formulated as a rotation of 2x1 vectors in various Coordinate systems. By varying a few simple parameters, the same CORDIC processor is capable of iteratively evaluating these elementary functions using the same hardware within the same amount of time. This regular unified formulation makes the CORDIC based architecture very appealing for implementation with pipelines VLSI array processors

The CORDIC is a class of hardware-efficient algorithms for the computation of trigonometric and other transcendental functions that use only shifts and adds to perform. The CORDIC set of algorithms for the computation of trigonometric functions was developed by Jack E. Volder in 1959 to help in building a real-time navigational system for the B-58 supersonic bomber. Later, J. Walther in 1971 extended the CORDIC scheme to other transcendental functions. The CORDIC method of functional computation is used by most handheld calculators (such as the ones by Texas Instruments and Hewlett-Packard) to approximate the standard transcendental functions.

Depending on the configuration defined by the user, the resulting module implements pipelined parallel, word-serial, or bit-serial architecture in one of two major modes: rotation or vectoring. In rotation mode, the CORDIC rotates a vector by a specified angle. This mode is used to convert polar coordinates to Cartesian coordinates. For example consider the multiplication of two complex numbers x+jy and $(\cos(\theta) + j \sin(\theta))$.The result u+jv, can be obtained by evaluating the final coordinate after rotating a 2x1 vector $[x\ y]^T$ through an angle θ and then scaled by a factor r.This is accomplished in CORDIC via a three-phase procedure: angle conversion, Vector rotation and scaling.

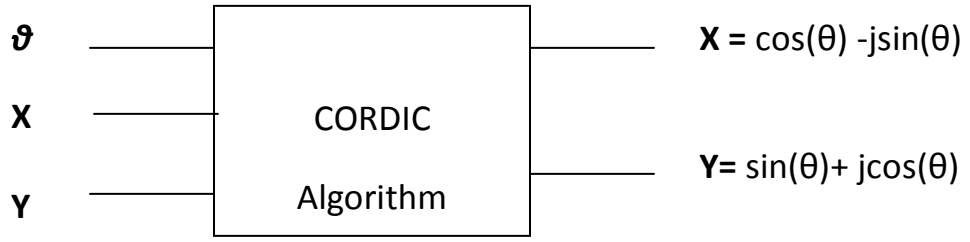


Figure 6.3 The basic block diagram of CORDIC processing

6.2.1The Rotation Transform

All the trigonometric functions can be computed or derived from functions using vector rotations. The CORDIC algorithm provides an iterative method of performing vector rotations by arbitrary angles using only shift and add operations. The algorithm is derived using the general rotation transform:

$$X' = X\cos(\phi) - Y\sin(\phi)$$

$$Y' = X\cos(\phi) + Y\sin(\phi)$$

where (X', Y') are the coordinates of the resulting vector after rotating a vector with coordinates (X,Y) through an angle of φ in the Cartesian plane. These equations can be rearranged to give:

$$X' = \cos(\phi).[X - Y\tan(\phi)]$$

$$Y' = \cos(\phi).[Y + X\sin(\phi)]$$

Now, if the angles of rotation are restricted such that $\tan(\varphi = \pm 2^{-i}$ then the tangent multiplication term is reduced to a simple shift operation. Hence arbitrary angles of rotation can be obtained by performing a series of successively smaller elementary rotations. The iterative equation for rotation can now be expressed as:

$$X_{i+1} = K_i(X_i - Y_i\sigma_i 2^{-i})$$

$$Y_{i+1} = K_i(Y_i + X_i\sigma_i 2^{-i})$$

where $K_k = cos(tan^{-1}(2^{-k}))$ and $\partial_k = \pm 1$ depending upon the previous iteration. Removing the scale constant from the above equations yields a shift-add algorithm for vector rotation. The product K_k approaches the value of 0.6073. The CORDIC algorithm in its binary version can be expressed as a set of three equations as follows:

$$X_{k+1} = [X_k - mY_k\partial_k 2^{-k}]$$

$$Y_{k+1} = [Y_k - mY_k\partial_k 2^{-k}]$$

$$Z_{k+1} = [Z_k - \partial_k\varepsilon_k]$$

Where $m = \pm 1$ and ε_k are prestored constants. The values of ε_k will become apparent from the following example for computing the sine and cosine functions.

To compute the *sin* θ and *cos* θ for $\theta \leq \pi/2$, we let $m = 1$, $\varepsilon_k = tan^{-1}(2^{-k})$ and define:

$$C = \prod_{k=0}^{n} \cos(\varepsilon_k)$$

Then the equations of the CORDIC algorithm for computing sine and cosine functions can be written down as:

$$X_{k+1} = [X_k - mY_k\partial_k 2^{-k}]$$

$$Y_{k+1} = [Y_k - mY_k\partial_k 2^{-k}]$$

$$Z_{k+1} = [Z_k - \partial_k\varepsilon_k]$$

$\delta_k = sgn(Z_k)$, $X_0 = C$, $Y_0 = 0$ and $Z_0 = \theta$ and *n* is the number of iterations performed. Then

$$X_{k+1} \approx \cos(\theta)$$

$$Y_{k+1} \approx \sin(\theta)$$

6.3 Linear Feedback Shift Register:

A Linear feedback shift register (LFSR) is a shift register that utilizes a special feedback circuit to generate the serial input value. . The feedback circuit is essentially the next-state logic. It performs xor operation on certain bits of the register and forces the register to cycle through a set of unique states In a properly designed n-bit LFSR, we can use a few xor gates to force the register to circulate through 2^n - 1 states. The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, the sequence of values produced by the register is completely determined by its current (or previous) state. Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, an LFSR with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long cycle Applications of LFSRs include generating pseudo-random number, pseudo-noise fast digital counters, and whitening sequence.

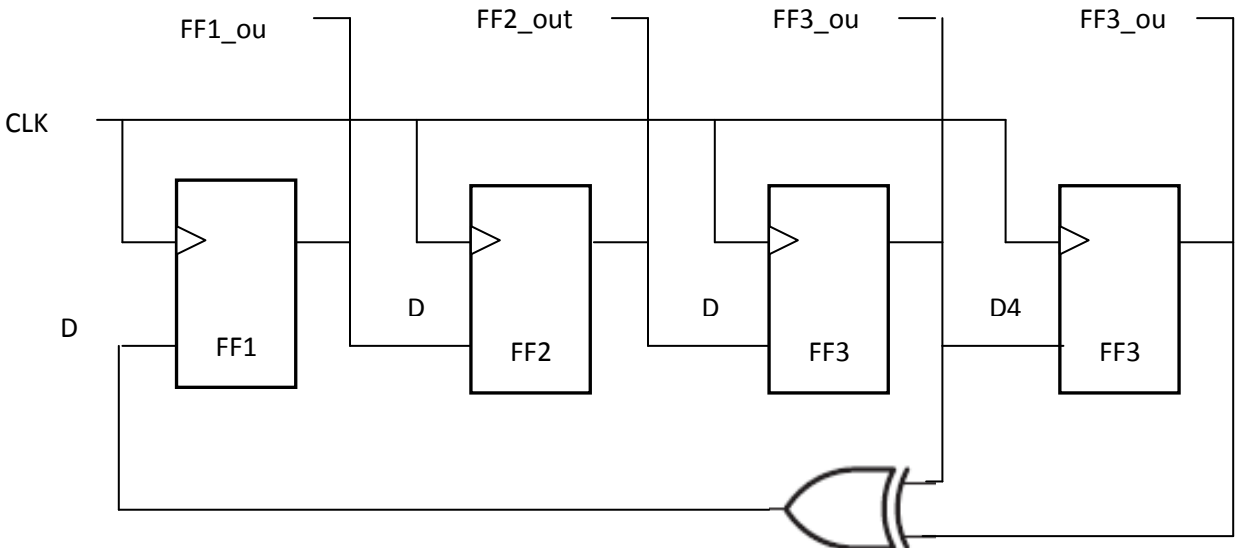


Figure 6.4 Block diagram of 4-bit LSFR

The diagram of a 4-bit LFSR is shown in Figure 9.7. The two LSB signals of the register are xored to generate a new value, which is fed back to the serial-in port of the shift register. Assume that the initial state of register is "1000". The circuit will circulate through the 15 (i.e., 24 - 1)

states as follows: "1000", "0l00", "0010", "1001", '1100", "0110", "1011", "0101", "1010", "1101", "1110","1 11 l", "01 1 l", "0011", "0001".

Note that the "0000" state is not included and constitutes the only missing state. If the LFSR enters this state accidentally, it will be stuck in this state. The construction of LFSRs is based on the theoretical study of finite fields. The term linear comes from the fact that the general feedback equation of an LFSR is described by an expression of the **and** and **xor** operators, which form a linear system in algebra. The theoretical study shows some interesting properties of LFSRs:

- An n-bit LFSR can cycle through up to $\mathbf{2^n}$ - 1 states.

- The all-zero state is excluded from the sequence.

A feedback circuit to generate maximal number of states exists for any n. The sequence generated by the feedback circuit is pseudorandom, which means that the sequence exhibits a certain statistical property and appears to be random. The feedback circuit depends on the number of bits of the LFSR and is determined on an ad hoc basis. Despite its irregular pattern, the feedback expressions are very simple, involving either one or three xor operators most of the time. Table 9.1 lists the feedback expressions for register sizes between 2 and 8 as well as several larger values. We assume that the output of the n-bit shift register is *qn-1 qn-2, . . . ,* q1 ,q0 **.** The result of the feedback expression is to be connected to the serial-in port of the shift register (Le., the input of the (n - 1)th FF).Once we know the feedback expression, the coding of LFSR is straightforward. Note that the LFSR cannot be initialized with the all-zero pattern. In pseudo number generation, the initial value of the sequence is known as a ***seed.*** We use **a** constant to define the initial value and load it into the LFSR during system initialization.

Register size	Feed back expression
2	$q_1 \oplus q_0$
3	$q_1 \oplus q_0$
4	$q_1 \oplus q_0$
5	$q_2 \oplus q_0$
6	$q_1 \oplus q_0$
7	$q_3 \oplus q_0$
8	$q_4 \oplus q_3 \oplus q_2 \oplus q_0$
16	$q_5 \oplus q_4 \oplus q_3 \oplus q_0$
32	$q_{22} \oplus q_2 \oplus q_1 \oplus q_0$
64	$q_4 \oplus q_3 \oplus q_1 \oplus q_0$
128	$q_{29} \oplus q_{17} \oplus q_2 \oplus q_0$

Table 3 Feedback expression for LSFR

6.4 Design of LUT

The implementation of the excitation function in FPGA is done using the LUT that enables to use the inbuilt RAM available in FPGA integrated chip (IC). The use of LUTs reduces the resource requirement and improves the speed. In addition, the implementation of LUT needs no external RAM since the inbuilt memory is sufficient to implement the excitation function. As the excitation function is highly nonlinear a general procedure adopted to obtain an LUT of minimum size for a given resolution is detailed as follows.

1) Let n be the number of bits

$$y = f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

2) Determine the range of input (x) for which the range of output (y) is between 2^{-n} and $1 - 2^{-n}$. Let x_1 and x_2 be the upper and lower limits of the input range.

Substituting $\frac{e^{x1}-e^{-x1}}{e^{x1}+e^{-x1}}1=2^{-n}$ and $\frac{e^{x2}-e^{-x2}}{e^{x2}+e^{-x2}}=1-2^{-n}$, it is found that

$$x_1=-\frac{1}{2}\ln\left[\frac{1-2^{-n}}{1+2^{-n}}\right] \quad \text{and} \quad x_2=-\frac{1}{2}\ln\left[\frac{2^{-n}}{2-2^{-n}}\right]$$

3) Determine the change in input Δx that produces change in output (Δy)equal to 2^{-n} at the point of maximum slop. For tansigmoid excitation function , the maximum slope is at x=0; The value of Δx for the output change of 2^{-n} can be obtained from

$$\Delta x=\frac{1}{4}\ln\left[\frac{(1-2^{-n})^2}{(1+2^{-n})^2}\right]$$

4) The minimum number of LUT values is given by

$$(LUT)_{\min}=\frac{x_1-x_2}{\Delta x}$$

Appropriate number of bits that can address $(LUT)_{min}$ are chosen. For the tansigmoid function with 8 bit resolution, x_1, x_2 and Δx are calculated from (7) and (8) .It is found from (8) that $(LUT)_{min}$=799. In order to accommodate for $(LUT)_{min}$, 10 bit address is required. Hence , 1k RAM is used as LUT for tansigmoid excitation function. The 1K RAM 1024 divisions; each of step size Δx =.0039 can accommodate the value of x in the range -3.99 to + 3.99. Here we are using symmetry property of the tansigmoid function. So we can calculate negative value of function. With this method of LUT design, the nonlinearity of the excitation function is maintained for a given 8-bit resolution. Thus, a LUT of 1-K RAM for log-sigmoid excitation function with 8-bit resolution replaces the complete computation of the excitation function. However, the size of the LUT increases for higher resolution.

6.5 VHDL simulation Results

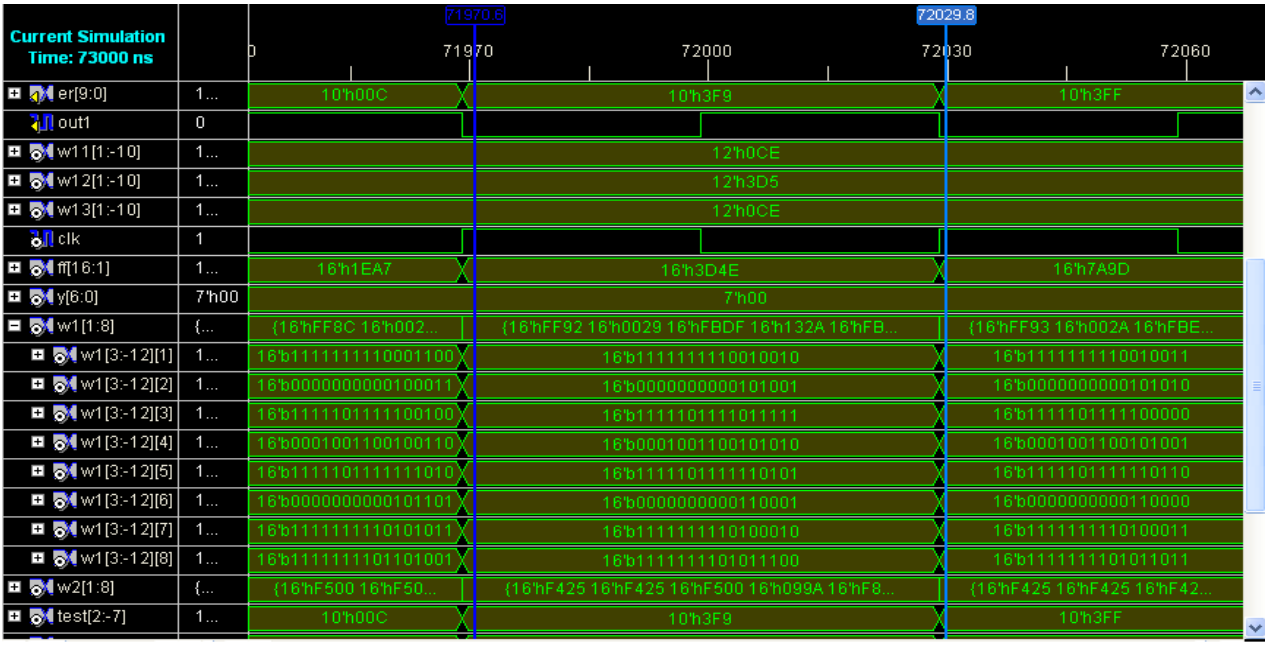


Fig 6.5 (a) Output waveform of weight updating for FLANN structure (CH=4:0.341+.876Z^{-1}+.341Z^{-2} ,NL=2)

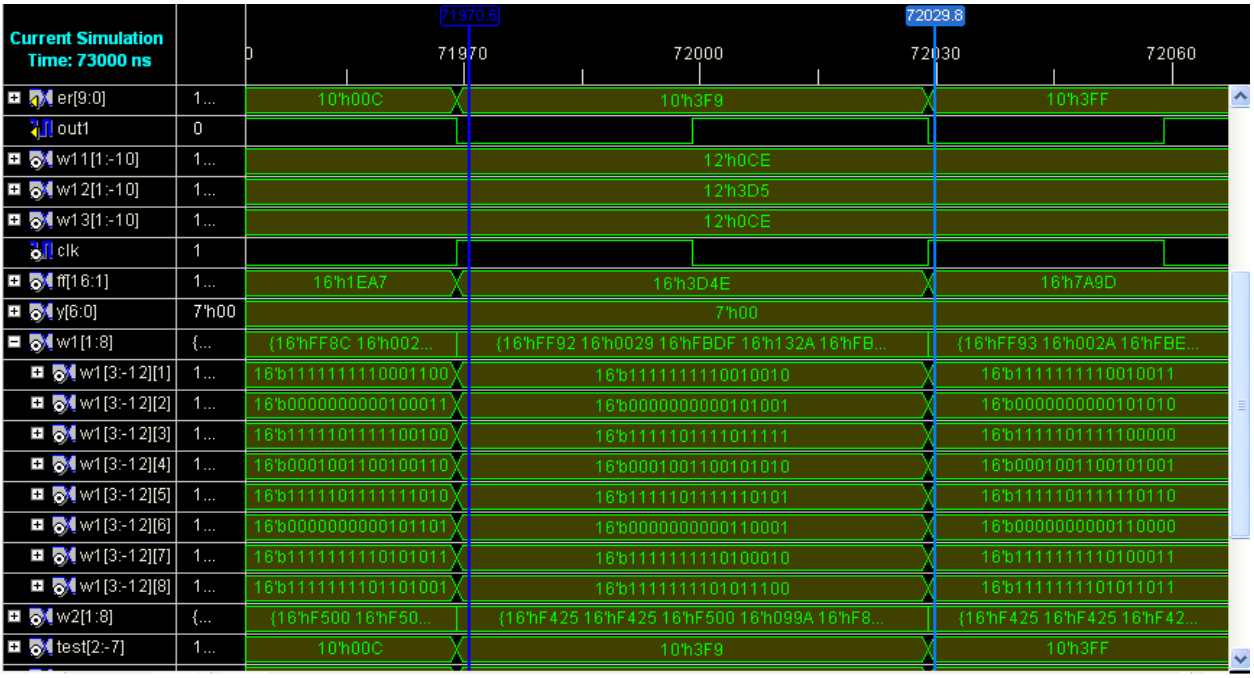


Fig 6.5(b) Output waveform of weight updating for LIN structure

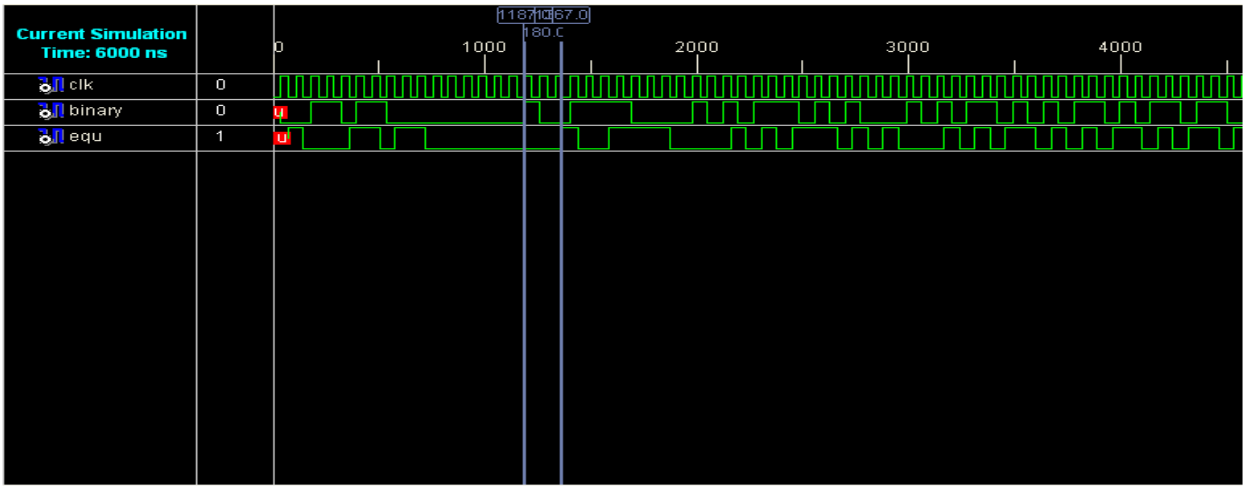


Fig 6.6 Output waveform of Nonlinear channel equalizer using FLANN Structure

(CH=4:0.341+.876Z^{-1}+.341Z^{-2} ,NL=2)

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	608	13,824	4%	
Number of 4 input LUTs	3,990	13,824	28%	
Logic Distribution				
Number of occupied Slices	2,390	6,912	34%	
Number of Slices containing only related logic	2,390	2,390	100%	
Number of Slices containing unrelated logic	0	2,390	0%	
Total Number of 4 input LUTs	4,387	13,824	31%	
Number used as logic	3,990			
Number used as a route-thru	394			
Number used as Shift registers	3			
Number of bonded IOBs	3	404	1%	
IOB Flip Flops	3			
Number of GCLKs	2	4	50%	
Number of GCLKIOBs	1	4	25%	
Number of RPM macros	1			
Total equivalent gate count for design	50,344			
Additional JTAG gate count for IOBs	192			

Fig 6.7(a)

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	160	13,824	1%	
Number of 4 input LUTs	3,090	13,824	22%	
Logic Distribution				
Number of occupied Slices	1,906	6,912	27%	
Number of Slices containing only related logic	1,906	1,906	100%	
Number of Slices containing unrelated logic	0	1,906	0%	
Total Number of 4 input LUTs	3,486	13,824	25%	
Number used as logic	3,090			
Number used as a route-thru	393			
Number used as Shift registers	3			
Number of bonded IOBs	3	404	1%	
IOB Flip Flops	3			
Number of GCLKs	2	4	50%	
Number of GCLKIOBs	1	4	25%	
Total equivalent gate count for design	39,893			

Fig 6.7(b)

Fig 5.5 (a),(b) shows the Design summery of FLANN and LIN structure respectively

6.6 Comparison of VHDL and MATLAB simulation results

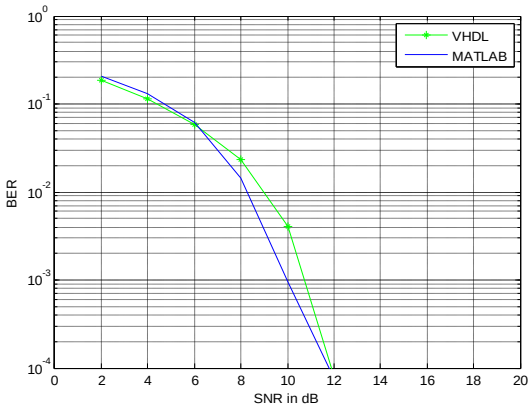


Fig 6.8(a)

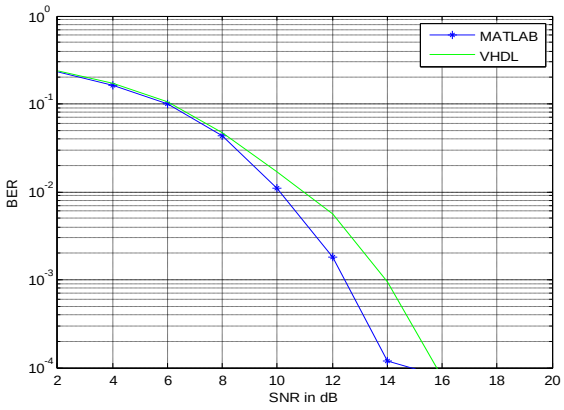


Fig 6.8(b)

Fig 5.5(a) shows the comparison VHDL and Matlab result for FLANN structure with NL=$b(k)=a(k)+.2a^2(k)-.1a^3(k)$ and (b) shows for LIN structure.

Chapter 8

CONCLUSION

The architecture and training procedure of a novel RNN, called MFLNN, have been described. The recurrences in the network structure have been introduced through the use of three feedback layers with nonlinear processing units. In these feedback layers, weighted sums of the delayed outputs of the hidden and of the output layers are passed through certain activation functions and applied to the feed forward neurons via adjustable weights. Thus, the feedback signals are processed in the feedback layers in the same way as the feed forward layers.The BPTT-like derivative calculation is required to train the recurrent systems. Since, the calculation of the derivatives by the chain rule for the unfolded structure is very complicated, the adjoint model of the MFLNN is built to simplify the computations. The fast convergence of the MLFNN weights is obtained by the LM algorithm. The performance of the MFLNN is compared with several feedforward and recurrent networks to show the structural capabilities of the network and the effectiveness of the training method. It has been shown that the proposed MFLNN achieves faster convergence rate and higher design accuracy with fewer parameters in all cases examined. The main distinguishing features of the MLFNN can be summarized as follows.

- The main difference with the available RNNs is that the temporal relations in the MLFNN are provided by means of the neurons, not by the simple feedback elements, which enrich the representation capabilities of the recurrent networks.
- It has a flexible feedback structure which enables use of different kinds of activation functions and the number of feedback neurons for different applications.
- The online training procedure based on a certain history of the patterns stored in the stack at each time step improves the adaptation performance.
- The adjoint model of the MFLNN is built to efficiently compute the derivatives for training.
- A fast convergence of the MLFNN weights is obtained by means of the LM method with the trust region approach.
- In the light of the simulation studies, we conclude that the developed MFLNN can be regarded as a new general RNN and can be effectively used for a wide class of temporal problems.

We may treat the channel equalization as a problem associated with the classification of data. The simulation results reveal that FLANN has much better performance than LIN on MSE and BER. But FLANN requires about more chip area than LIN. One way to reduce the chip area is to replace the processing architecture by serial processing. By doing so, the chip area can be tremendously saved, but the processing time will at least become doubled, which is not acceptable in most applications. Another way to decrease the required chip area is to lower the number of data bits in the decimal fractions. This provides a tradeoff between hardware cost and system performance.

REFERENCES

[1] 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.

[2] W. T. Miller, R. S. Sutton, and P. J.Werbos, *Neural Networks for Control*.Cambridge, MA: MIT Press, 1992.

[3] T. Conner, D. Martin, and L. E. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE Trans. Neural Netw.*, vol. 5, no. 2,pp. 240–253, Mar. 1994.

[4] 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.

[5] S Rajasekaran and G.A. Vijayalakshmi pai" neural net works, fuzzy logic, and genetic lgorithm"

[6] Martin .T .hagan and howard b Dcmuth " neural network design "

[7] P. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

[8] C.-F. Juang and C.-T. Lin, "A recurrent self-organizing neural fuzzy inference network," *IEEE Trans. Neural Netw.*, vol. 10, no. 4, pp. 828–845, Jul. 1999.

[9] Cha and S. Kassam, "Channel equalization using adaptive complex radial basis function networks," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 122–131, Jan. 1995.

[10] J. C. Patra, R. N. Pal, R. Baliarsingh, and G. Panda, "Nonlinear channel equalization for QAM signal constellation using artificial neural networks,"IEEE Trans. Syst., Man, Cybern. B., vol. 29, pp. 262–271, Apr. 1999

[11] C. You and D. Hong, "Adaptive equalization using the complex backpropagationalgorithm," in *Proc. IEEE Int. Conf. Neural Networks,* vol. 4, Jun. 1996, pp. 2136–2141.

[12] J. C. Patra, R. N. Pal, B. N. Chatterji, and G. Panda, "Identification of nonlinear dynamic systems using functional link artificial neural networks,"*IEEE Trans. Syst., Man, Cybern. B.*, vol. 29, pp. 254–262, Apr.1999.

[13] PONG.P.CHU ,"RTL Hardware Design Using VHDL*"*

[14] Volnei A.Pedroni. "Circuit Design With VHDL." New Delhi: Prentice-Hall of India, 2004

[15] Uwe Meyer-Baese ,"Digital Signal Processing with Field Programmable Gate Array*"*

[16] C. J. Lin and C. C. Chin, "Prediction and identification using waveletbased recurrent fuzzy neural networks," *IEEE Trans. Syst., Man, Cybern.B, Cybern.*, vol. 34, no. 5, pp. 2144–2154, ct. 2004.

[17] G. C. Mouzouris and J. M. Mendel, "Dynamic non-singleton fuzzy logic systems for nonlinear modeling," *IEEE Trans. Fuzzy Syst.*, vol.5, no. 2, pp. 199–208, May 1997.

[18] C.-F. Juang and C.-T. Lin, "A recurrent self-organizing neural fuzzy inference network," *IEEE Trans. Neural Netw.*, vol. 10, no. 4, pp. 828–845, Jul. 1999.

[19] C.-H. Lee and C.-C. Teng, "Identification and control of dynamic systems using recurrent fuzzy neural networks," *IEEE Trans. Fuzzy Syst.*,vol. 8, no. 4, pp. 349–366, Aug. 2000.

[20] C. F. Juang, "ATSK-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithms," *IEEE Trans.Fuzzy Syst.*, vol. 10, no. 2, pp. 155–170, Apr. 2002.

[21] S. F. Su and F. Y. Yang, "On the dynamical modeling with neural fuzzy networks," *IEEE Trans. Neural Netw.*, vol. 13, no. 6, pp. 1548–1553,Nov. 2002.

[22] C. J. Lin and C. C. Chin, "Prediction and identification using waveletbased recurrent fuzzy neural networks," *IEEE Trans. Syst., Man, Cybern.B, Cybern.*, vol. 34, no. 5, pp. 2144–2154, Oct. 2004.

[23] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nded. Englewood Cliffs, NJ: prentice-Hall, 1999.

[24] S. Siu, G. J. Gibson, and C. F. N. Cowan, "Decision feedback equalizationusing neural network structures and performance comparison with standard architecture," *Proc. Inst. Elect. Eng.*, pt. 1, vol. 137, pp.221–225, Aug. 1990.

[25] A. Zerguine, A. Shafi, and M. Bettayeb, "Multilayer perceptron-based DFE with lattice structure," *IEEE Trans. Neural Networks*, vol. 12, pp. 532–545, May 2001.

[26] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, pp. 287–289, 1977.

[27] 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.

[23] www.xilinx.com