# Various Nonlinear Models and their Identification, Equalization and Linearization

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

**Master of Technology**

in

**Telematics and Signal Processing**


By

**P.SUJITH KUMAR**

**ROLL No: 20607023**




**Department of Electronics and Communication Engineering**

**National Institute Of Technology**

**Rourkela**


2006-2008

# Various Nonlinear Models and their Identification, Equalization and Linearization

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

**Master of Technology**

in

**Telematics and Signal Processing**


By

**P.SUJITH KUMAR**

**ROLL No: 20607023**


Under the Guidance of

**Prof. G. Panda**



**Department of Electronics and Communication Engineering**

**National Institute Of Technology**

**Rourkela**


2006-2008


# National Institute Of Technology
# Rourkela

## CERTIFICATE

This is to certify that the thesis entitled, **"Various Nonlinear Models and their Identification, Linearization and Equalization"** submitted by **P.Sujith Kumar** in partial fulfillment of the requirements for the award of Master of Technology Degree in **Electronics & Communication Engineering** with specialization in **"Telematics and Signal Processing"** 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.

Date:

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

# ACKNOWLEDGEMENTS

This project is by far the most significant accomplishment in my life and it would be impossible without people (especially my family) 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. G.S. Rath**, **Prof. S.K. Patra, Prof. K. K. Mahapatra,** and **Prof. S. Meher** for providing a solid background for my studies and research thereafter.

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

***P.SUJITH KUMAR***

***ROLL No: 20607023***

# CONTENTS

# ABSTRACT

System identification is a pre-requisite to analysis of a dynamic system and design of an appropriate controller for improving its performance. The more accurate the mathematical model identified for a system, the more effective will be the controller designed for it. The identification of nonlinear systems is a topic which has received considerable attention over the last two decades. Generally speaking, when it is difficult to model practical systems by mathematical analysis method, system identification may be an efficient way to overcome the shortage of mechanism analysis method. The goal of the modeling is to find a simple and efficient model which is in accord with the practical system. In many cases, linear models are not suitable to present these systems and nonlinear models have to be considered. Since there are nonlinear effects in practical systems, e.g. harmonic generation, intermediation, desensitization, gain expansion and chaos, we can infer that most control systems are nonlinear. Nonlinear models are more widely used in practice, because most phenomena are nonlinear in nature. Indeed, for many dynamic systems the use of nonlinear models is often of great interest and generally characterizes adequately physical processes over their whole operating range. Thus, accuracy and performance of the control law increase significantly. Therefore, nonlinear system modeling is much more important than linear system identification. We will deal with various nonlinear models and their processing.

## THESIS LAYOUT

Identification, equalization in presence of outliers in training signal is a challenge and a very useful method is dealt in this work in chapter1 which is very robust to outliers. Volterra modeling is very useful in representing nonlinear models and many nonlinear devices needs to be linearized before use. This is dealt in chapter 2. Chapter 3 introduces two important block models namely Weiner model and Hammerstein model. These two models are very useful as most of the nonlinear devices can be represented by this model. Their identification and linearization is studied in this chapter. Chapter 4 introduces genetic algorithm, and its simultaneous use in pruning a FLANN structure and identifying parameters of a Hammerstein model with linear part represented by an IIR structure. Finally conclusions are given which were derived from the work done.

# LIST OF FIGURES

# ABBREVIATIONS USED

| | |
|---|---|
| AF | Adaptive Filter |
| WNN | Wilcoxon Neural Network |
| WFLANN | Wilcoxon Functional Link Artificial Neural Network |
| LMS | Least Mean Square |
| RLS | Recursive Least Square |
| ANN | Artificial Neural Network |
| MLP | Multi Layer Perceptron |
| FLANN | Functional Link Artificial Neural Network |
| DSP | Digital Signal Processing |
| FIR | Finite Impulse Response |
| MSE | Mean Square Error |
| NMSE | Normalised Mean Square Error |
| BER | Bit Error Rate |
| GA | Genetic Algorithm |

# Chapter 1

# WILCOXON LEARNING AND ITS USE IN MLP AND FLANN

## 1.1. Introduction

Robust and non-parametric smoothening is a central idea in statistics that aim to simultaneously estimate and model the under lying structure. One important method belonging to this category is the Wilcoxon approach, which is usually robust against outliers. Outliers are observations that are separated in some fashion from the rest of the data. Hence, outliers are data points that are not typical of the rest of the data. Depending on their location, outliers may have moderate to severe effects on the regression model. A regressor or a learning machine is said to be robust if it is not sensitive to outliers in the data.

Our motivation for robust and nonparametric regression is different from those which were previously developed. As is well known in statistics, the resulting linear regressor by using the rank-based Wilcoxon approach to linear regression problems are usually robust against (or insensitive to) outliers. It is then natural to generalize the Wilcoxon approach for linear regression problems to nonparametric Wilcoxon learning machines for nonlinear regression problems.

In the following section, two new learning machines are investigated which are very effective in dealing with various problems in presence of outliers namely Wilcoxon neural network(WNN) and Wilcoxon functional link approximation neural network(WFLANN).Then these learning algorithms will be applied to various applications like function approximation, channel equalization and system identification.

## 1.2. Wilcoxon norm.

Before investigating the Wilcoxon learning machines, we first introduce the Wilcoxon norm of a vector [22], which will be used as the objective function for all Wilcoxon learning machines. To define the Wilcoxon norm of a vector, we need a score function. A score function is a function $\phi$: [0, 1] ⟶ N which is non-decreasing such that

$$\int_0^1 \phi^2(u)du < \infty$$

The score associated with the score function $\phi$ is defined by

$$a(i) = \phi\left(\frac{i}{l+1}\right), \qquad i \in l$$

Where $l$ is a fixed positive integer.

Then Wilcoxon norm of a given vector v is given by

$$\|v\|_W := \sum_{i=1}^{l} a(R(v_i)) * v_i = \sum_{i=1}^{l} a(i) * v_{(i)} \tag{1.1}$$

$$v := [v_1 \ \ .... \ v_l]^T \in \Re^l$$

where $R(v_i)$ denotes the rank of $v_i$ among $v_1, \ldots\ldots, v_l$

$v_{(1)} \leq \ \ldots\ldots \leq v_l$ are the ordered values of $v_1, \ \ldots., v_l$

$\phi(u) := \sqrt{12} * (u - 0.5)$

Though there are other score functions available, the one presented here is the most frequently used one.

## 1.3. Wilcoxon Neural Network

The robustness of linear Wilcoxon robustness against outliers motivates us to consider the Wilcoxon neural networks (WNNs).

Consider the NN as shown in Figure. (1). There are one input layer with n + 1 nodes, one hidden layer with m + 1 nodes, and one output layer with p nodes. We also have p bias terms at the output nodes.

Let the input vector be

$x := [x_1 \ldots. x_n]^T \in \Re^n$

or

$z := [z_1 \ldots. z_n \ z_{n+1}]^T = [x_1 \ldots. x_n \ 1]^T \in \Re^{n+1}$

Let $v_{ji}$ denote the connection weight from the $i$th input node to the input of the $j$th hidden node. Then, the input $u_j$ and output $r_j$ of the $j$th hidden node are given by, respectively

$$u_j = \sum_{i=1}^{n+1} v_{ji} * z_i \qquad z_{n+1} := 1 \qquad r_j = f_{hj}(u_j) \qquad j \in \underline{m} \qquad (1.2)$$

where $f_{hj}$ is the activation function of the $j$th hidden node.

Commonly used activation functions are sigmoidal functions, i.e., monotonically increasing S-shaped functions and in this work we mainly use bipolar sigmoidal function given by

$$r_j = f_{hj}(u_j) = \frac{1 - e^{-u_j}}{1 + e^{-u_j}}$$

Let $w_{kj}$ denote the connection weight from the output of the $j$th hidden node to the input of the $k$th output node. Then, the input $s_k$ and output $t_k$ of the $k$th output node are given by, respectively

$$s_k = \sum_{j=1}^{m+1} w_{kj} * r_j \qquad r_{m+1} := 1 \qquad t_k = f_{ok}(s_k) \qquad k \in \underline{p} \qquad (1.3)$$

where $f_{ok}$ is the activation function of the $k$th output node. For classification problems, the output activation functions can be chosen as sigmoidal functions, while for regression problems, the output activation functions can be chosen as linear functions with unit slope.
The final output of the network is given by

$$y_k = t_k + b_k , \qquad k \in \underline{p}$$

where $b_k$ is the bias.

Define

$u := [u_1 \ldots . u_m]^T \in \Re^m$

$r := [r_1 \ldots . r_m \, r_{m+1}]^T \in \Re^{m+1}$

$r_{m+1} = 1$ 1.4$a$

$w_k := [w_{k1} \ldots . w_{km} \, w_{k(m+1)}]^T \in \Re^{m+1}$

$V := [v_{ji}] \in \Re^{m*(n+1)}$ 1.4$b$

$f_h(u) := [f_{h1}(u_1) \, f_{h2}(u_2) \ldots . f_{hm}(u_m) \, f_{h(m+1)}(u)]^T \in \Re^{m+1}$

$f_{h(m+1)}(u) = 1$ 1.4$c$

From (1.2) – (1.4) , we get

$$u = V * z \,, \quad r = f_h(u) \,, \quad s_k = w_k^T * r \,,$$
$$t_k = f_{ok}(s_k) \,, \quad y_k = t_k + b_k \quad k \in \underline{p} \qquad 1.5$$

Suppose we are given the training set

$S = \{(x_q, d_q)\}_{q=1}^{l}$

here subscript q is used to represent qth example.

In a WNN, the approach is to choose network weights that minimizes the Wilcoxon norm of the total residuals

The Wilcoxon norm of residuals at the $kth$ output node is given by

$\Psi_k := \|\rho_k\|_W := \sum_{q=1}^{l} a\left(R(\rho_{qk})\right) * \rho_{qk} := \sum_{q=1}^{l} a(q) * \rho_{(q)k}$ 1.6$a$

$\rho_{qk} := d_{qk} - t_{qk} \,, \quad q \in \underline{l}; \quad k \in \underline{p}$ 1.6$b$

$\rho_k := [\rho_{1k} \, \rho_{2k} \ldots . \rho_{lk}]^T \in \Re^l$

From (1.6) and (1.7)

$$\Psi_{total} := \sum_{k=1}^{p} \Psi_k$$

Thus we can minimize the total residual vector by minimizing the individual residual vector for each output.

The NN used here is the same as that used in standard ANN, except the bias terms at the outputs. The main reason is that the Wilcoxon norm is not a usual norm, but a pseudo norm (semi norm).Without the bias terms, the resulting predictive function with small Wilcoxon norm of total residuals may deviate from the true function by constant offsets.

Now, we introduce an incremental gradient–descent algorithm. In this algorithm, $\Psi_k$s are minimized in sequence. From the definition of $\Psi_k$ in (1.6a) together with (1.6b), we have

$$\Psi_k := \|\rho_k\|_W := \sum_{q=1}^{l} a(q) * [d_{(q)k} - t_{(q)k}]$$

$$:= \sum_{q=1}^{l} a(q) * \left[d_{(q)k} - f_{ok}\left(w_k^T * f_h\left(V * z_{(q)}\right)\right)\right] \qquad (1.8)$$

Updating of output weights is carried on according to the equation

$$w_k \leftarrow w_k - \eta * \frac{\partial \Psi_k}{\partial w_k}, \qquad k \in \underline{p}$$

Where $\eta > 0$ is the learning rate. From (1.8), we have

$$\frac{\partial \Psi_k}{\partial w_k} = \sum_{q=1}^{l} a(q) * (-1) * f'_{ok}\left(s_{(q)k}\right) * r_{(q)}$$

$$= -\sum_{q=1}^{l} a\left(R(\rho_{qk})\right) * f'_{ok}\left(s_{qk}\right) * r_q$$

where $f'_{ok}(.)$ denotes the total derivative of $f_{ok}(.)$ w.r.t. its arguments.

Hence, the updating rule becomes

$$w_k \longleftarrow w_k + \eta * \sum_{q=1}^{l} a\big(R(\rho_{qk})\big) * f'_{ok}\big(s_{qk}\big) * r_q$$

i.e. ,

$$w_{kj} \longleftarrow w_{kj} + \eta * \sum_{q=1}^{l} a\big(R(\rho_{qk})\big) * f'_{ok}\big(s_{qk}\big) * r_{qj}\,, \qquad j \in \underline{m+1} \qquad (1.9)$$

Updating of input weights is carried on according to the equation

$$V \longleftarrow V - \eta * \frac{\partial \Psi_k}{\partial V}$$

Now we have

$$\frac{\partial \Psi_k}{\partial V} = -\sum_{q=1}^{l} a(q) * f'_{ok}\big(s_{(q)k}\big) * \begin{bmatrix} w_{k1} * f'_{h1}(u_1) \\ w_{k2} * f'_{h2}(u_2) \\ \cdots\cdots \\ w_{km} * f'_{hm}(u_m) \end{bmatrix}_{(q)} * [z_1\ z_2\ \ldots . z_n\ z_{n+1}]_{(q)}$$

Where $f'_{hj}(.)$ denotes the total derivative of $f_{hj}(.)$ w.r.t. its arguments.

Hence, the updating rule becomes

$$v_{ji} \longleftarrow v_{ji} + \eta * \sum_{q=1}^{l} a\big(R(\rho_{qk})\big) * f'_{ok}\big(s_{qk}\big) * w_{kj} * f'_{hj}\big(u_{qj}\big) * z_{qi}\,, \qquad i \in \underline{n+1} \qquad (1.10)$$

The bias term $b_k$ , $k \in \underline{p}$ , is given by the median of the residuals at the $k$th output node, i.e.,

$$b_k = \text{med}_{1 \le q \le l}\{d_{qk} - t_{qk}\} \qquad (1.11)$$

We can write the above update equations in terms of sensitivities and can also include momentum term ($\gamma$ ) as:

$$s_2 = -a\big(R(\rho_{qk})\big) * f'_{ok}\big(s_{qk}\big)$$

$$s_1 = \ s_2 * w_{kj} * f'_{hj}\big(u_{qj}\big)$$

$$w_{kj} \longleftarrow w_{kj} - \eta * \frac{\left(\sum_{q=1}^{l} s_2 * r_{qj}\right)}{l} + \gamma * \Delta w_{kj} \qquad j \in \underline{m+1}$$

$$v \longleftarrow v_{ji} - \eta * \frac{\left(\sum_{q=1}^{l} s_1 * z_{qi}\right)}{l} + \gamma * \Delta v_{ji} \qquad i \in \underline{n+1}$$

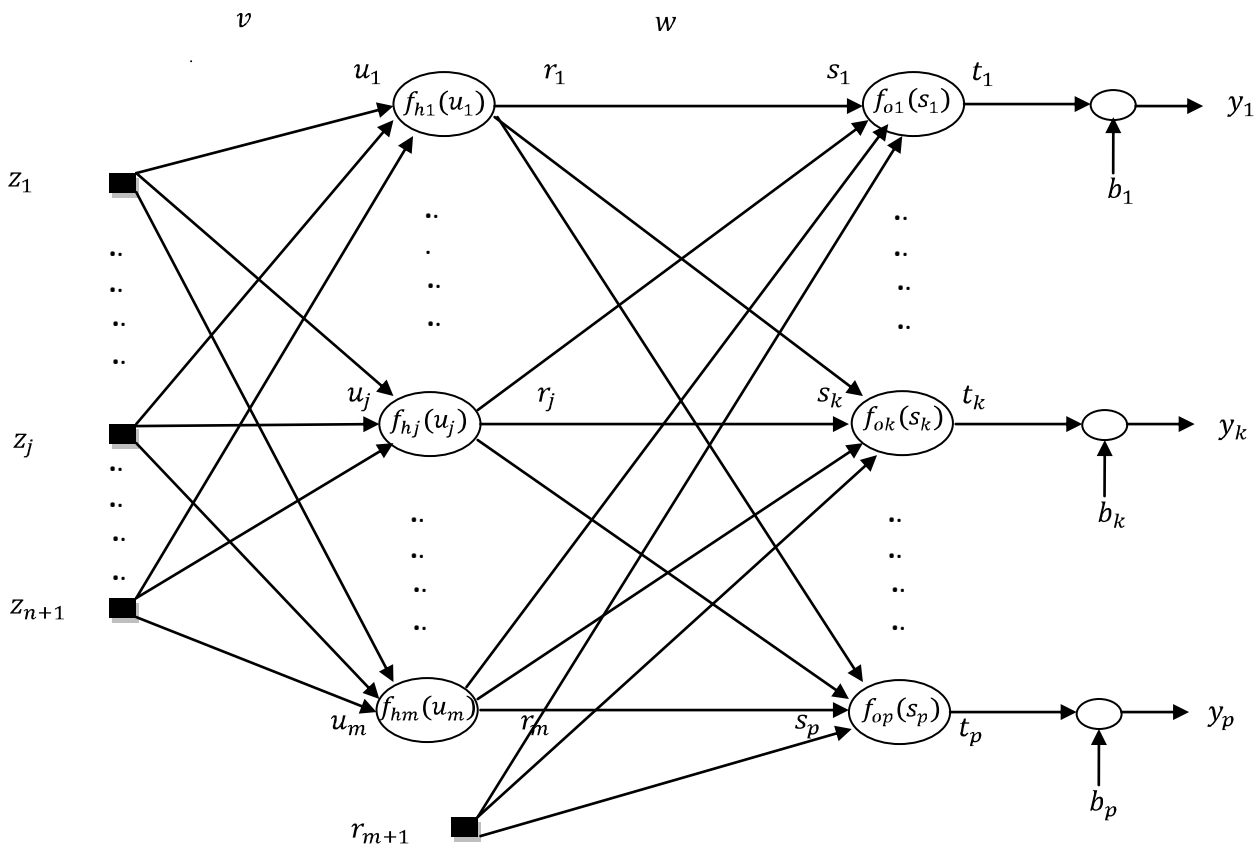



Figure 1 Wilcoxon neural network

## 1.4. Wilcoxon Functional Link Artificial Neural Network

Define

$$x := [x_1 \ \dots \ x_n]^T \ \in \ \Re^n$$

$$z := [z_1 \ \dots \ z_m]^T \ \in \ \Re^m$$

Where $x$ is the input vector which is functionally expanded using trigonometric function to get vector $z_k$ , $k \ \in \underline{p}$ ,which will then be multiplied with the corresponding weight vector and passed through an activation function to get the $k$th output.

$$u := [u_1 \ \dots \ u_p]^T$$

$$u_k := \sum_{j=1}^{m} w_{jk} * \ z_j \ , \qquad k \ \in \underline{p}$$

$$t_k := \ f_{ok}(u_k)$$

$$y_k = t_k + b$$

By using the same procedure used in WNN, we get the weight update equation as

$$w_{jk} \longleftarrow \ w_{jk} - \eta * \frac{\partial \Psi_k}{\partial w_{jk}}$$

$$:= \ w_{jk} \ + \ \eta * \sum_{q=1}^{l} a(R(\rho_{qk})) * f'_{ok}(u_k) * \ z_j \ , \qquad k \ \in \underline{p} \ , \ j \ \in \underline{m} \qquad (1.12)$$

Where $\eta > 0$ is the learning rate, and $f'_{ok}(.)$ denotes the total derivative of $f_{ok}(.)$ w.r.t. its arguments.

The bias term $b_k$ , $k \in \underline{p}$ , is given by the median of the residuals at the $k$th output node , i.e. , $b_k = \text{med}_{1 \le q \le l}\{d_{qk} - t_{qk}\}$.

We can write the above update equation in terms of sensitivities and can also include momentum term ($\gamma$ ) as:

$$s_1 = -a\big(R(\rho_{qk})\big) * f'_{ok}(u_k)$$

$$w_{jk} \leftarrow w_{jk} - \eta * \frac{\left(\sum_{q=1}^{l} s_1 * z_{jk}\right)}{l} + \gamma * \Delta w_{jk} \qquad k \in \underline{p} \ , \ j \in \underline{m}$$

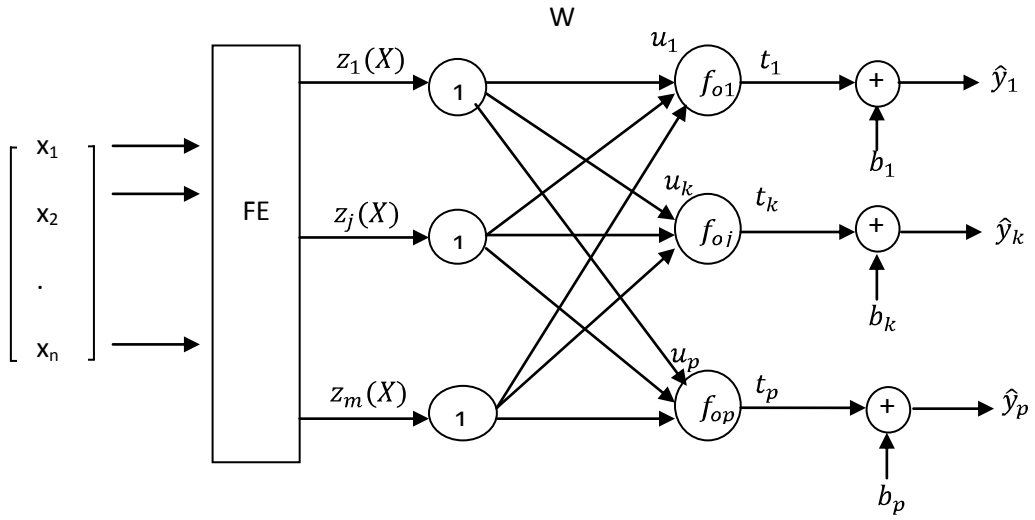



**Figure 2 Wilcoxon functional link network.**

## 1.5. Simulation & Results

In this section, we compare the performances of various learning machines for several illustrative nonlinear regression problems. Emphasis is put particularly on the robustness against outliers for various learning machines. We wish to point out that different parameter settings for learning machines might produce different results. For "fair" comparison, similar machines will use the same set of parameters in the simulation. Thus, for ANN and WNN, we use the same number of hidden nodes, the same activation functions for hidden nodes, and the output node. Similarly, for FLANN and WFLANN, we use the same expansions for both machines.

We will apply WNN and WFLANN to various applications like non-linear function approximation, system identification and channel equalization in presence of outliers and compare them with the results obtained using ANN and FLANN respectively.

### 1.5.1. Function approximation

In each simulation of Examples 1 and 2, the uncorrupted training data set consists of 50 randomly chosen $x-$ points (training patterns) with the corresponding $y-$ values (target values) evaluated from the underlying true function. The corrupted training data set is composed of the same $x-$ points as the corresponding uncorrupted one but with randomly chosen $y-$ values corrupted by adding random values from a uniform distribution defined on$[-1\,,1]$. It would be interesting to know what happens if the noise is progressively increased and if the number of outliers is increased. To this end, 20%, 30%, and 40% randomly chosen -values of the training data points will be corrupted.

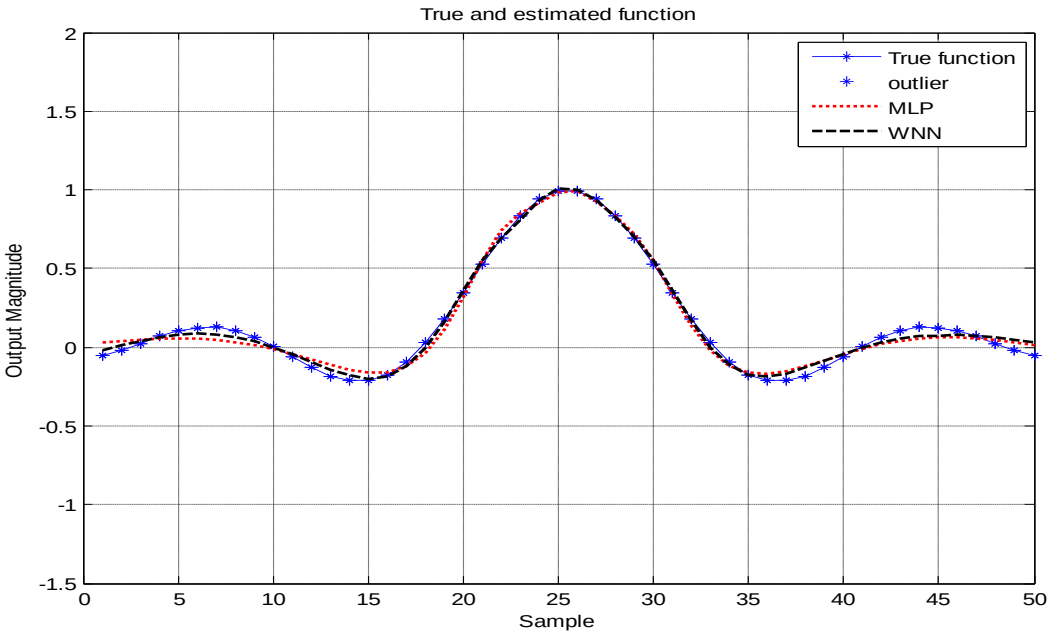
**Example 1 :**

Suppose the true function is given by the sinc function

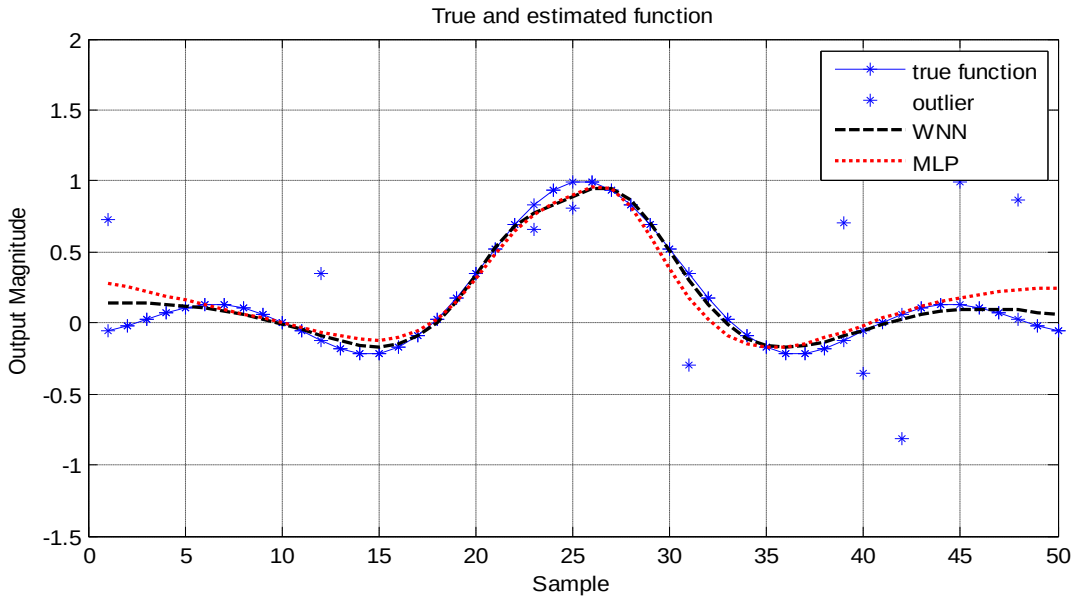
$y = \frac{\sin(x)}{x}$ , $x \in [-10, 10]$

In this example, we compare the performances of ANN, WNN, FLANN, and WFLANN. For ANN and WNN, the number of hidden nodes is 30, the activation functions of the hidden nodes are bipolar sigmoidal functions, and the activation function of the output node is a linear function with unit slope. For FLANN and WFLANN, the number of hidden nodes is 10, trigonometric expansion is used, and the activation function of the output node is a linear function with unit slope.

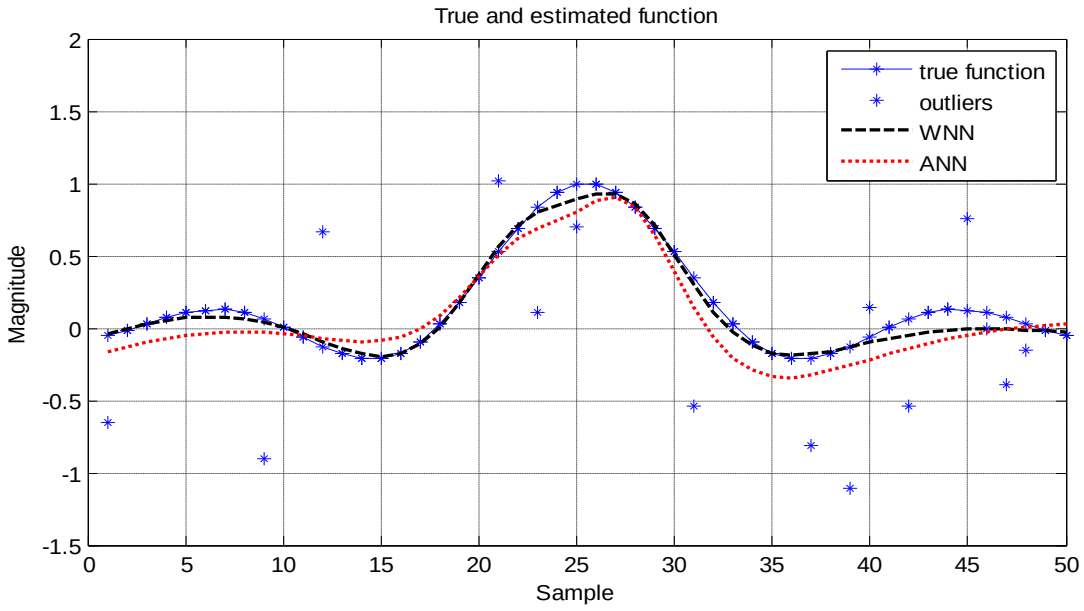
The simulation results for ANN and WNN are shown in Fig.(1). For uncorrupted data shown in Fig.( 1)( a), WNN performs better than ANN. For corrupted data shown in Fig.(1)(b) – (1)(d) with progressively increased corruption ,WNN estimates are almost unaffected by these corrupted outliers and outperforms ANN estimates



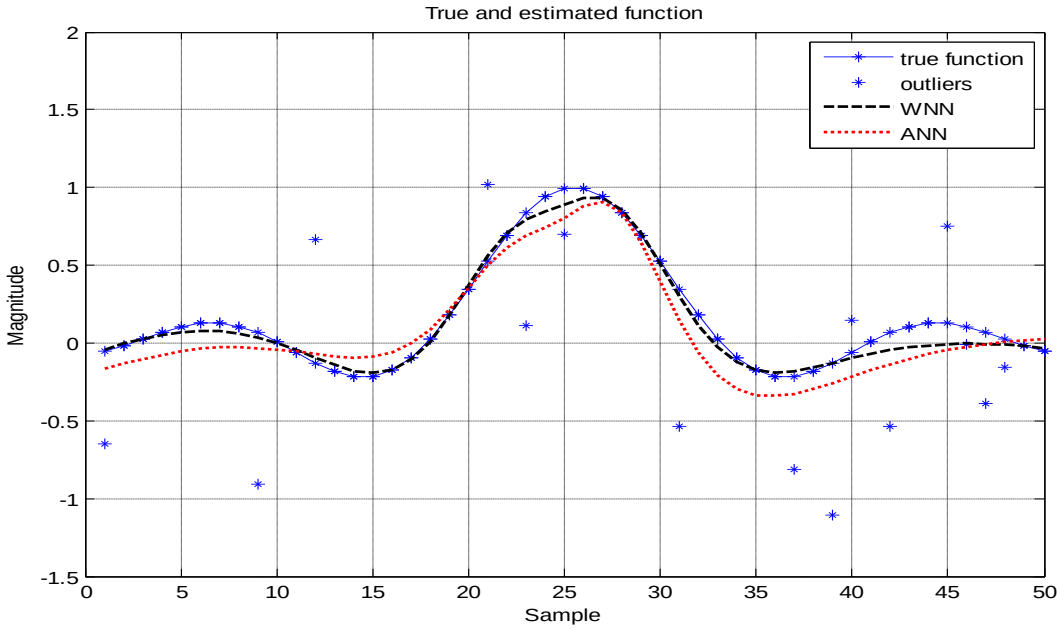

**(a)**


True and estimated function
true function
outlier
WNN
MLP
Output Magnitude
Sample
2
1.5
1
0.5
0
-0.5
-1
-1.5
0
5
10
15
20
25
30
35
40
45
50

**(b)**


True and estimated function
true function
outliers
WNN
ANN
Magnitude
Sample
2
1.5
1
0.5
0
-0.5
-1
-1.5
0
5
10
15
20
25
30
35
40
45
50

**(c)**

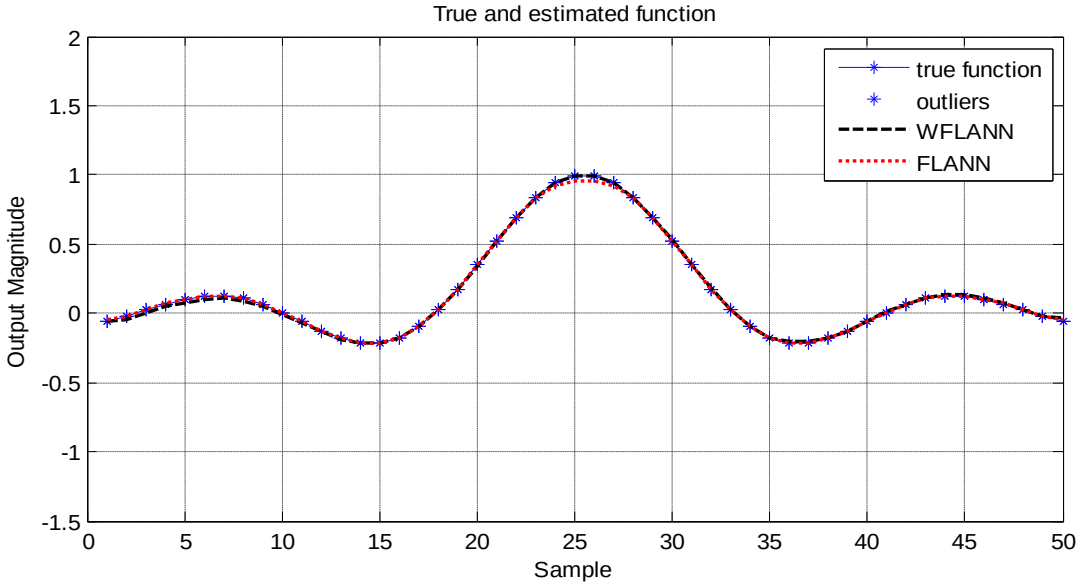



**(d)**

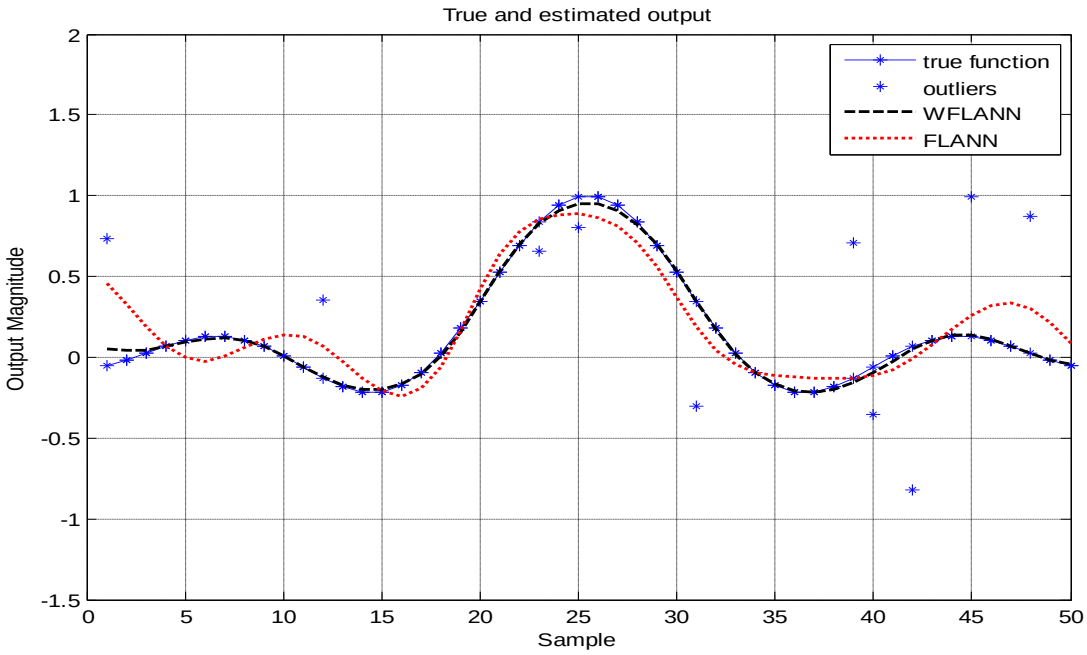
**Figure 1 Simulations for ANN and WNN of Example 1: (a) uncorrupted data, (b) 20% corrupted data**

**(c) 30% corrupted data (d) 40% corrupted data**

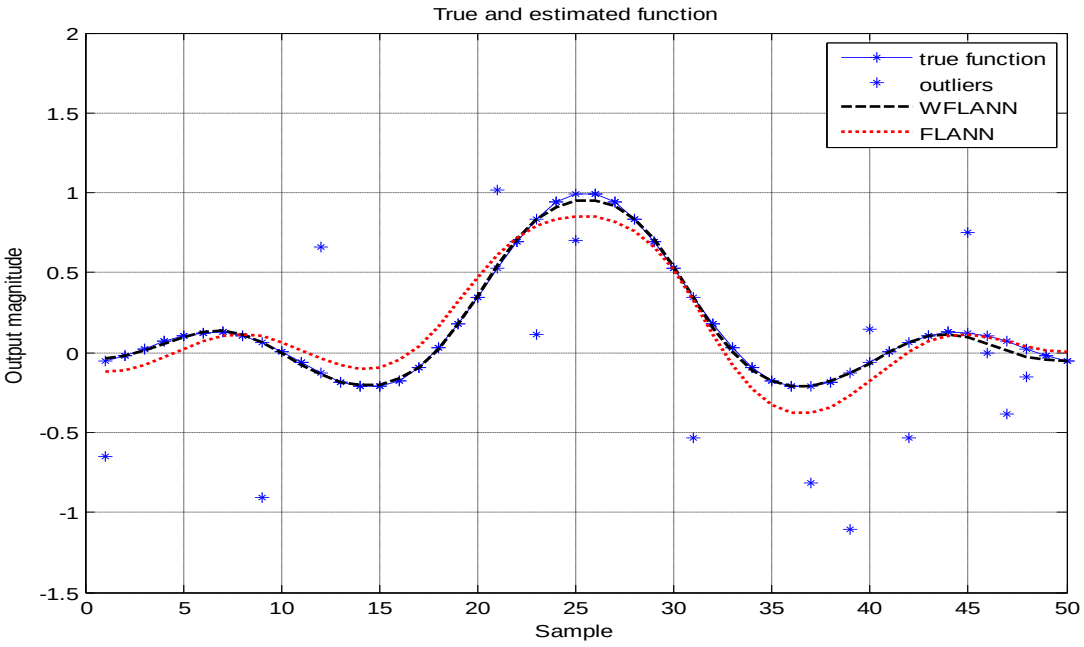
Results are shown in Fig.(2) for WFLANN and FLANN approximates.



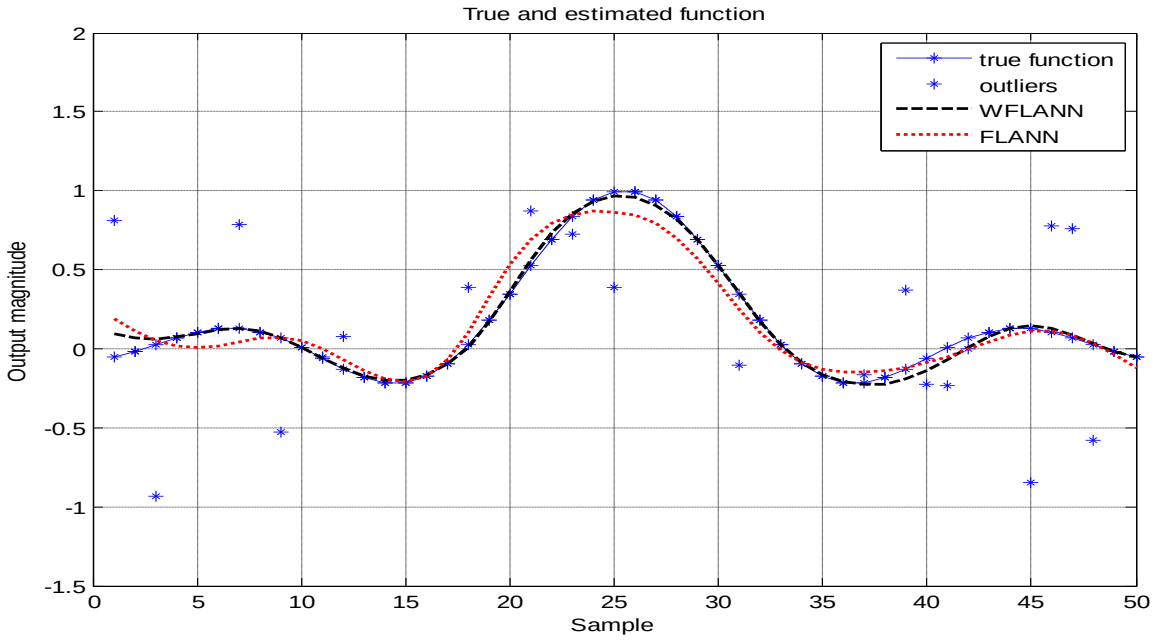

**(a)**


True and estimated output
true function
outliers
WFLANN
FLANN
Output Magnitude
Sample
2
1.5
1
0.5
0
-0.5
-1
-1.5
0
5
10
15
20
25
30
35
40
45
50

**(b)**


True and estimated function
true function
outliers
WFLANN
FLANN
Output magnitude
Sample
2
1.5
1
0.5
0
-0.5
-1
-1.5
0
5
10
15
20
25
30
35
40
45
50

**(c)**




**(d)**

**Figure 2 Simulations for FLANN and WFLANN of Example 1: (a) uncorrupted data, (b) 20% corrupted data (c) 30% corrupted data (d) 40% corrupted data**

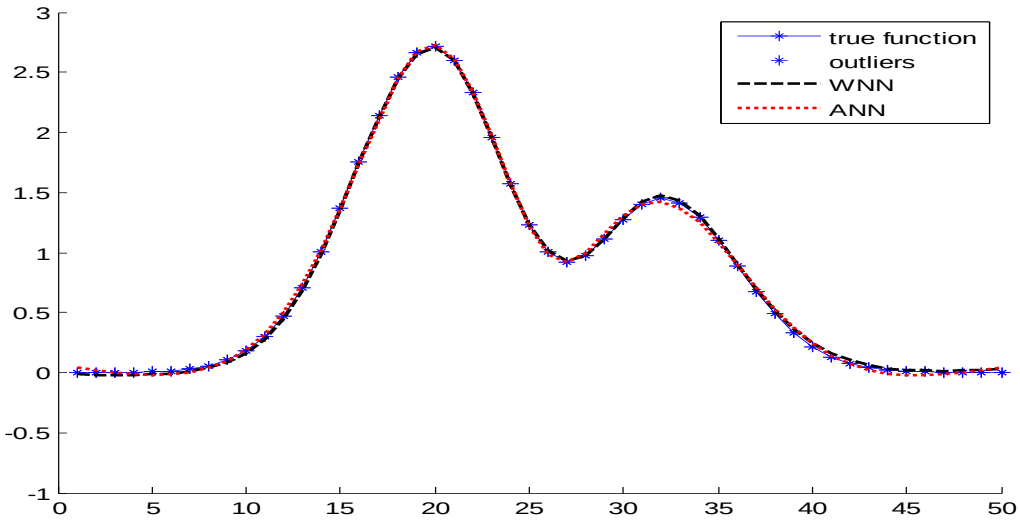
## Example 2:

Suppose the true function is given by the Hermite function

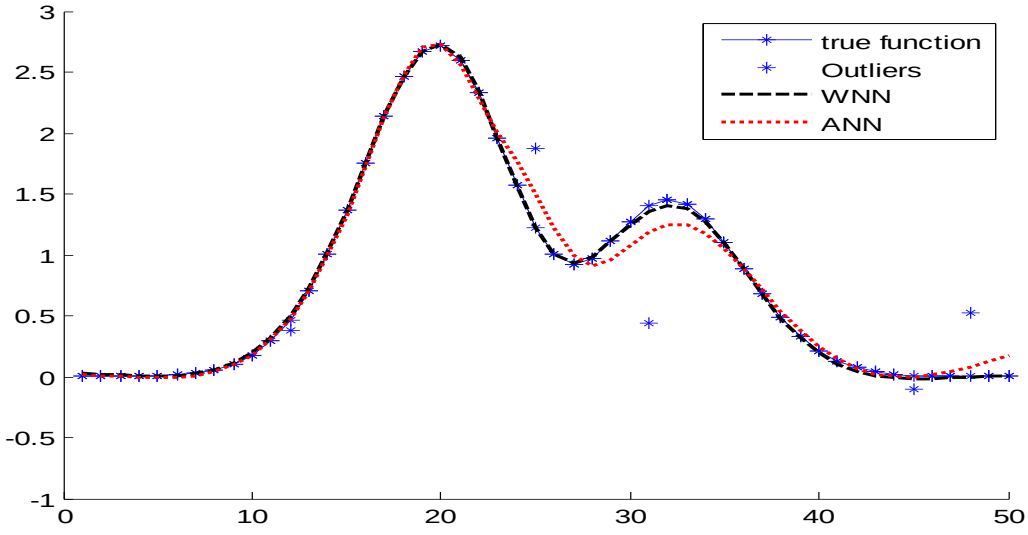
$$y = 1.1 * (1 - x + 2 * x^2) * e^{-\frac{x^2}{2}}, \qquad x \in [-5, 5]$$

In this example, we compare the performances of ANN, WNN, FLANN, and WFLANN. For ANN and WNN, the number of hidden nodes is 20, the activation functions of the hidden nodes are bipolar sigmoidal functions, and the activation function of the output node is a linear function with unit slope. For FLANN and WFLANN, the number of hidden nodes is 10, trigonometric expansion is used, and the activation function of the output node is a linear function with unit slope.

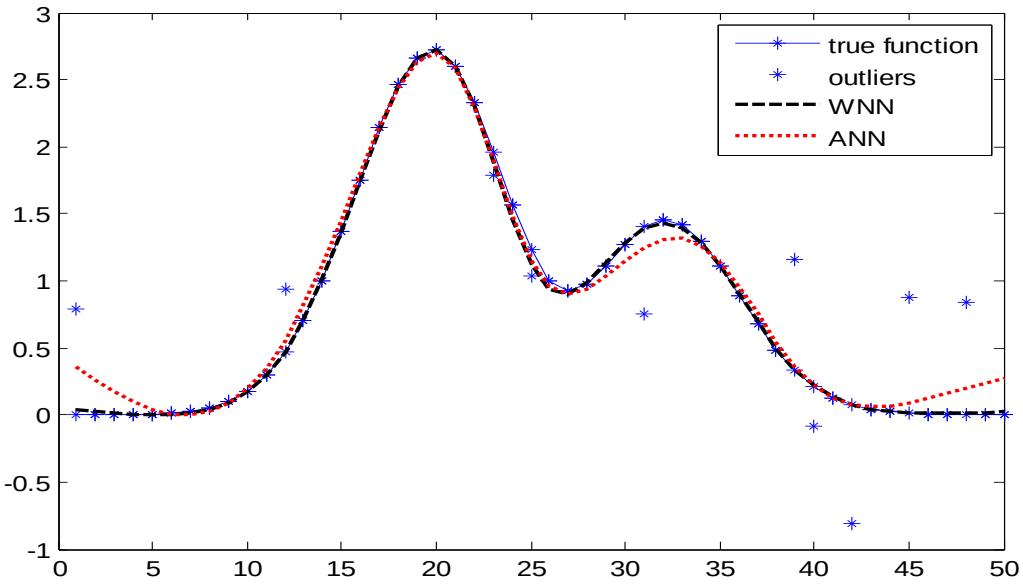
The simulation results for ANN and WNN are shown in Fig.(3). For uncorrupted data shown in Fig.(3)(a), WNN performs better than ANN.For corrupted data shown in Fig.(3)(b) – (3)(d) with progressively increased corruption ,WNN estimates are almost unaffected by these corrupted outliers and outperforms ANN estimates


true function
outliers
WNN
ANN
3
2.5
2
1.5
1
0.5
0
-0.5
-1
0
5
10
15
20
25
30
35
40
45
50

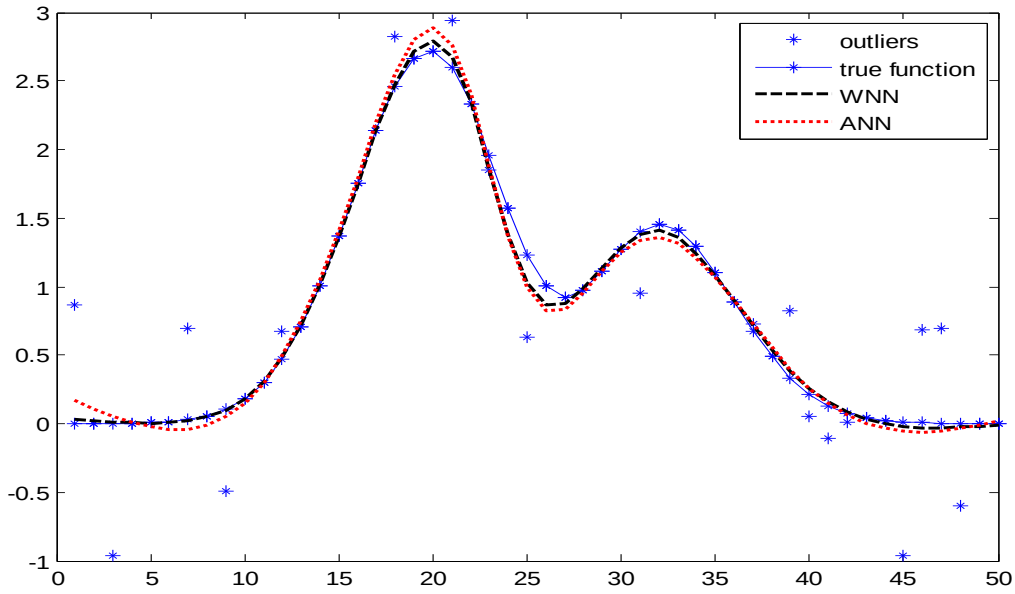
**(a)**


true function
Outliers
WNN
ANN
3
2.5
2
1.5
1
0.5
0
-0.5
-1
0
10
20
30
40
50

**(b)**



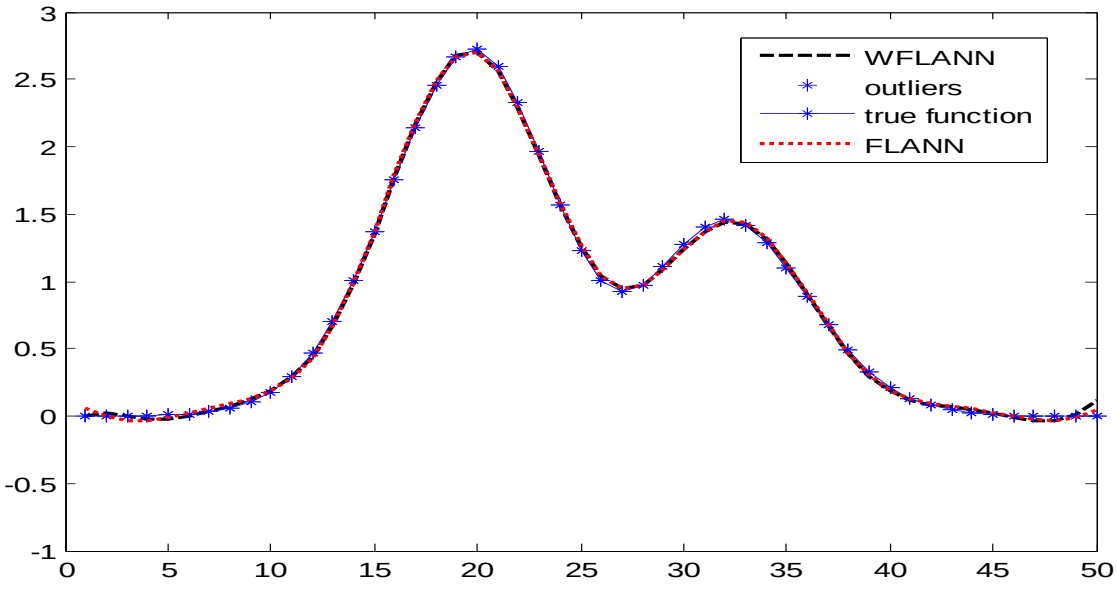

**(c)**



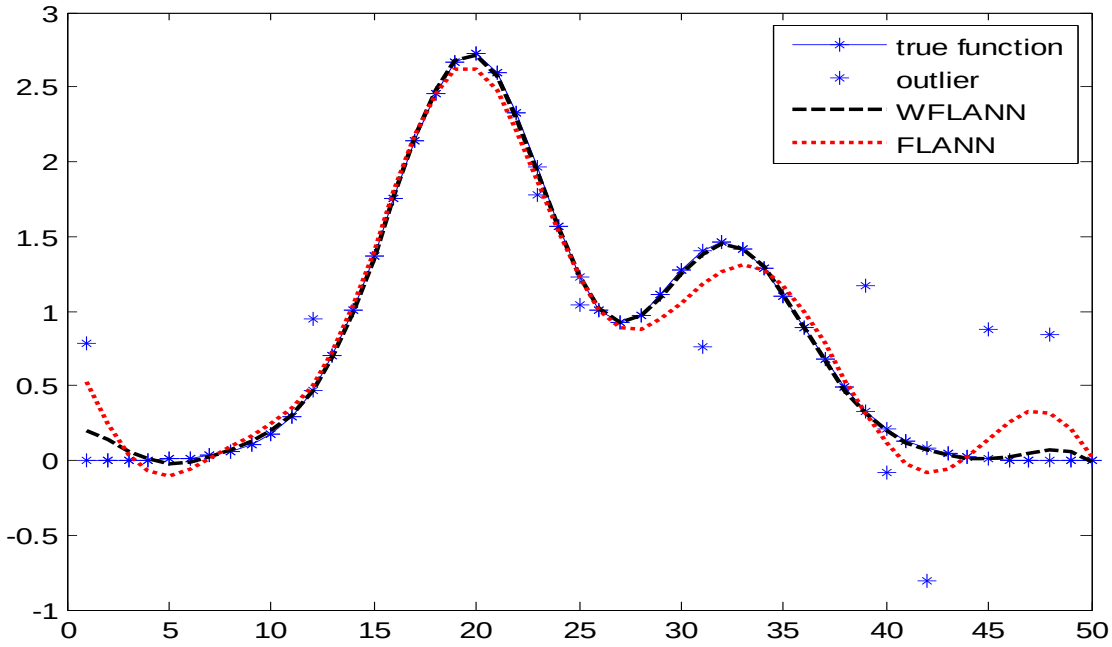

**(d)**

**Figure 3 Simulations for ANN and WNN of Example 2: (a) uncorrupted data, (b) 10% corrupted data (c) 20% corrupted data (d) 40% corrupted data**

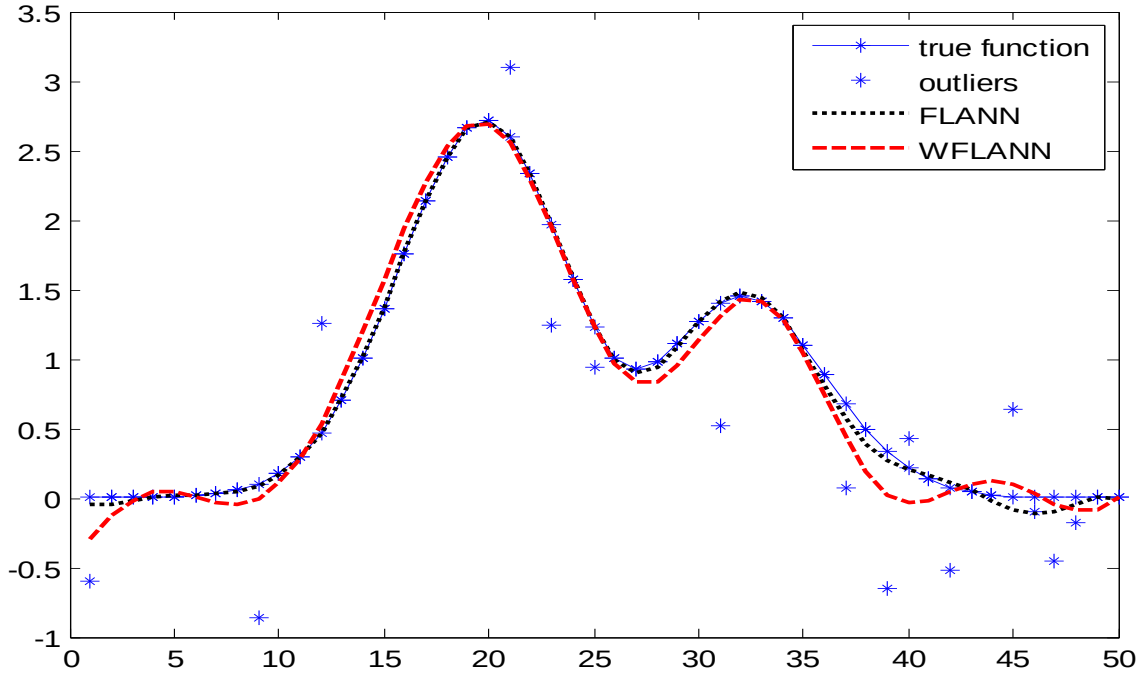
Results are shown in Fig. (4) for WFLANN and FLANN approximates.



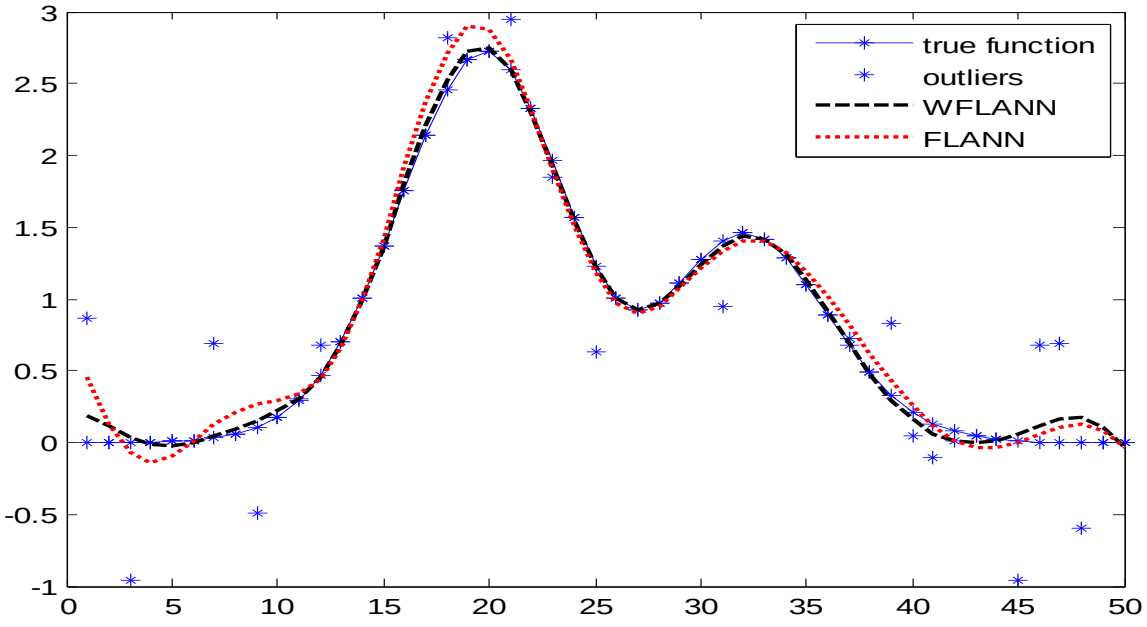

(a)




(b)




**(c)**




**(d)**

**Figure 4 Simulations for FLANN and WFLANN of Example 2: (a) uncorrupted data, (b) 20% corrupted data (c) 30% corrupted data (d) 40% corrupted data**

In previous examples we saw the effectiveness of Wilcoxon learning in non-linear function approximation.

In the following examples we will see the performance of these machine learning algorithms when these are applied to non-linear channel equalization in the presence of outliers.

### 1.5.2. Channel Equalization

Adaptive channel equalization has been found to be very important for effective digital data transmission over linear dispersive channels. In high speed data transmission, the amplitude and phase distortion due to variation of channel characteristics to which the data signal will be subjected is to be suitably compensated. This compensation is usually accomplished by passing samples of the received signal through a linear adaptive equalizer consisting of a tapped delay line (TDL) having adjustable coefficients. In this form of equalizer structure, the current and past values of the received signal are linearly weighted by equalizer coefficients and summed to produce the output. Most of the known methods used to adjust the tap coefficients of the equalizer are iterative in which some error criterion is minimized. In such techniques, a known sequence of a white spectrum is transmitted; based on the difference between this known sequence and the output sequence of the equalizer its coefficients are determined. However, the distortion caused by the dispersive channel is nonlinear in nature in most of the practical situations. The received signal at each sample instant may be considered as a nonlinear function of the past values of the transmitted symbols. Further, since the nonlinear distortion varies with time and from place to place, effectively the overall channel response becomes a nonlinear dynamic mapping. Because of this, the performance of the linear TDL equalizer is limited.

Because of their large parallelism and nonlinear processing characteristics, ANNs and FLANNs are capable of performing complex nonlinear mapping between their input space and output space. They are capable of forming arbitrarily nonlinear decision boundaries to take up complex classification tasks. Channel equalizers using a multilayer perceptron (MLP) and Functional link approximation network(FLANN) has been reported before. In this it has been

shown that the ANN and FLANN based equalizers are capable of performing quite well in compensating the nonlinear distortion introduced by the channel.

A basic block diagram of channel equalization is shown in Fig.(5).The transmitted signal *x(n)* passes through the channel .The block N.L accounts for the nonlinearity associated with the channel and *q(n)* is the Gaussian noise added through the channel. The equalizer is placed at the receiver end. The output of the equalizer is compared with the delayed version of the transmitted signal to calculate the error signal *e (n)*, which is used by the update algorithm to update the equalization coefficient such that the error becomes minimum.

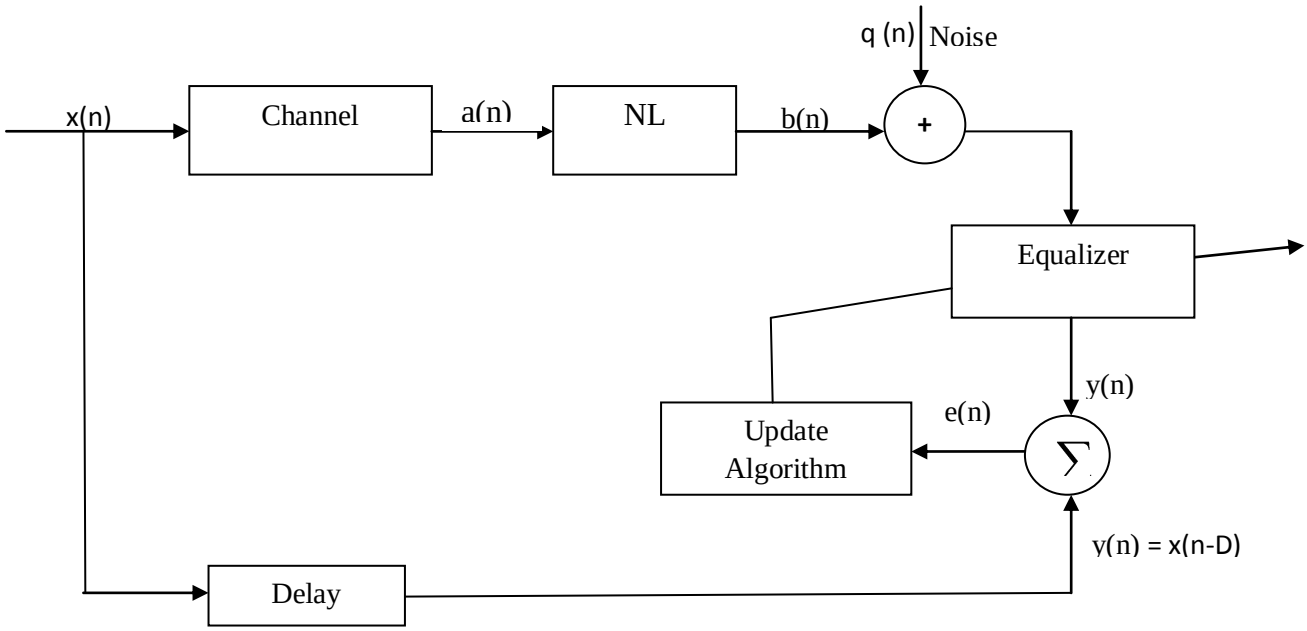



**Figure 5 Digital communication system with equalizer.**

Structures which are normally used for equalizers are:

a) LIN Structure:

The block diagram of a LIN structure is depicted in Fig.(6).The input signals are first passed through a bank of k delays to form $X(n) = [x(n)\ x(n-1)\ \dots\ \dots\ x(n-k)]^T$ , where the subscript $T$ denotes the transpose of a matrix, and this signal vector obtained is multiplied with a set of weights $W(n) = [w_0(n)\ w_1(n)\ \dots\ \dots\ w_k(n)]$ which gives us

$\hat{y}(n)$. The error function $e(k)$ is computed as the difference between $\hat{y}(n)$ and $(n)$ . This error is then minimized in several iterations using LMS algorithm.

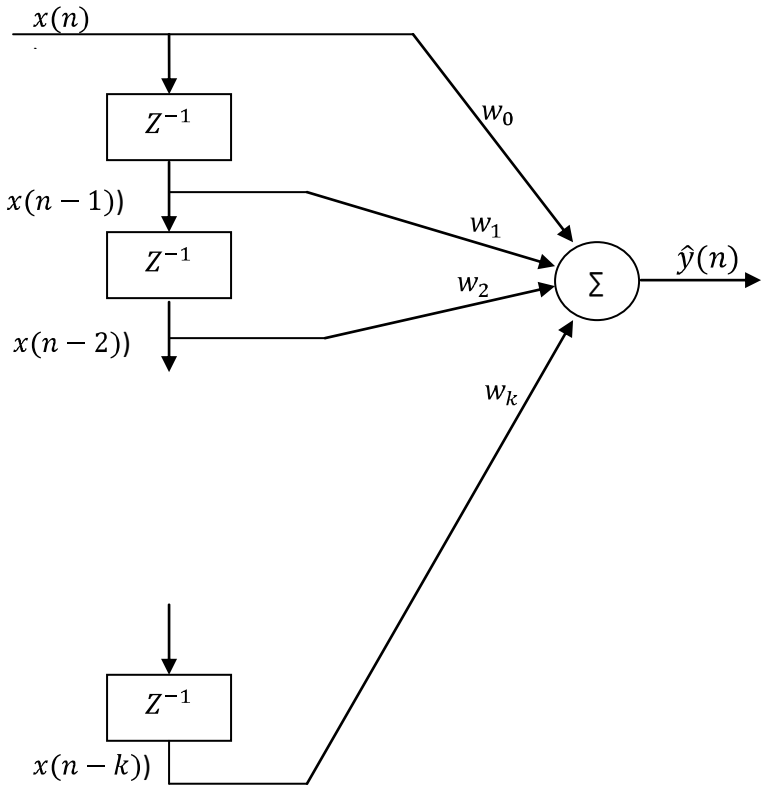



**Figure 6 LIN Structure**

b) MLP Structure

The block diagram of a system exploiting MLP networks is given in Fig. (7). The multilayer structure of an MLP networks is composed of an input layer, an output layer and one or more hidden layers. It is indicated in previous works that about 2 to 3 hidden layers are enough for most systems. In the figure the structure has k inputs, 2 hidden layers with m and p nodes respectively and r outputs. The structure of a system applying MLP network is pretty simply. The node output from each of the layers is directed used as the input to the successive layer nodes. The numbers of nodes as well as the transfer functions in the layers are allowed to be different from each other.

Through the multilayer structure, we can attain nonlinear mapping from input to output signals. Generally, we use the BP algorithm to train the MLP networks.

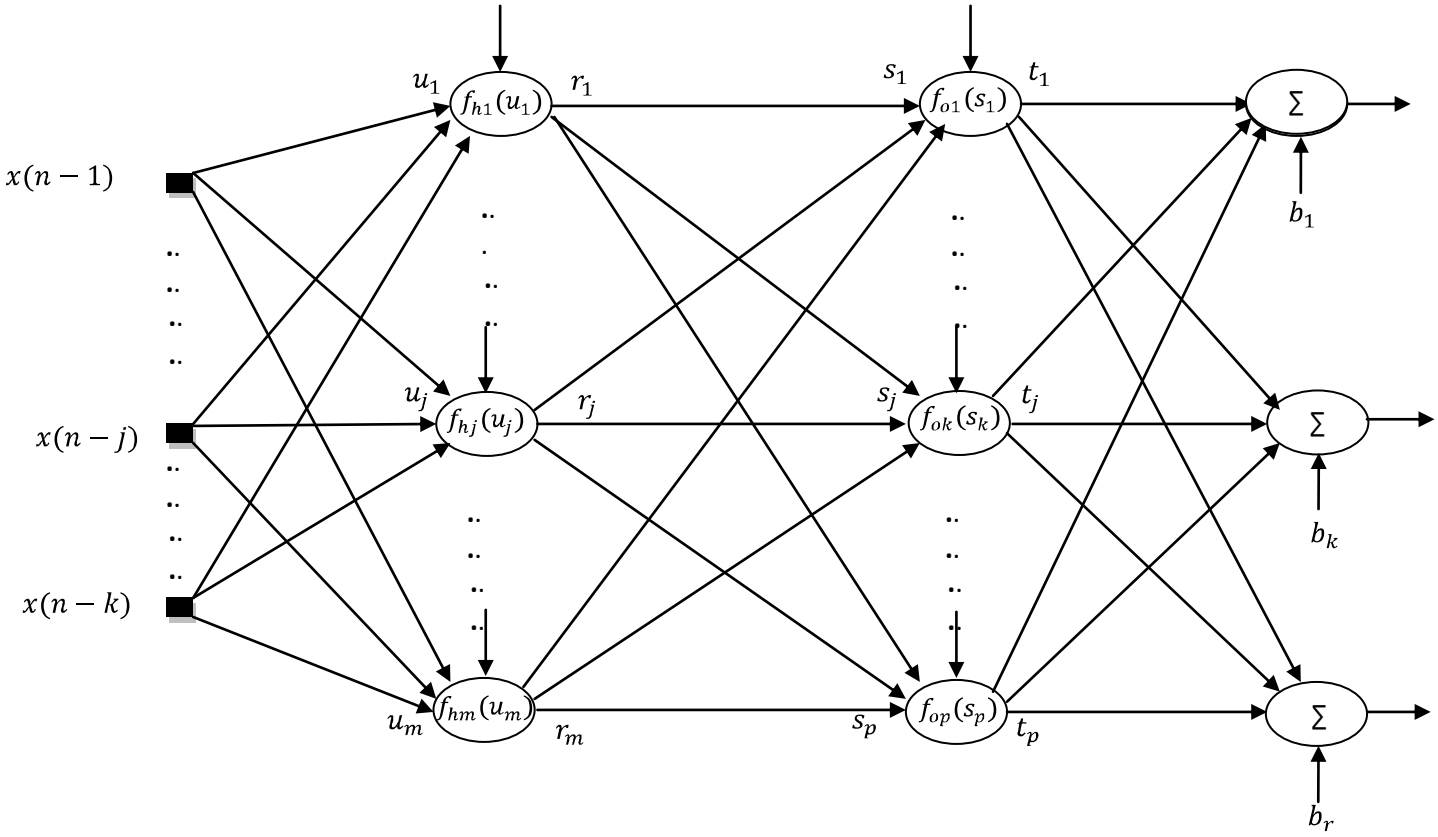



**Figure 7 MLP Structure**

c) FLANN Structure

$b_r$

The block diagram of a system with FLANN is shown in Fig. (8), where the block labeled F.E. denotes a functional expansion. These functions map the input signal vector $X = [x_1\ x_2\ \dots\ x_n]^T$ into $N$ linearly independent functions. $[\phi_1(X)\ \phi_2(X)\ \dots\ \phi_N(X)]^T$ . The linear combination of these function values is presented in its matrix form, that is, $S = W * \phi$, where $S = [s_1\ s_2\ \dots\ s_m]^T$, 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 $\hat{Y} = [\hat{y}_1\ \hat{y}_2\ \dots\ \hat{y}_m]^T$ , where $\hat{y}_j = \rho(s_j)$ ,$j = 1,2,\dots,m$ . Here the nonlinear function is normally defined as $\rho(.) =$

tanh (.) or any other activation function. 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.

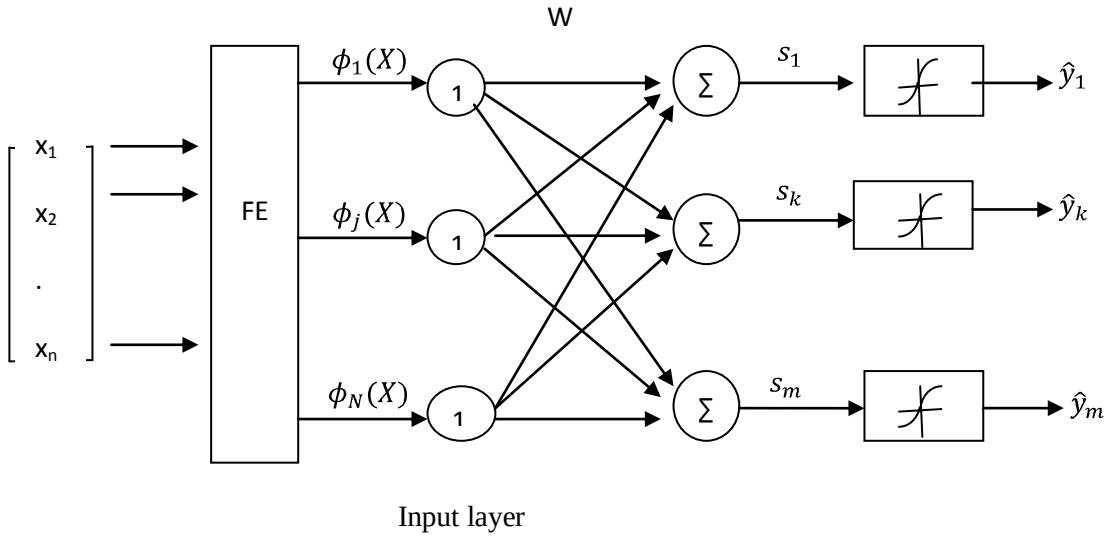



**Figure 8 FLANN Structure**

I ıly chosen binary (-1,1) $x-$ points (training patterns) with the corresponding $y-$ values (target values) composed of the same $x-$ points but with randomly chosen position where the binary values are reversed and these acts as outliers in the process of channel equalization. To this end, 20%, 30%, and 40% randomly chosen -values of the training data points will be corrupted. Then the trained equalizer will be used for testing. The channel is represented using a linear part in series with the non-linearity. Noise representing error in channel is added after the nonlinearity.

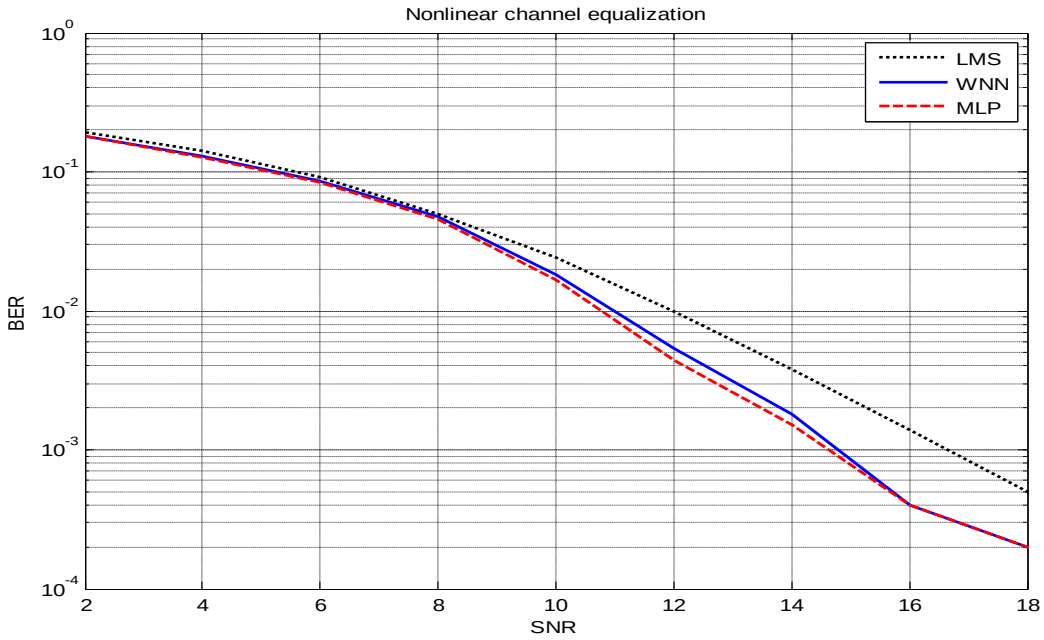
**Example 3:**

$$\text{CH} = 0.26 + 0.93 * z^{-1} + 0.26 * z^{-2}$$

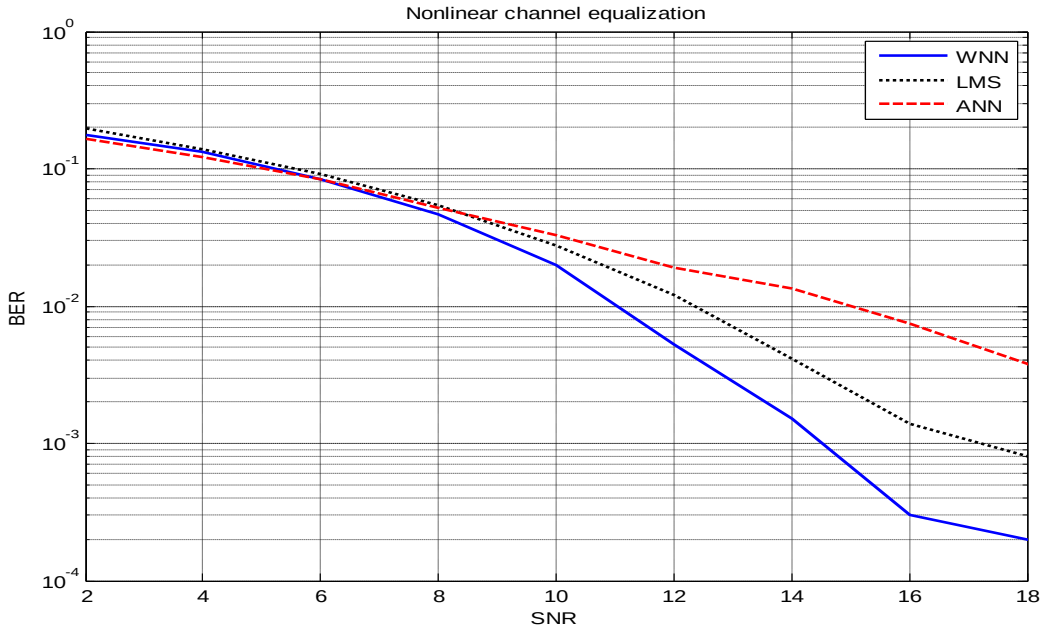
$$\text{NL} = b(k) = a(k) + 0.2 * a^2(k) - 0.1 * a^3(k)$$

In this example, we compare the performances of LIN, ANN, WNN, FLANN, and WFLANN. For LIN structure we use an 8 tap linear filter . For ANN and WNN, we use a structure consisting of 4 inputs, 1 hidden layer with 8 nodes and an output node and a unit bias at each hidden and output node..The activation functions of the hidden nodes as well as output nodes are bipolar sigmoidal functions. For FLANN and WFLANN, the number of functional expansion is 18 along with a unit bias, trigonometric expansion along with cross multiplication of input signals is used, and the activation function of the output node is a bipolar sigmoidal function.

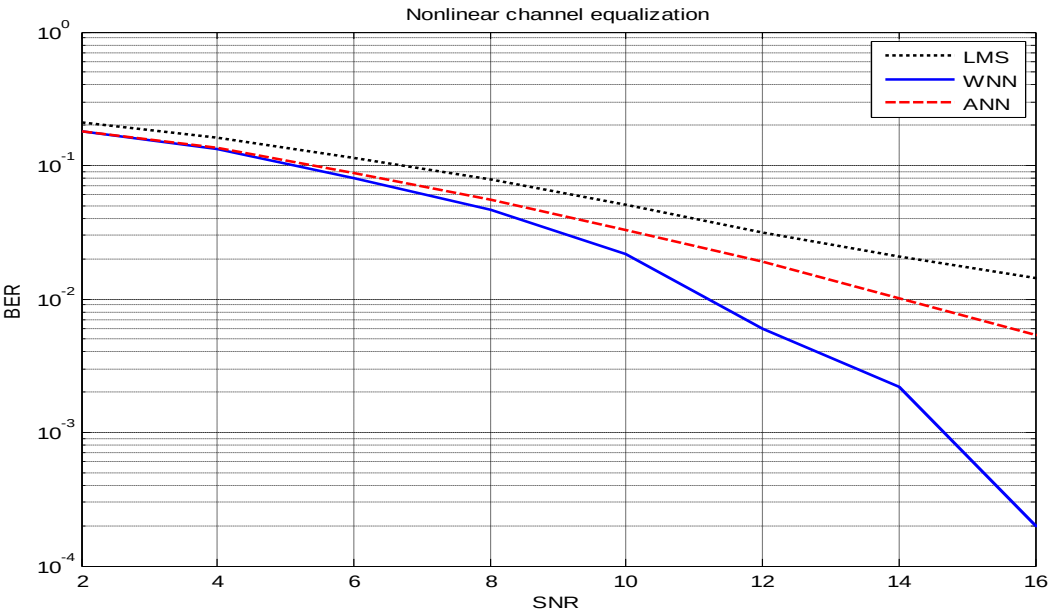
The below figure shows the comparision between performance of WNN,MLP & linear structure in equalization:



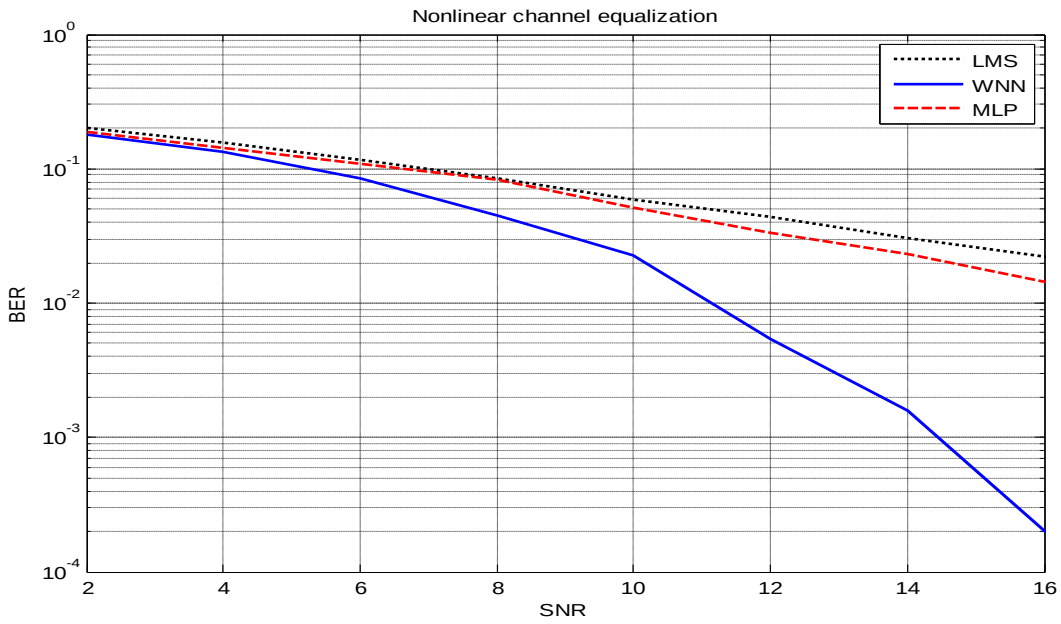

**(a)**


Nonlinear channel equalization
WNN
LMS
ANN
BER
SNR
10^0
10^-1
10^-2
10^-3
10^-4
2
4
6
8
10
12
14
16
18

**(b)**


Nonlinear channel equalization
LMS
WNN
ANN
BER
SNR
10^0
10^-1
10^-2
10^-3
10^-4
2
4
6
8
10
12
14
16

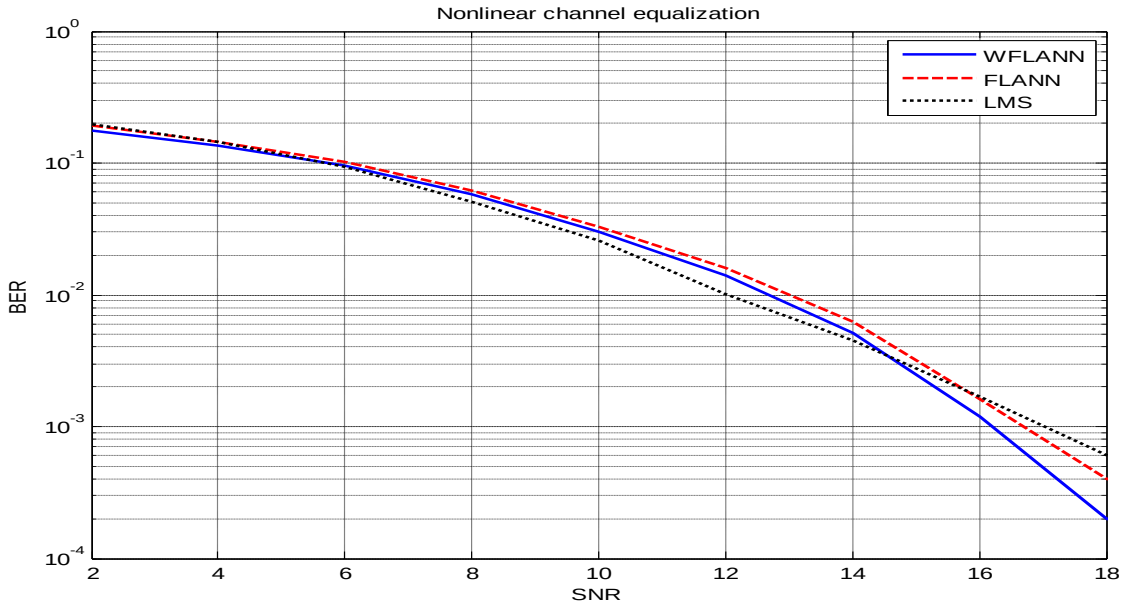
**(c)**



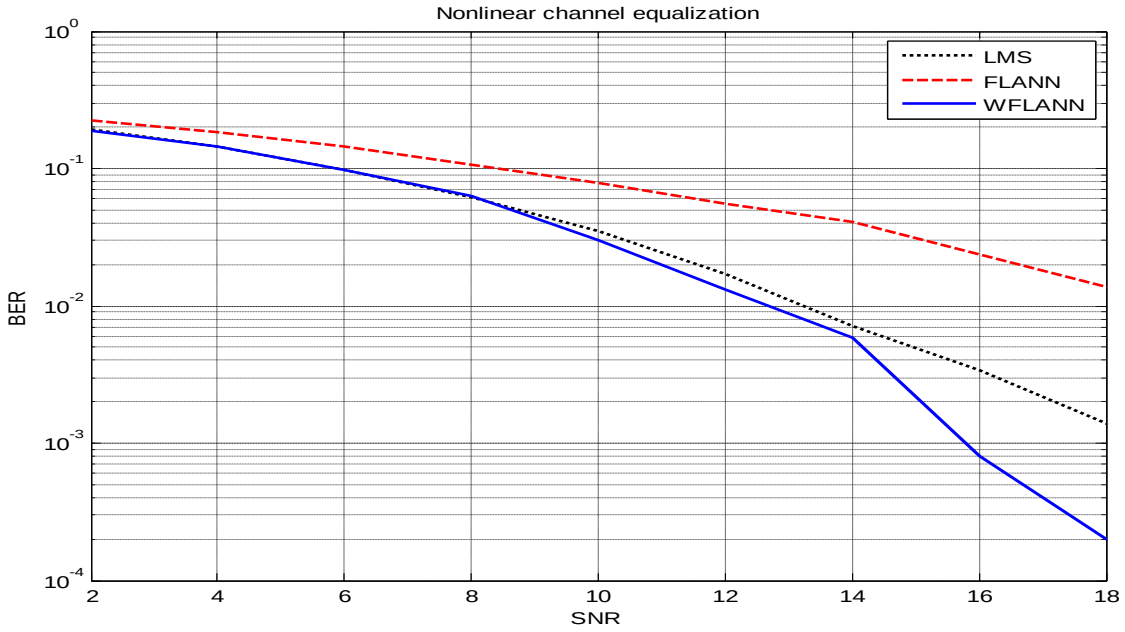

**(d)**

**Figure 9 Simulations for ANN and WNN of Example 3 with training using : (a) uncorrupted data, (b) 20% corrupted data (c) 30% corrupted data (d) 40% corrupted data**

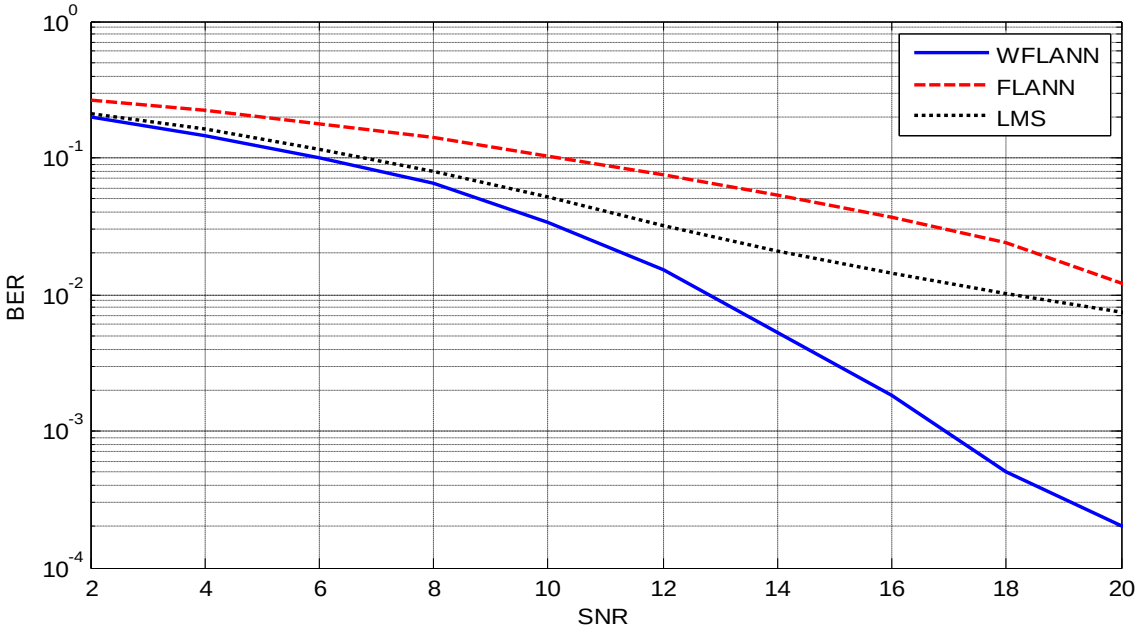
The below figure shows the comparision between performance of WFLNN,FLANN & linear structure in equalization:



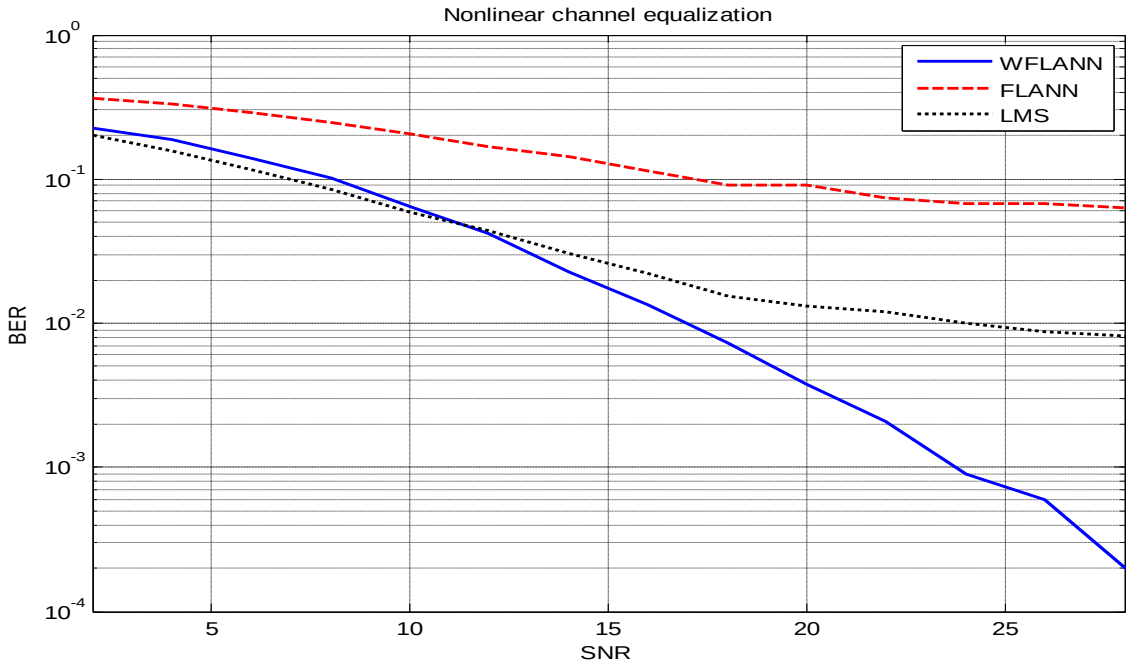

**(a)**


Nonlinear channel equalization
LMS
FLANN
WFLANN
BER
SNR
10^0
10^-1
10^-2
10^-3
10^-4
2
4
6
8
10
12
14
16
18

**(b)**


WFLANN
FLANN
LMS
BER
SNR
10^0
10^-1
10^-2
10^-3
10^-4
2
4
6
8
10
12
14
16
18
20

**(c)**




**(d)**

**Figure 10 Simulations for FLANN and WFLANN of Example 3 with training using : (a) uncorrupted data, (b) 10% corrupted data (c) 20% corrupted data (d) 30% corrupted data (e) 40% corrupted data**

**Example 4:**

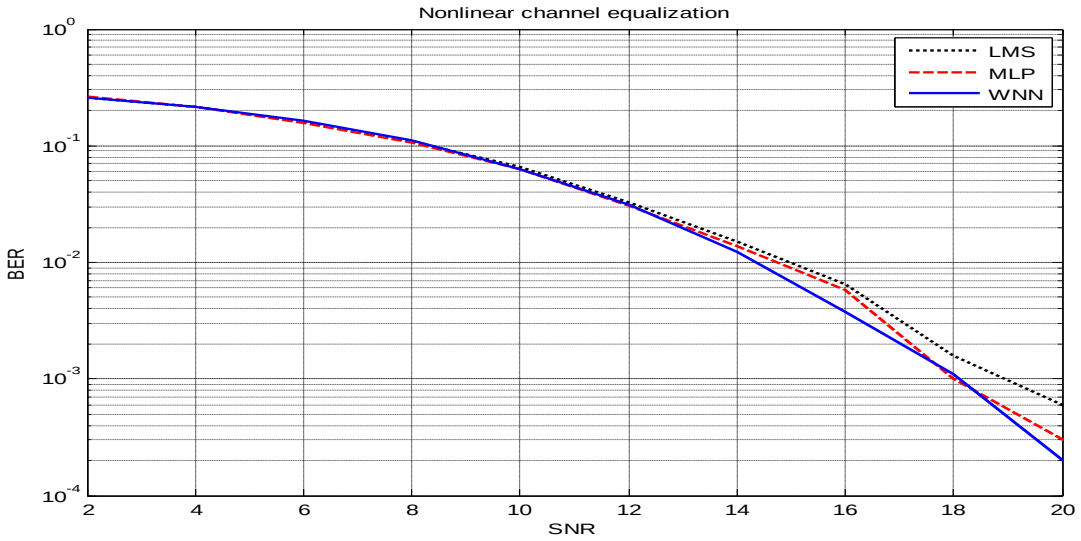
$$\text{CH} = 0.340 + 0.876 * z^{-1} + 0.340 * z^{-2}$$

$$\text{NL} = b(k) = \tanh(a(k))$$

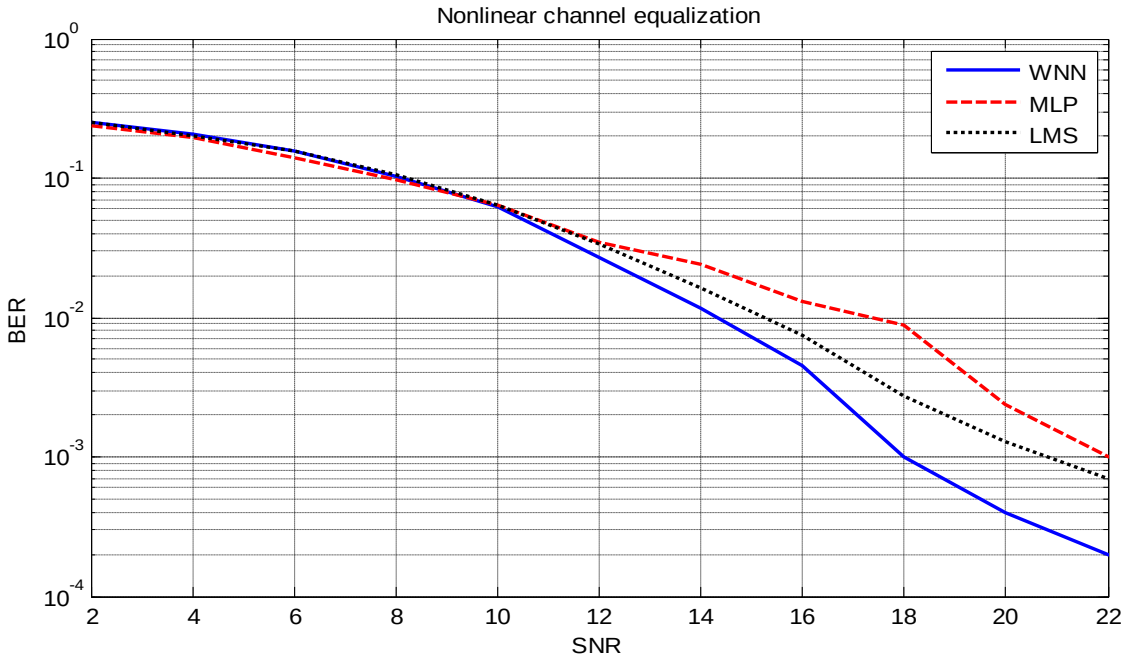
In this example, we compare the performances of LIN, ANN, WNN, FLANN, and WFLANN. For LIN structure we use an 8 tap linear filter. For ANN and WNN, we use a structure consisting of 4 inputs, 1 hidden layer with 8 nodes and an output node and a unit bias at each hidden and output node..The activation functions of the hidden nodes as well as output nodes are bipolar sigmoidal functions. For FLANN and WFLANN, the number of functional expansion is 18 along with a unit bias, trigonometric expansion along with cross multiplication

of input signals is used, and the activation function of the output node is a bipolar sigmoidal function.

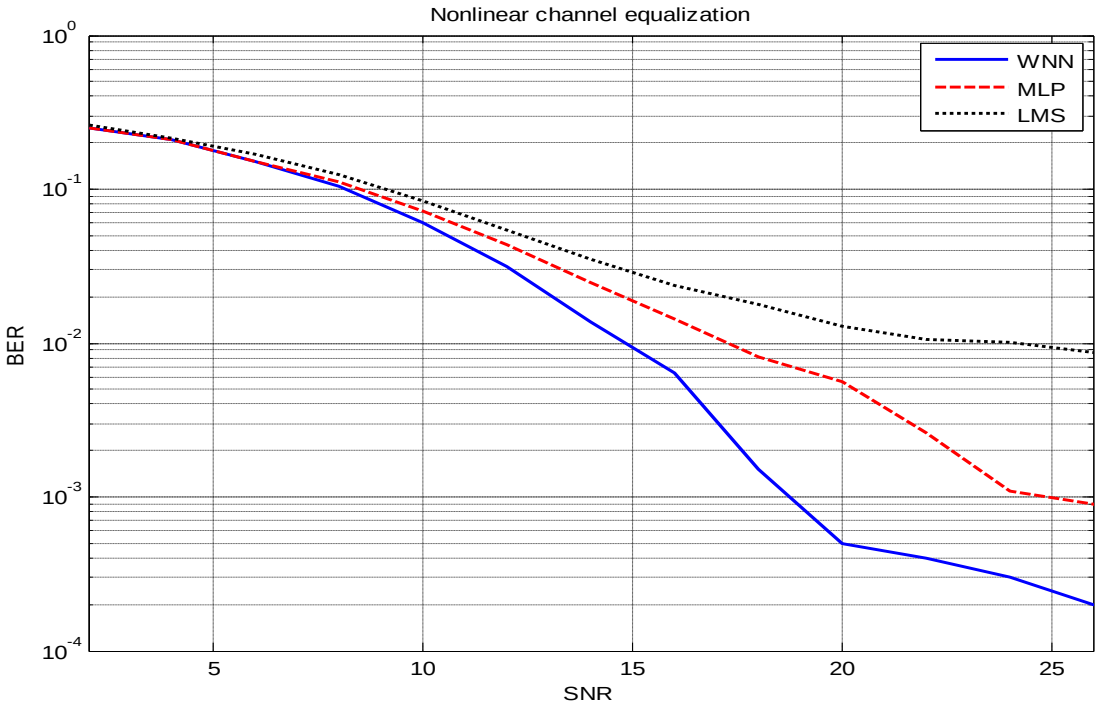
The below figure shows the comparison between performance of WNN,MLP & linear structure in equalization:



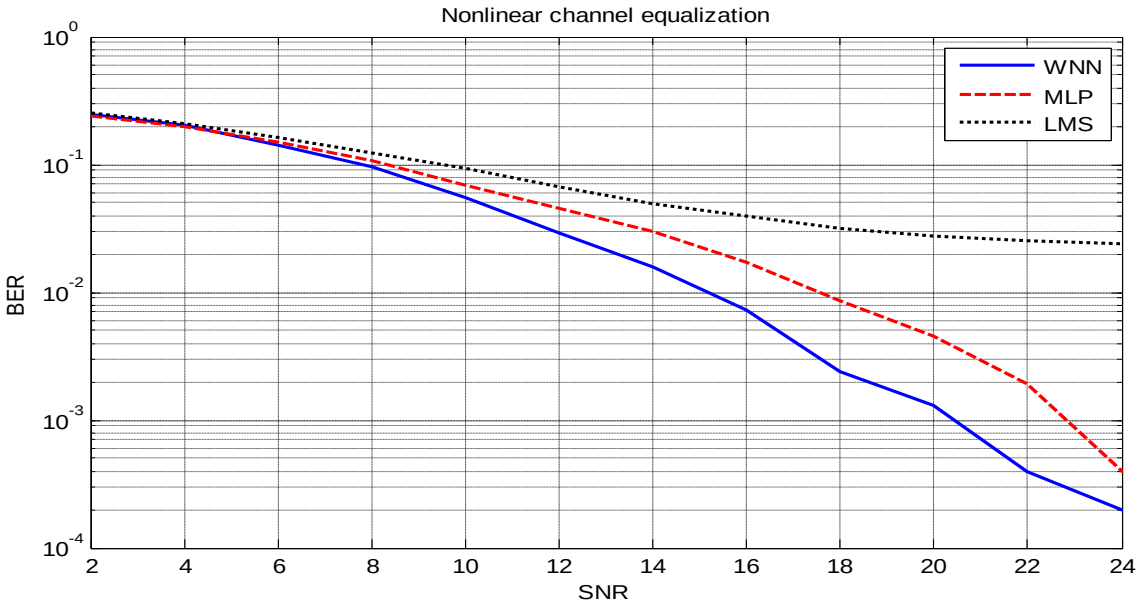

**(a)**




**(b)**



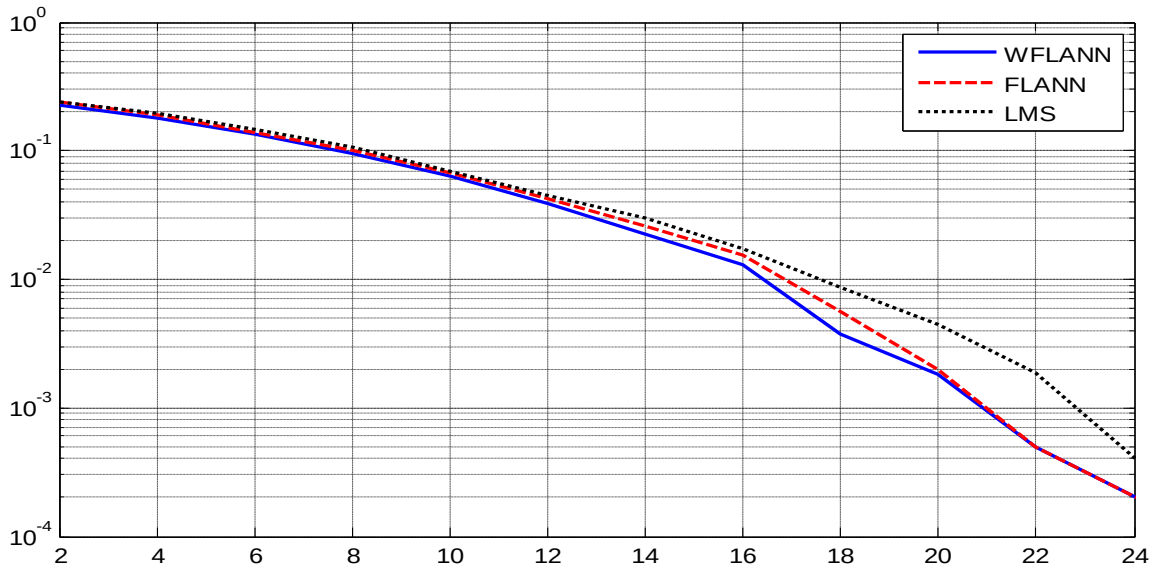

**(c)**



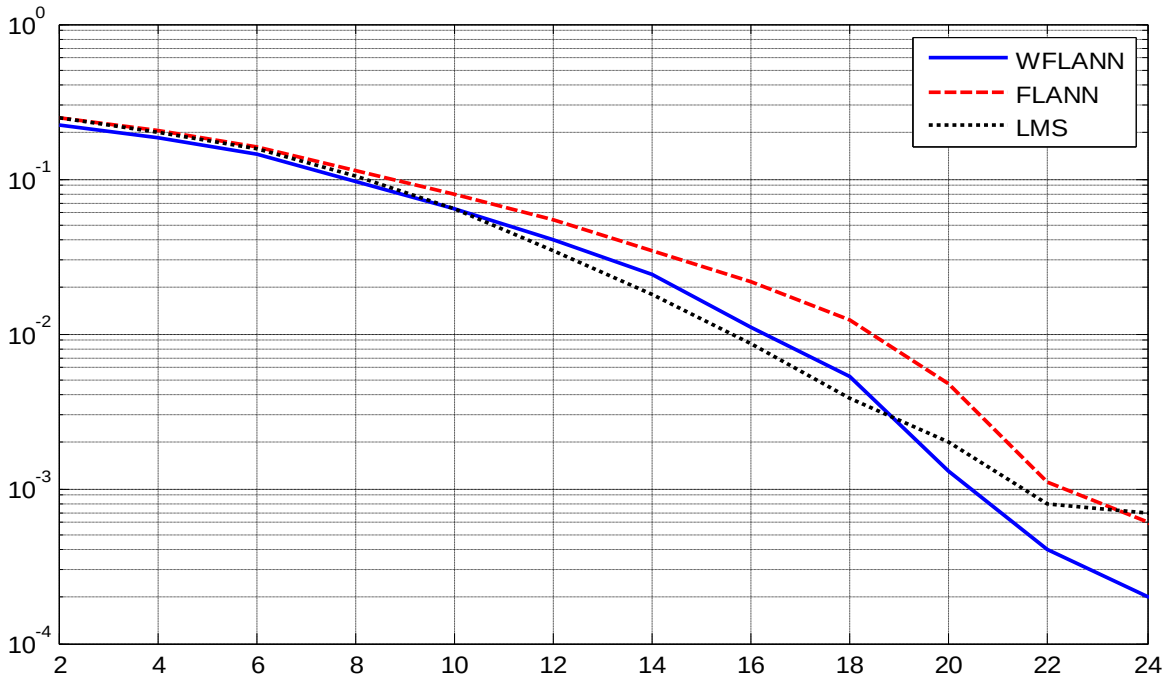

**(d)**

**Figure 11 Simulations for ANN and WNN of Example 4 with training using : (a) uncorrupted data, (b) 10% corrupted data (c) 20% corrupted data (d) 30% corrupted data (e) 40% corrupted data**

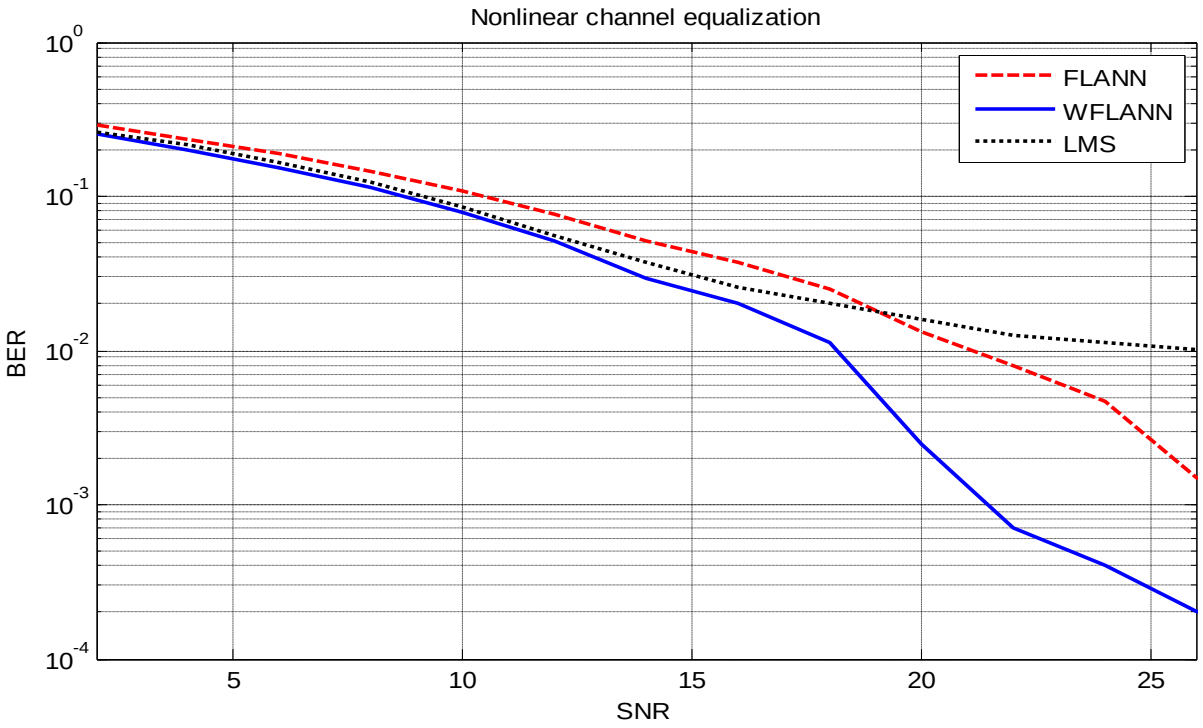
The below figure shows the comparison between performance of WNN,MLP & linear structure in equalization:



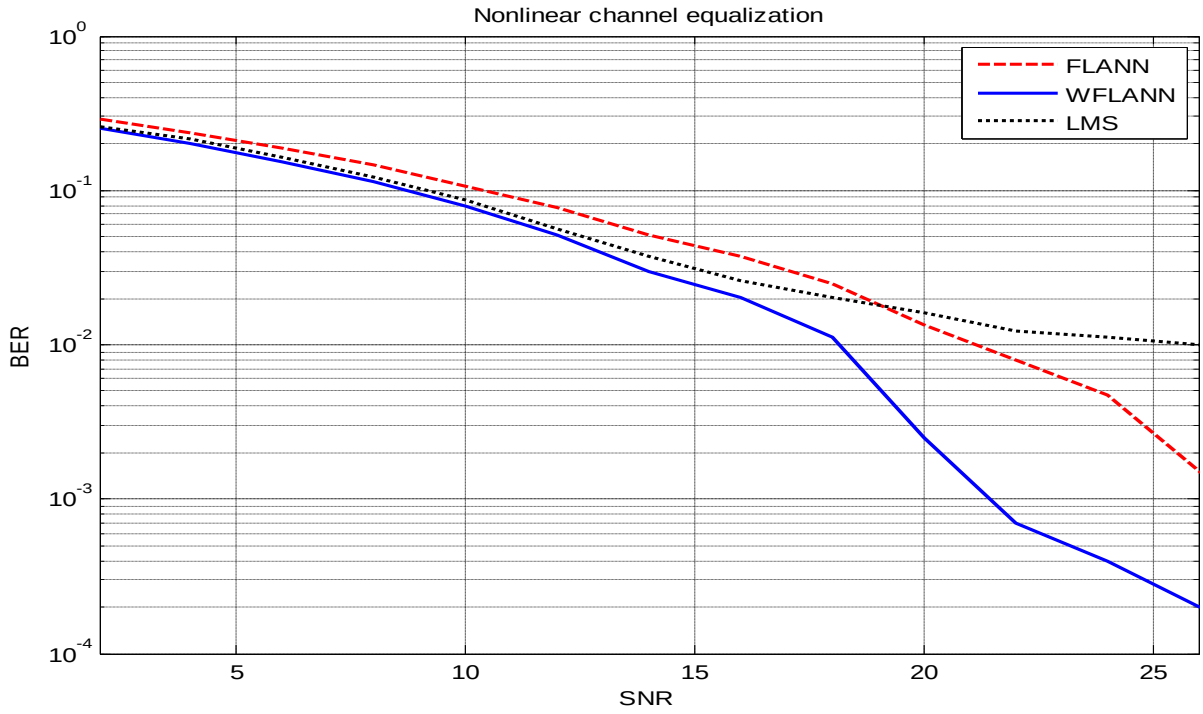

**(a)**




**(b)**




**(c)**




**(d)**

**Figure 12 Simulations for FLANN and WFLANN of Example 3 with training using : (a) uncorrupted data, (b) 10% corrupted data (c) 20% corrupted data (d) 30% corrupted data**

# CHAPTER 2

# NON-LINEAR SYSTEM MODELING USING VOLTERRA SERIES AND ITS LINEARIZATION

## 2.1. Introduction

Volterra series expansions form the basis of the theory of polynomial nonlinear systems. Volterra expansion is a general method to model nonlinear systems with soft or weak nonlinearities. This includes saturation type nonlinearities observed in power amplifiers and loudspeakers.

A truncated p-th order Volterra expansion is given as:

$$y(n) = \sum_{k=0}^{p} H_k\,[x(n)]$$

In this representation, $H_k = h_k(m_1, m_2, \dots\dots\dots, m_k)$ is the k-th order operator and $h_k(\dots)$ is called the k-th order volterra kernel. Volterra series expansion is linear w.r.t. the kernel coefficients. In other words, the nonlinearity of the expansions is completely due to the multiple products of the delayed input values.

$$y(n) = h_0 + \sum_{m_1=0}^{M1} h_1\,(m_1) * x(n-m_1) + \sum_{m_1=0}^{M_2}\sum_{m_2=0}^{M_2} h_2\,(m_1, m_2) * x(n-m_1) * x(n-m_2)$$
$$+ \dots\dots\dots + \sum_{m_1=0}^{M_p}\sum_{m_2=0}^{M_p} \dots\dots\dots\dots\dots \sum_{m_p=0}^{M_p} h_p\left(m_1, m_2, \dots\dots\dots, m_p\right) * x(n-m_1)$$
$$* x(n-m_2) * \dots\dots\dots\dots * x\left(n-m_p\right) \qquad (2.1)$$

Volterra series can be regarded as a power series with memory or the extension of FIR filters to representation of nonlinear systems. Small loudspeakers and other non linear devices can be sufficiently modeled by a 2nd or 3rd order Volterra model. The 2nd order Volterra model is given as:

$$y(n) = h_0 + \sum_{m_1=0}^{M1} h_1\,(m_1) * x(n-m_1) + \sum_{m_1=0}^{M_2}\sum_{m_2=0}^{M_2} h_2\,(m_1, m_2) * x(n-m_1) * x(n-m_2)$$

The first term is a constant and is generally assumed to be zero, the second term is the linear response (H1), and the third term is the nonlinear response (H2) .

Figure. (13) shows the p-th order Volterra model based on equation (2.1).The model parameters are found by minimizing the weighted mean square error(WMSE).

$$E = \sum_{n=0}^{N} \lambda^{N-n} * (d(n) - y(n))^2$$

Where, $\lambda$ is the weight factor, N is the adaptation length and d(n) is the desired nonlinear system output. The minimization is accomplished using the LMS or RLS algorithms [17].

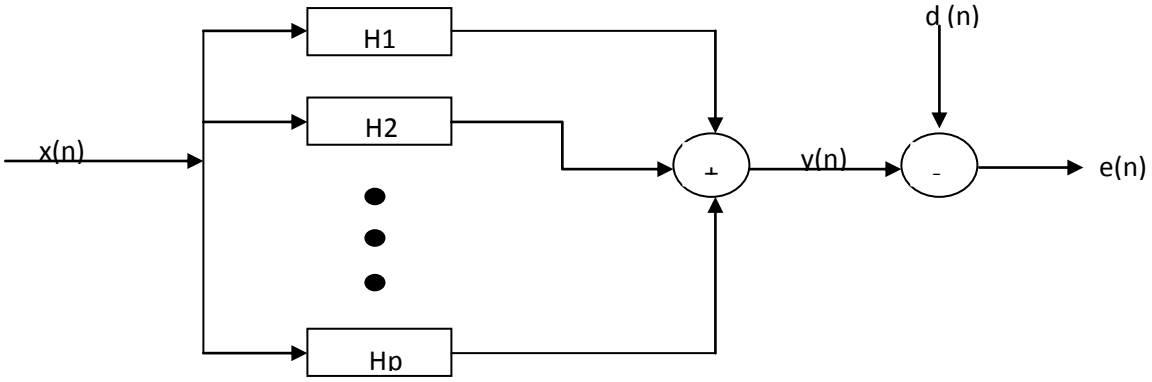



**Figure 13 Pth order Volterra Model**

The Volterra series have been widely applied as nonlinear system modeling technique with considerable success. When the nonlinear system order is unknown, adaptive methods and algorithms are widely used for the Volterra kernel estimation. The accuracy of the Volterra kernels will determine the accuracy of the system model and the accuracy of the inverse system used for compensation of the nonlinearity of the system.

## 2.2. Volterra Kernels Estimation and input vectors:

A third order nonlinear system with memory is identified using the adaptive algorithm (LMS / RLS) for Volterra kernels estimation. The implementation of the adaptive Volterra filter is based on the extended input vector and on the extended filter coefficients vector. Due to the linearity of the input-output relation of the Volterra model with respect to filter coefficients, the implementation of the adaptive algorithm was realized as an extension of the algorithm for linear filters.

Next we will introduce the input vectors corresponding to different orders kernels. The first order input vector, corresponding to a filter length $M$ = 3, is defined as follows:

$$X^{(1)T} = [x(n)\ x(n-1)\ x(n-2)] \tag{2.2}$$

If we consider equal memories for different orders filters, "the second order input vector" can be expressed by:

$$X^{(2)} = X^{(1)} * X^{(1)T} \tag{2.3}$$

For symmetric kernels only the elements $x_{i,j}$ , having $i \geq j$ , of $X^{(2)}$ , are selected in the input-output relation of the Volterra filter. Hence "the second order input vector", written in vector form is:

$$X^{(2)T} = \begin{bmatrix} x^2(n) & x(n)*x(n-1) & x(n)*x(n-2) & x^2(n-1) \\ & x(n-1)*x(n-2) & x^2(n-2) & \end{bmatrix} \tag{2.4}$$

and has the dimension (1×6).

For "the third order input vector" we propose to express the multiple input delayed signal products by matrices elements. These matrices can be generated by multiplying "the second order input vector" defined according to Eq. (2.3) by the elements of the first order input vector. If we consider equal filters, *M=3*, and symmetric kernels it follows:

$$X^{(2)} * x(n) = \begin{bmatrix} x^3(n) & x^2(n) * x(n-1) & x^2(n) * x(n-2) \\ \dots & x(n) * x^2(n-1) & x(n) * x(n-1) * x(n-2) \\ \dots & \dots & x(n) * x^2(n-2) \end{bmatrix} \quad (2.5)$$

$$X^{(2)} * x(n-1) = \begin{bmatrix} \dots & \dots & \dots \\ \dots & x^3(n-1) & x^2(n-1) * x(n-2) \\ \dots & \dots & x(n-1) * x^2(n-2) \end{bmatrix} \quad (2.6)$$

$$X^{(2)} * x(n-1) = \begin{bmatrix} \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & x^3(n-2) \end{bmatrix} \quad (2.7)$$

Hence, "the third order input vector" consists in fact, in that case, of 3 matrices as indicated in Equations $(2.5) - (2.7)$ and corresponds to a symmetric third order Volterra kernel. We can write "the third order input vector" in vector form as follows:

$$X^{(3)T} = \begin{bmatrix} x^3(n) & x^2(n) * x(n-1) & \dots & x(n) * x^2(n-2) & x^3(n-1) \\ & & \dots & x(n-1) * x^2(n-2) & x^3(n-2) \end{bmatrix} \quad (2.8)$$

its dimension is (1×10).

The defined input vectors will be used to implement the LMS and RLS Volterra filter in a typical nonlinear system identification application.

## 2.3. Volterra Kernels Estimation by the LMS Adaptive Algorithm

A typical adaptive technique employing LMS algorithm used for Volterra kernels identification is shown in Figure. (14).

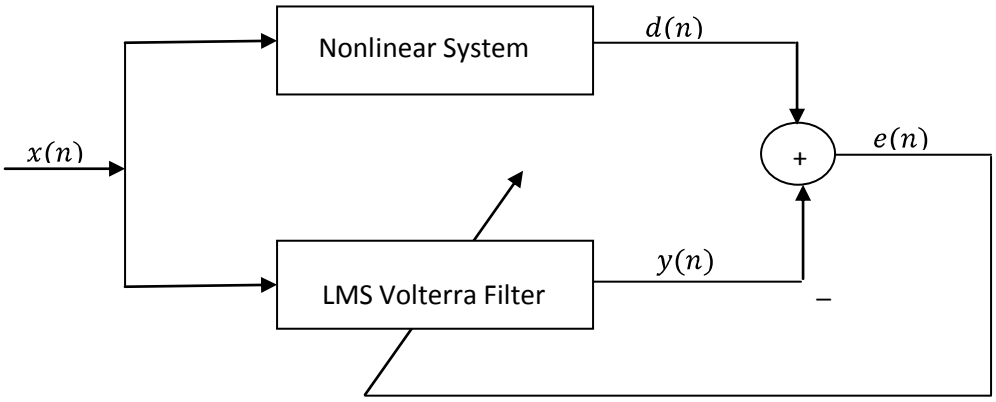



**Figure 14 Volterra kernel identification using adaptive method**

The Volterra filter of fixed order and fixed memory adapts to the unknown nonlinear system using one of the various adaptive algorithms. A simple and commonly used algorithm uses an LMS adaptation criterion. The aim of this section is to discuss the simplest of the algorithms, the LMS algorithm. Although the LMS algorithm has its weaknesses, such as its dependence on signal statistics, which can lead to low speed or large residual errors, it is very simple to implement and well behaved compared to the faster recursive algorithms. The main topic of this section is to discuss the extension of the algorithm to the nonlinear case using the previously defined input vectors. The discrete time impulse response of a first order (linear) system with memory span *M*, is written in vector form as in Eq. (2.9) and the input vector as in Eq.(2.10).

$$H_k^{(1)T} = \left[h_k^{(1)}(0) \;\; h_k^{(1)}(1) \;\ldots\; h_k^{(1)}(M-1)\right] \tag{2.9}$$

$$X_k^{(1)T} = [x_k \;\; x_{k-1} \;\; \dots \;\; x_{k-M+1}] \tag{2.10}$$

In Eq.(2.9) the filter order is written as superscript. This notation will be kept consistent for the rest of the section. Then, the output of a linear system is written as:

$$y_k = H_k^{(1)T} * X_k^{(1)} \tag{2.11}$$

At sample *k*, the desired output is $d_k$ and the linear adaptive filter output is $y_k$ . For the LMS algorithm, we minimize the Eq.(2.12).

$$E[e_k^2] = E\left[d_k - H_k^{(1)T} * X_k^{(1)}\right] \tag{2.12}$$

The vector $H^*$ that minimizes the Eq. (2.12 ) is given by :

$$H^* = R_{xx}^{-1} * G$$

Where: $R_{xx} = E\left[X_k^{(1)} * X_k^{(1)T}\right]$ is the input correlation matrix and $G = E\left[X_k^{(1)} * y_k\right]$

The well known LMS update equation for a first order filter is:

$$H_{k+1}^{(1)} = H_k^{(1)} + \mu * e_k * X_k^{(1)}$$

where μ is a small positive constant (referred to as the step size) that determines the speed of convergence and also affects the final error of the filter output.

The extension of the LMS algorithm to higher order (nonlinear) Volterra filters involves a few simple changes. Firstly the vector of the impulse response coefficients becomes the vector of Volterra kernels coefficients. Also the input vector, which for the linear case contained only a linear combination, for nonlinear Volterra filters, complicates.

Consider the Volterra representation with symmetric kernels. There are two parts of this representation: (1) the Volterra kernel estimates, and (2) the products of the delayed input signal. If we express the Volterra kernels and the input signal products in vector form, then we can write the adaptive Volterra filter output using the vector notation. Each Volterra kernel (estimate at sample *k*) can be written in vector form.

For simplicity we have constructed the nonlinear adaptive filter considering only first order and 3rd order Volterra kernels.

The Eq.(2.13) gives "the input matrix" at sample *k*, containing the first, second and the third order input vectors defined previously.

$$X_k = \begin{bmatrix} X_k^{(1)T} \\ X_k^{(2)T} \\ X_k^{(3)T} \end{bmatrix} \tag{2.13}$$

The size of the input matrix is determined by the size of the third order input vector$X_k^{(3)}$. **"**The filter coefficients matrix" at sample *k* is given by:

$$H_k = \begin{bmatrix} H_k^{(1)T} \\ H_k^{(2)T} \\ H_k^{(3)T} \end{bmatrix} \tag{2.14}$$

where $H_k^{(1)T}$ is given by the Eq. (2.10), $H_k^{(2)T}$ and $H_k^{(3)T}$ are the second and third order kernel expressed in vector form as indicated in Eq.(2.15) and (2.16) respectively.

$$H_k^{(2)T} = \left[h_k^{(2)}(0,0) \;\; h_k^{(2)}(0,1) \;\ldots\; h_k^{(2)}(M-1, M-1)\right] \tag{2.15}$$

$$H_k^{(3)T} = \left[h_k^{(3)}(0,0,0) \quad h_k^{(3)}(0,0,1) \quad \ldots \quad h_k^{(3)}(M-1,M-1,M-1)\right] \tag{2.16}$$

The update equation for the LMS Volterra filter can be written also in matrix form:

$$\begin{bmatrix} H_{k+1}^{(1)T} \\ H_{k+1}^{(2)T} \\ H_{k+1}^{(3)T} \end{bmatrix} = \begin{bmatrix} H_k^{(1)T} \\ H_k^{(2)T} \\ H_k^{(3)T} \end{bmatrix} + e_k * \begin{bmatrix} \mu_1 & 0 & 0 \\ 0 & \mu_2 & 0 \\ 0 & 0 & \mu_3 \end{bmatrix} \begin{bmatrix} H_k^{(1)T} \\ H_k^{(2)T} \\ H_k^{(3)T} \end{bmatrix} \tag{2.17}$$

In the nonlinear case it is possible to set different step sizes for different order kernels. Consequently we have introduced the step size matrix *M*, defined by

$$\mu = \begin{bmatrix} \mu_1 & 0 & 0 \\ 0 & \mu_2 & 0 \\ 0 & 0 & \mu_3 \end{bmatrix}$$

## 2.4. Volterra Kernels Estimation by the RLS Adaptive Algorithm

The Volterra filter of a fixed order and a fixed memory adapts to the unknown nonlinear system using one of the various adaptive algorithms. The use of adaptive techniques for Volterra kernel estimation has been well studied. A simple and commonly used algorithm is based on the LMS adaptation criterion. Adaptive Volterra filters based on the LMS adaptation algorithm are computational simple but suffer from slow and input signal dependant convergence behavior and hence are not useful in many applications.

The aim of this section is to discuss the efficient implementation of the RLS adaptive algorithm on a third order Volterra filter. Due to the linearity of the input-output relation of the Volterra model with respect to filter coefficients, the implementation of the RLS algorithm can be realized as an extension of the RLS algorithm for linear filters. Hence we define the extended input vector, for a third order Volterra filter, as:

$$X = [x(n) \ldots. \ x(n-M+1) \quad x^2(n) \quad x(n) * x(n-1) \quad \ldots. x^2(n-M+1) \quad x^3(n) \quad x^2(n) * x(n-1) \quad \ldots. x^3(n-M+1)] \tag{2.18}$$

and the extended filter coefficients vector as:

$$H = \begin{bmatrix} h(0) \ldots. \ h(M-1) & h(0,0) & h(0,1) & \ldots. & h(M-1,M-1) \\ h(0,0,0) & h(0,0,1) & \ldots. & h(M-1,M-1,M-1) \end{bmatrix} \tag{2.19}$$

The elements of the extended input vector can be easily actualized based on the first order, second order and third order input vectors using the proposed relations $(2.5) - (2.7)$ As in the linear case the adaptive nonlinear system minimizes the following cost function at each time:

$$J(n) = \sum_{k=0}^{n} \lambda^{n-k} * (d(k) - H(n) * X'(k))^2 \tag{2.20}$$

Where H(n) and X(n) are the coefficients and the input signal vectors, respectively, as defined in (2.19) and (2.18), $\lambda$ is a factor that controls the memory span of the adaptive filter and $d(k)$ represents the desired output. The solution of equation (2.20) can be obtained recursively using the RLS algorithm.

The RLS algorithm updates the filter coefficients according to the following steps:

I. Initialization:

*Define the filter memory length for H (n) and X (n).*

*H (0) = [0 0 ... 0];*

$C_{XX}(0) = \delta * I$ where $\delta$ is a small positive constant ;

II. Operations: for an iteration (n)

1. Create the input vector:

X (n)

2. Compute the error:

$$e(n/n-1) = d(n) - H(n-1) * X'(n)$$

3. Compute the scalar:

$$\mu(n) = X(n) * C_{XX}(n-1) * X'(n)$$

4. Compute the matrix:

$$G(n) = (C_{XX}(n-1) * H'(n-1))/(\lambda + \mu)$$

5. Updates the filter vector:

$$H(n) = H(n-1) + e(n/n-1) * G'(n)$$

6. Updates the matrix $C_{XX}$ :

$$C_{XX} = \lambda^{-1} * (C_{XX}(n-1) - G(n) * X(n) * C_{XX}(n-1)$$

In the relations above $C_{XX}$ denotes the inverse autocorrelation matrix of the extended input signal. Inversion was done according to the matrix inversion lemma.

## 2.5. Nonlinearity compensation using Exact Inverse of Volterra models

To compensate for the nonlinearity of the nonlinear system, the signal is passed through a *predistortion* filter placed between the input signal and the nonlinear system as the shown in figure (15).

The function h(x) is approximated by a third order Volterra model as described in section 2.3 or section 2.4.

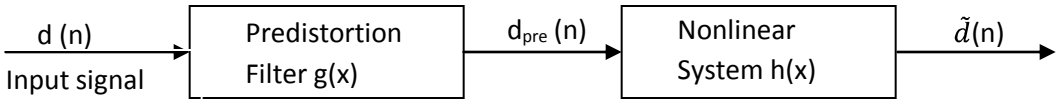



**Figure 15 Predistortion filter for nonlinearity compensation**

Ideally, the inverse of a nonlinear system must exactly compensate for both the linear and nonlinear distortions of the system. In contrast to the Volterra inverse that has a specific structure, we do not impose any constraints on the structure of the exact inverse. Instead of

defining the filter structure and finding its parameters as is customary, we directly compute the output of the predistortion filter $d_{pre}(n)$ so as to minimize the precompensation error $\mathrm{e(n)} = \mathrm{d(n)} - \hat{\mathrm{d}}\mathrm{(n)}$ as shown in fig (15). Input signal d (n) is fed into a time-varying predistortion filter. The output of the predistortion filter is routed into a mathematical model of the nonlinear system and also to the actual nonlinear system. The mathematical model of the loudspeaker predicts the next output of the loudspeaker $\hat{\mathrm{d}}\mathrm{(n)}$. This predicted output is used to derive a precompensation error signal

($e(n) = d(n) - \hat{\mathrm{d}}\mathrm{(n)}$) that is the difference between the ideal output and the predicted nonlinear system output. The parameters of the predistortion filter are then adjusted so that the instantaneous precompensation error e (n) is minimized.

For exact compensation, we have:

$$\hat{\mathrm{d}}\mathrm{(n)} = \mathrm{d(n)} \tag{2.21}$$

Assuming $x(n) = d_{pre}(n), y(n) = \hat{\mathrm{d}}\mathrm{(n)}, \mathrm{h}_0 = 0$ in Eq. (2.1), the value of $d_{pre}(n)$ that satisfies (2.21) is given as the solution of the following equation:

$$A(n) * d_{pre}^3(n) + B(n) * d_{pre}^2(n) + C(n) * d_{pre}^1(n) + D(n) = 0; \tag{2.22}$$

Where the coefficients { A(n) , B(n) , C(n) ,D(n) } are given as:

$$A(n) = h_3(0,0,0) \tag{2.23}$$

$$B(n) = h_2(0,0) + \sum_{j=1}^{M_3} h_3(0,0,j) * d_{pre}(n-j) \tag{2.24}$$

$$C(n) = h_1(0) + \sum_{j=1}^{M_2} h_2(0,j) * d_{pre}(n-j) + \sum_{j=1}^{M_3}\sum_{k=j}^{M_3} h_3(0,j,k) * d_{pre}(n-j) * d_{pre}(n-k) \tag{2.25}$$

$$D(n) = -d(n) + \sum_{j=1}^{M_1} h_1(j) * d_{pre}(n-j) + \sum_{j=1}^{M_2}\sum_{k=j}^{M_2} h_2(j,k) * d_{pre}(n-j) * d_{pre}(n-k)$$
$$+ \sum_{i=1}^{M_3}\sum_{j=i}^{M_3}\sum_{k=j}^{M_3} h_3(i,j,k) * d_{pre}(n-i) * d_{pre}(n-j) * d_{pre}(n-k) \quad (2.26)$$

Figure 8 shows the structure of the predistortion filter based on the inverse technique described called exact inverse technique. As seen here, the predistorted signal $d_{pre}(n)$ is the root of a quadratic equation whose coefficients depend on the parameters of the lnonlinear system model {H1,H2,H3} , the past values of the predistortion signal $d_{pre}(n)$ (the states) and the input signal d(n) .

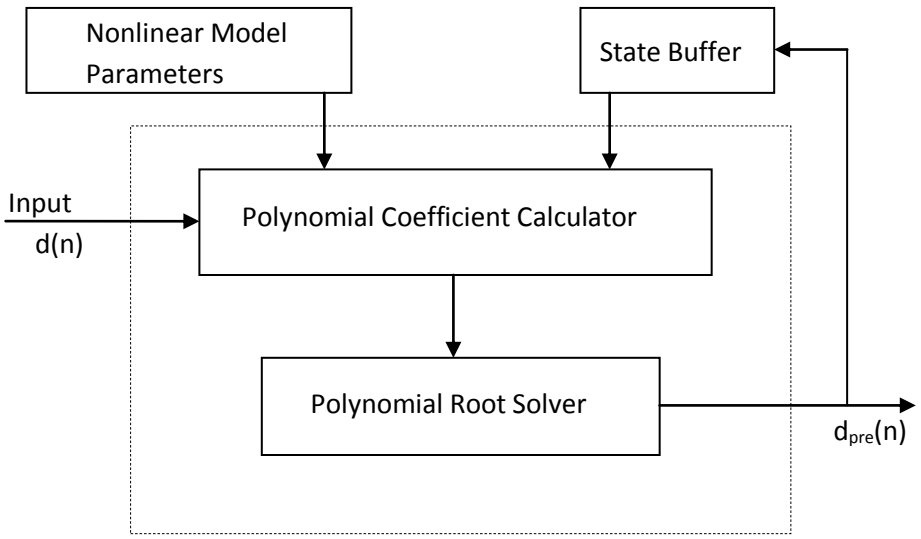



**Figure 16 Structure of pre-distortion Filter (exact inverse model used)**

The exact inverse is a nonlinear filter with parameters varying on a sample-by-sample basis as illustrated by equations (2.22) and (2.26).

For a p-th order loudspeaker model, the exact inverse is given as the root of a p-th order polynomial whose coefficients can be computed in a fashion similar to the derivation of (2.22) through (2.26). If $p$ is odd, at least one real root is guaranteed to exist. If p is even and no real

root exists, a (p-1)-th order polynomial is derived from the p-th order polynomial by differentiating relative to $d_{pre}(n)$ . The derived polynomial has order (p-1) which will be odd and is guaranteed to have a real root. The real root of the (p-1)-th order polynomial minimizes the precompensation error. If there are multiple real roots, the root with the smallest absolute value is selected.

## 2.6. Simulation & Results:

a) We will identify first a nonlinear system described below:

System used is a 10 tap linear FIR filter followed by nonlinearity given by nonlinearity given by: $b(n)=a(n)+0.5*a^3(n)$

A noise is added such that SNR=20dB.

LMS algorithm took more than 10000 samples for convergence whereas RLS algorithm took less than 5000 samples for convergence and also gave better result. The below figure shows the identification results:

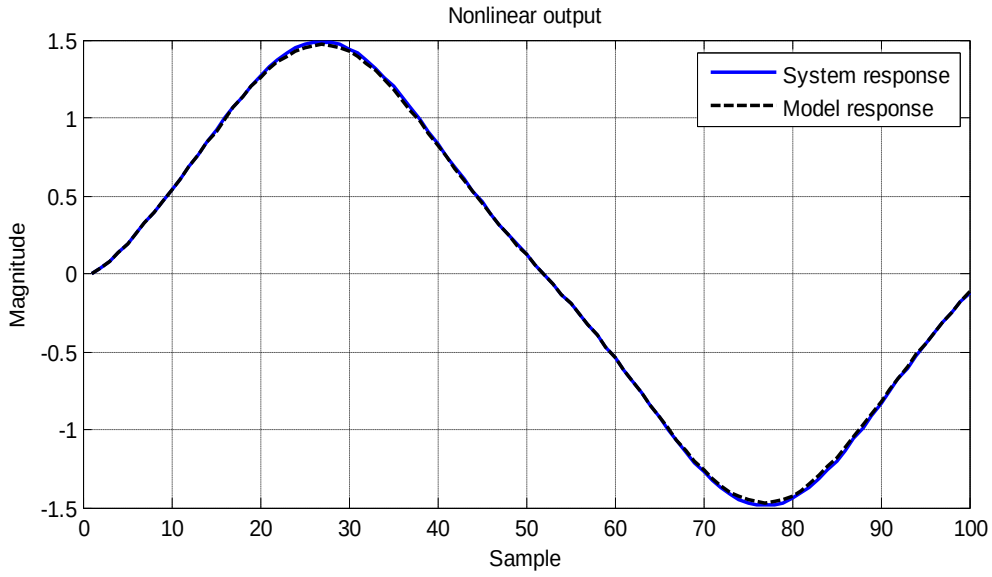



**Figure 17 Nonlinear system identification using volterra model with LMS algorithm**

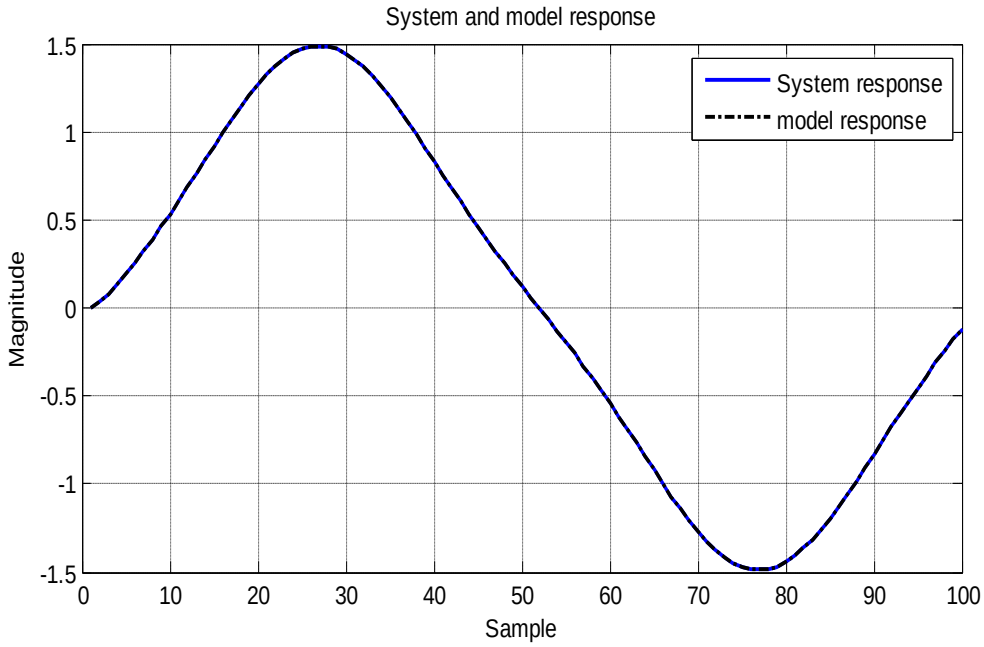



**Figure 18 Nonlinear system identification using Volterra model with RLS algorithm**

b) After identification we use the linearization technique using precompensator described in section 2.5.

The figure below shows perfect linearization and contains both input and linearized output overlapping.

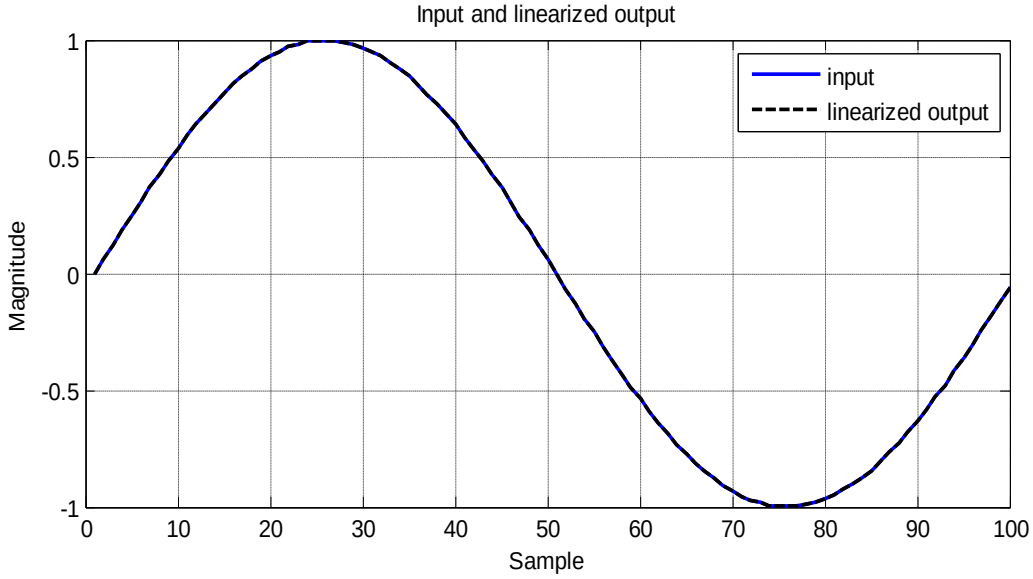



**Figure 19 Linearization of Volterra Model**

# CHAPTER 3

# WIENER AND HAMMERSTEIN MODEL IDENTIFICATION AND THEIR LINEARIZATION

# 3.1. Introduction

Many devices such as amplifiers, transmitters used in satellite channels and transducers like electrodynamic loudspeakers exhibit nonlinear behavior especially at high signal levels. Power amplifiers operating at nominal power levels are assumed linear, when driven at higher power levels, they show saturation-type nonlinearities. Small loudspeakers used in cell phones produce acceptable sound quality at low playback levels and are suitable for applications where the phone is held close to the ear. In hands-free or multimedia applications such as videophones, the loudspeaker is at about an arm's length from the user requiring higher sound levels. To reduce the nonlinear distortion of these devices, their characteristics must be modeled and inverse of these models must be computed. Many approaches have been used in the literature to address this problem. Physical models have been extensively used in characterizing amplifiers. Physical models such as the Small-Thiele model have also been developed for loudspeakers. Identification of physical models usually requires extensive measurements and does not lend itself to frequent parameter identification.

Volterra expansion [5] is a general method for modeling weak nonlinearities (i.e. saturation-type) with memory. Adaptive algorithms such as LMS and RLS [21] have been developed to determine the Volterra model parameters using the input/output measurements only [17]. A major limitation of the Volterra model is that the number of parameters grows exponentially with the model order; third or higher order models typically require several thousand parameters. Hammerstein and Wiener models consisting of the cascade of linear systems and memory-less polynomial nonlinearities are simpler models of nonlinearity and have far fewer parameters. The major disadvantage of these models is that due to the lack of memory they may not adequately model the inter-modulation distortions. To compensate for the nonlinear distortions, inverse of the nonlinear model must be found. Both feedback and open-loop solutions based on physical and Volterra models have been reported in the literature. Feedback based solutions typically use microphone, acceleration or impedance feedback. Adaptive nonlinear filters for open-loop compensation have been studied for some time, and applied in other fields as well . Most Volterra based pre-compensators use the p-th order inverse developed by Schetzen [18]. One disadvantage of the p-th order inverse is that high orders are needed to

find a proper inverse which is computationally very intensive. Exact inverse of the Volterra model with the same order as the forward model have also been reported in [7] and is computationally much more economical. The solution in [7] may not always result in a stable inverse and suboptimal pseudo-exact inverses may have to be used. Although Wiener and Hammerstein models are limited in their modeling capabilities, they are parsimonious and lend themselves to having an exact nonlinear inverse. An adaptive linearization scheme for Wiener systems is reported in [4].

In the following section we will derive the LMS algorithm for identification of Weiner and Hammerstein model. Also we will present an exact inverse for the Wiener and Hammerstein models that are fast and result in complete removal of the nonlinear distortions.

## 3.2. Block Structured Models:

Block structured models are nonlinear systems made up of interconnected linear and nonlinear subsystems. The problem in their identification is to find a model and their parameter values for each subsystem. Major constraint with block model is that the inner signals between the subsystems are not measurable. Basic building blocks for block-oriented models are a linear dynamic system and a nonlinear static transformation.

Typical block oriented models are

A Wiener model: In this a dynamic linear system is followed by a static non-linear system.

A Hammerstein model: In this a static non-linear system is followed by a dynamic linear system.

A Hammerstein-Wiener model: In this a dynamic linear system is placed between two static non-linear systems.

Block models mentioned above are important as they depict most of the practical system which exhibits some kind of non-linearities.

Many approaches have been proposed before for the identification of these structures:

- Iterative approach
- Over parameterization method
- Separable least-squares approach
- Frequency domain approach

• Stochastic method (kernel approach)

• Subspace approach

In this chapter we will go with an adaptive method of identifying the Weiner and Hammerstein model parameters using gradient descent algorithm which works well as shown in the results.

## 3.3. Weiner Model and its parameter estimation :

Figure .20 shows the block diagram of a Wiener system that consists of a linear system followed by a memory less polynomial nonlinearity. The linear system can be specified by a finite impulse response (FIR) filter $H = \{h(0), h(1), \dots \dots, h(L-1)\}$ or an IIR (pole-zero) transfer function $B(z)/A(z)$ .

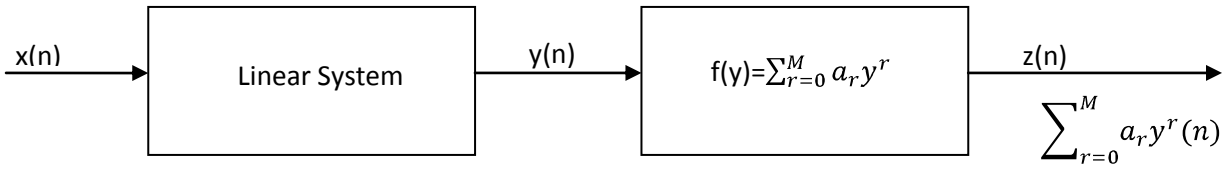



**Figure 20 Weiner System**

We derive the parameters of the Wiener models using a gradient descent algorithm. The arrangement is shown in Figure 21 The adaptation algorithm computes the parameters of the linear system and the coefficients of the polynomial nonlinearity such that the error between the model output and the desired output of the Wiener model is a minimum. We use the mean square error criterion and the gradient algorithm to perform this minimization. Assuming that the linear system is represented by a FIR impulse response, the signals at various stages of the Wiener system can be written as:

$$y(n) = \sum_{l=0}^{L-1} h(l) * x(n-l) \qquad (3.1a)$$

$$z(n) = \sum_{r=0}^{M} a_r * y^r(n) \qquad (3.1b)$$

$$z(n) \sum_{r=0}^{M} a_r * \left( \sum_{l=0}^{L-1} h(l) * x(n-l) \right)^r \qquad (3.1c)$$

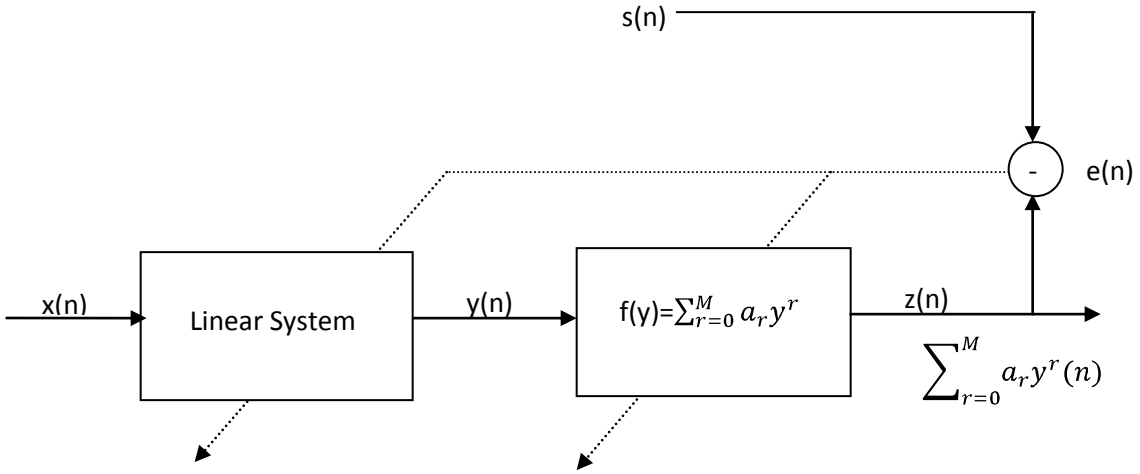



**Figure 21 Derivation of Weiner Model**

The sample error $e(n)$ at time $n$ is given by :

$$e(n) = s(n) - z(n) \qquad (3.2)$$

The total error over a frame of length N is given by:

$$E = \sum_{n=0}^{N-1} e^2(n) \qquad (3.3)$$

From (3.3) , the gradient is given as:

$$\Delta E = 2 * \sum_{n=0}^{N-1} e(n) * \Delta e(n) \tag{3.4}$$

From (3.2), we have :

$$\Delta e(n) = -\Delta z(n) \tag{3.5a}$$

$$\Delta e(n) = -\left[\frac{dz(n)}{da_0}, \quad \frac{dz(n)}{da_1}, \ldots\ldots\ldots, \frac{dz(n)}{da_M}, \frac{dz(n)}{dh(0)}, \ldots\ldots, \frac{dz(n)}{dh(L-1)}\right] \tag{3.5b}$$

Let:

$$P = [a_0, a_1, \ldots\ldots, a_M, h(0), h(1), \ldots\ldots, h(L-1)] \tag{3.6}$$

be the vector of $L + M + 1$ model parameters. Then starting from an initial guess $P_0$ and using the gradient descent algorithm with a step size $\mu$, the parameter vector $P^{(k)}$ at iteration $k$ can be updated as:

$$P^{(k)} = P^{(k-1)} + \mu * \Delta E \quad \ldots\ldots\ldots\ldots\ldots\ldots \; k = 1,2, \ldots \tag{3.7}$$

From (3.1b), we have

$$\frac{\partial z(n)}{\partial a_i} = y^i(n) \quad \ldots\ldots\ldots\ldots\ldots \; \mathrm{i} = 0,1, \ldots, \mathrm{M} \tag{3.8}$$

From (3.1a), (3.1c) we have

$$\frac{\partial z(n)}{\partial h(j)} = \sum_{r=1}^{M} r * a_r * x(n-j) * y^{r-1} \ldots\ldots\ldots\ldots\ldots \; \mathrm{j} = 0,1, \ldots\ldots\ldots, \mathrm{L}-1 \tag{3.9}$$

The gradient vector can be computed by substituting (3.8) and (3.9) into (3.5b) and then (3.5b) into (3.4). The parameter vector is then updated according to equation (3.7). The algorithm continues until some termination criterion is met such as a predetermined number of iterations is reached or the total error E is below some predetermined value E.

## 3.4. Hammerstein Model and its parameter estimation

Figure .22 shows the Hammerstein system consisting of a memoryless polynomial nonlinearity followed by a linear system. Again, the linear system can be specified by a FIR filter

$H = \{h(0), h(1), \ldots\ldots, h(L-1)\}$ or an IIR (pole- zero) transfer function $B(z) / A(z)$.

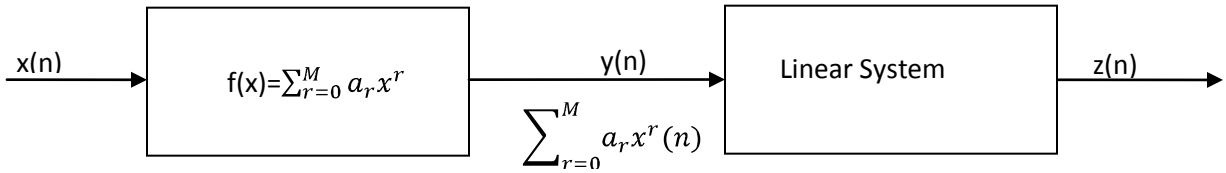



**Figure 22 Hammerstein Model**

We derive the parameters of the Hammerstein models using a gradient descent algorithm. The arrangement is shown in Figure. 23. The adaptation algorithm computes the parameters of the linear system and the coefficients of the polynomial nonlinearity such that the error between the model output and the desired output of the Hammerstein model is a minimum. We use the mean square error criterion and the gradient algorithm to perform this minimization. Assuming that the linear system is represented by a FIR impulse response, the signals at various stages of the Hammerstein system can be written as:

$$y(n) = \sum_{r=0}^{M} a_r * x^r(n) \qquad (3.10)$$

$$z(n) = \sum_{l=0}^{L-1} h(l) * y(n-l) \qquad (3.11)$$

$$z(n) = \sum_{l=0}^{L-1} h(l) * \sum_{r=0}^{M} a_r * x^r(n-l) \tag{3.12}$$

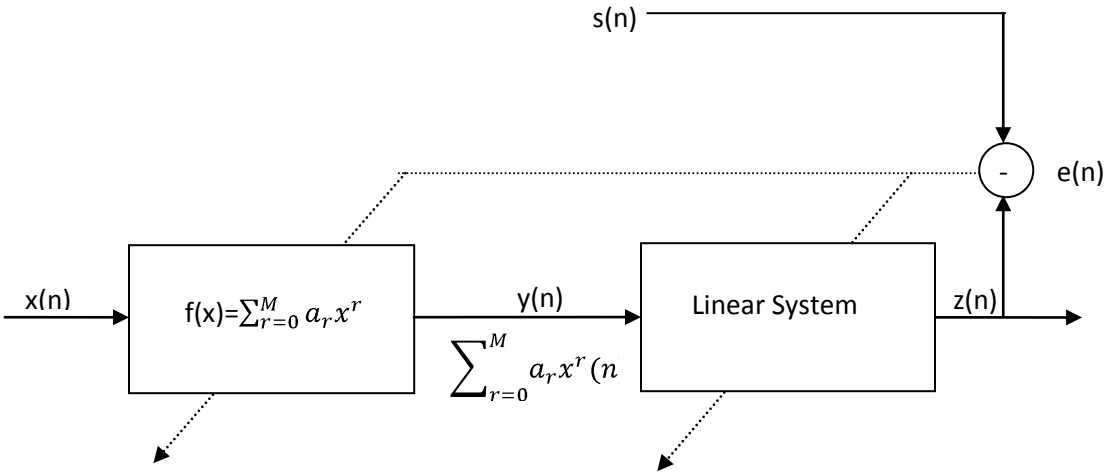



**Figure 23 Derivation of Hammerstein model**

The sample error $e(n)$ at time $n$ is given by:

$$e(n) = s(n) - z(n) \tag{3.13}$$

The total error over a frame of length N is given by:

$$E = \sum_{n=0}^{N-1} e^2(n) \tag{3.14}$$

From (3.14) , the gradient is given as:

$$\Delta E = 2 * \sum_{n=0}^{N-1} e(n) * \Delta e(n) \tag{3.15}$$

From (3.13), we have:

$$\Delta e(n) = -\Delta z(n)$$

$$\Delta e(n) = -\left[\frac{dz(n)}{da_0}, \quad \frac{dz(n)}{da_1}, \ldots\ldots\ldots, \frac{dz(n)}{da_M}, \frac{dz(n)}{dh(0)}, \ldots\ldots, \frac{dz(n)}{dh(L-1)}\right] \tag{3.16}$$

Let:

$P = [a_0 , a_1 , \dots\dots , a_M , h(0), h(1), \dots\dots , h(L-1)]$

be the vector of $L + M + 1$ model parameters. Then starting from an initial guess $P_0$ and using the gradient descent algorithm with a step size $\mu$ , the parameter vector $P^{(k)}$ at iteration $k$ can be updated as:

$$P^{(k)} = P^{(k-1)} + \mu * \Delta E \quad \dots\dots\dots\dots\dots\dots \; k = 1,2 , \dots \tag{3.17}$$

From (3.12), we have

$$\frac{\partial z(n)}{\partial a_i} = \sum_{l=0}^{L-1} h(l) * x^r (n-l) \dots\dots\dots\dots\dots \quad \text{i} = 1,2, \dots\dots\dots , \text{M} \tag{3.18}$$

and

$$\frac{\partial z(n)}{\partial h(l)} = \sum_{r=0}^{M} a_r * x^r (n-l) = y(n-l) \dots\dots\dots\dots\dots \; l = 0,1, \dots\dots\dots , L-1 \tag{3.19}$$

The gradient vector can be computed by substituting (3.18) and (3.19) into (3.16) and then (3.16) into (3.15). The parameter vector is then updated according to equation (3.17). The algorithm continues until some termination criterion is met such as a predetermined number of iterations is reached or the total error E is below some predetermined value E.

## 3.5. Inverse of the Weiner and Hammerstein Models:

Figure. 24 show the exact inverse of the Wiener model. The cascade of this system with the system in Figure.20 yields an identity system. Similarly Figure.25 shows the exact inverse of the Hammerstein system. Again we observe that the inverse always exists and by construction is stable.

In Figure. 24, the first part consists of a root solver that finds the roots of the polynomial $f(y) = x(n)$ .that are passed through the linear inverse system H to produce the pre-distorted signal $d_{pre}(n)$ . We note that this inverse always exists because we can always find the roots of the polynomial equation f(y) =x (n). If the polynomial's order is odd, there is at least one real root. The linear FIR part H may not be a minimum phase impulse response. In such cases, a

stable delayed inverse of the FIR impulse response can always be found using the QR decomposition or the FFT method.

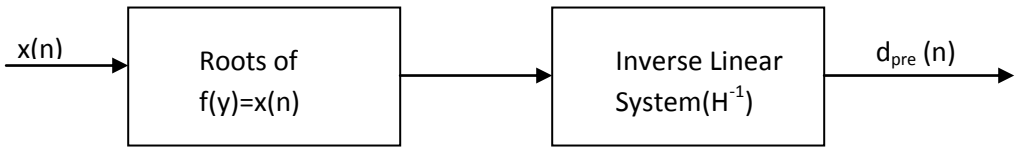



**Figure 24 Exact inverse of Weiner system**

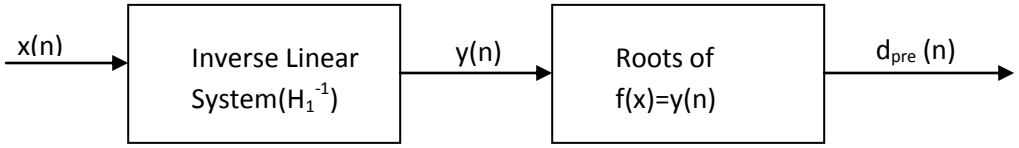



**Figure 25 Exact inverse of Hammerstein system**

The input signal $x(n)$ in Figure 25 is passed through the inverse of the linear FIR impulse response H to generate $y(n)$ which is then passed through a polynomial root solver that finds the roots of $f(d_{pre}) = y(n)$.

The roots form the pre-distorted signal $d_{pre}(\mathrm{n})$ . It will be verified that passing $d_{pre}(\mathrm{n})$ through the Hammerstein system in Figure 22 will yield back x (n) the original input signal.

## 3.6. Simulation & Results

a) Here we try to identify a given Weiner system which represents a nonlinear system using the technique described in section 3.3. The system contains a linear part which is an FIR filter with 3 taps and a third order nonlinear part in cascade with the linear part. Also noise is added after the nonlinear part to make the system look practical. For training the model random input is passed through both the system and the model.2000 input samples were used in this example.

After the model is trained a single tone signal of frequency 10 Hz was passed through the model and the response of the model was compared with the response of the actual system Figure 26.

After this a pre-compensator was placed before the model as was described in Section 3.5. The inverse FIR was designed using the inverse FFT method. We find in simulation that more the number of taps we take in inverse FIR filter better is the linearized output obtained. The input to this the pre-compensator is the input to the total system and its output is the input to the nonlinear model. The output of the nonlinear model preceded by the pre-compensator is shown in Figure 27.

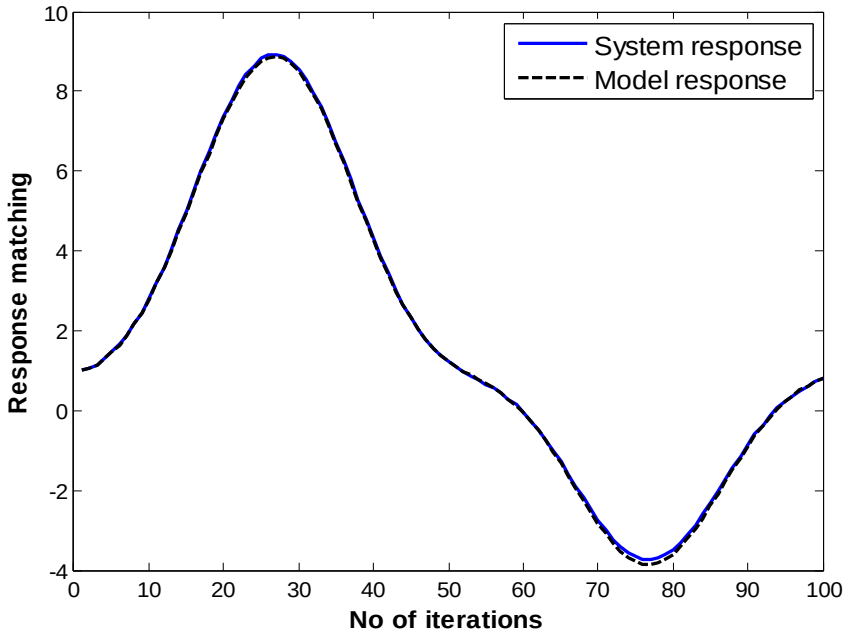
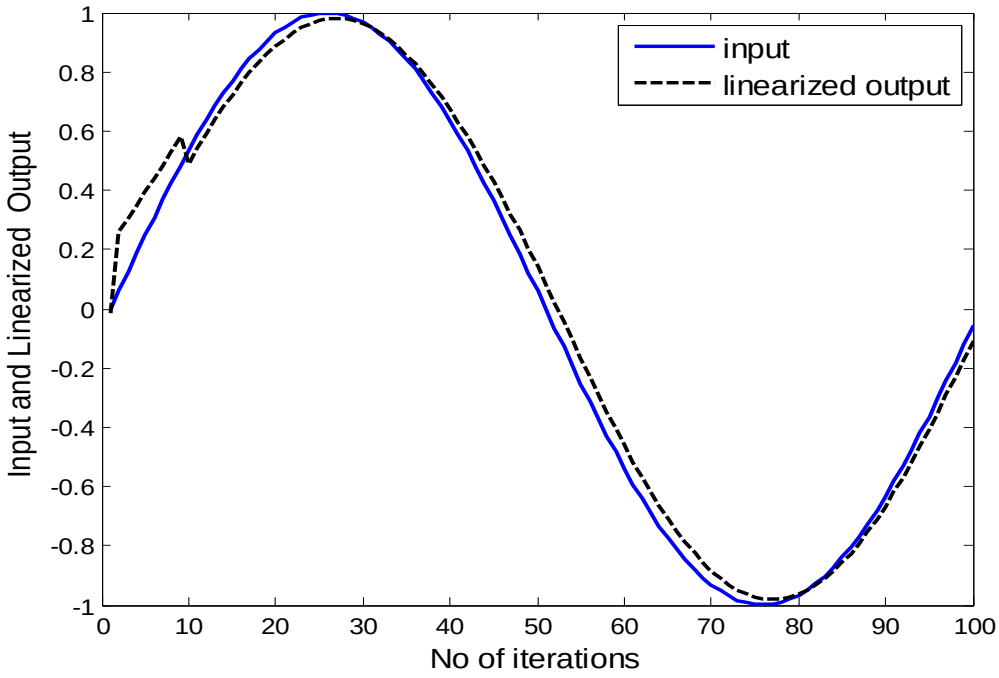
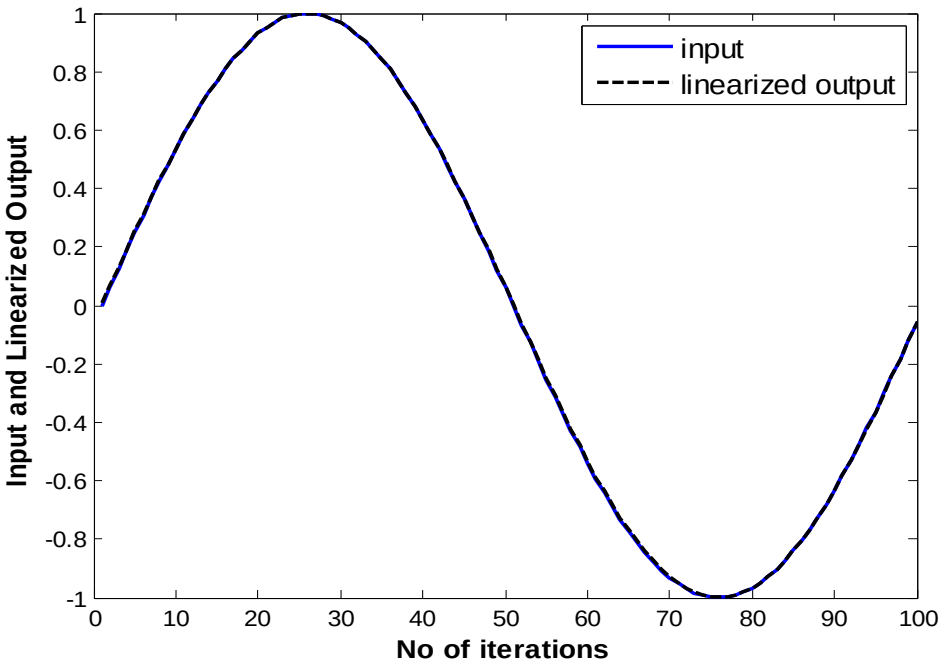



**Figure 26 System and identified model output response matching**

.




**(a)**




**(b)**

**Figure 27 Actual output and precompensated nonlinear system output matching with different length of inverse FIR filter.**
**a) 3 taps b) 24 taps**

b) Now will identify a given Hammerstein system which represents a nonlinear system using the technique described in section 3.4. The system contains a nonlinear part (third order) in cascade with a 3 tap FIR filter. Also noise is added after the linear part to make the system look practical. For training the model random input is passed through both the system and the model.2000 input samples were used in this example.SNR of 20dB is used in the simulation.

After the model is trained a single tone signal of frequency 10 Hz was passed through the model and the response of the model was compared with the response of the actual system Figure 28.

After this a pre-compensator was placed before the model as was described in Section 3.5. The inverse FIR was designed using the inverse FFT method. We find in simulation that more the number of taps we take in inverse FIR filter better is the linearized output obtained. The input to this the pre-compensator is the input to the total system and its output is the input to the nonlinear model. The output of the nonlinear model preceded by the pre-compensator is shown in Figure 29.

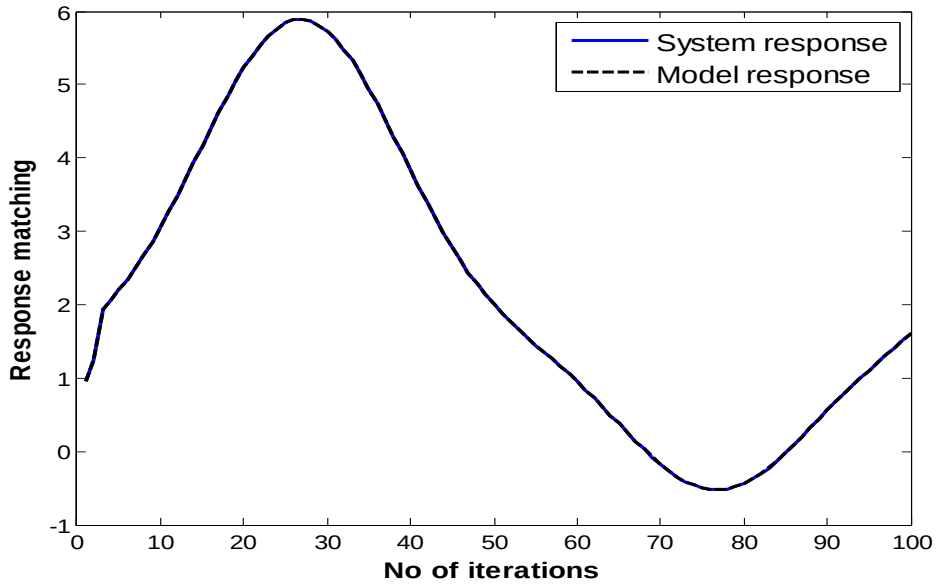
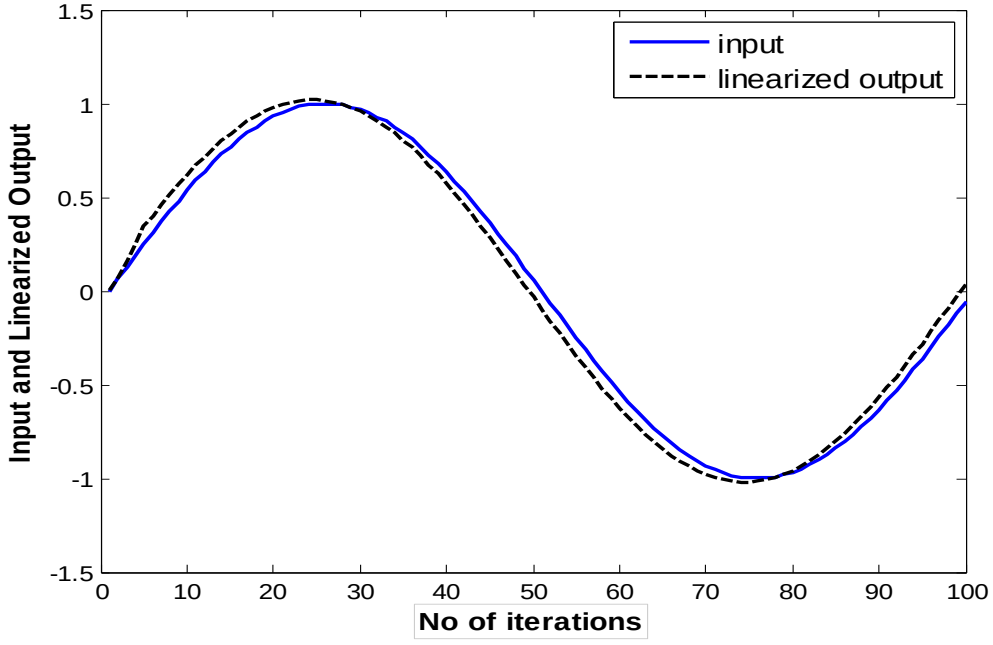
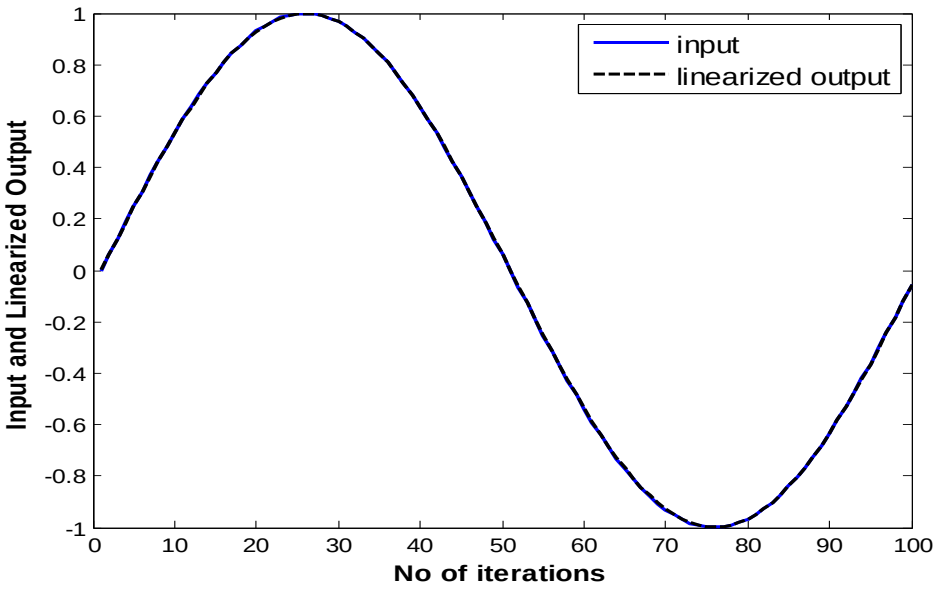



**Figure 28 System and identified model output response matching.**




**(a)**




**(b)**

**Figure 29 Actual output and precompensated nonlinear system output matching with d different length of inverse FIR filter.**

**a) 3 taps b) 24 taps**

# CHAPTER 4

# HAMMERSTEIN MODEL IDENTIFICATION WITH IIR LINEAR STRUCTURE USING GENETIC ALGORITHM

# 4.1. Introduction

System identification is a pre-requisite to analysis of a dynamic system and design of an appropriate controller for improving its performance. The more accurate the mathematical model identified for a system, the more effective will be the controller designed for it. In many identification processes, however, the obtainable model using available techniques is generally crude and approximate.

In conventional identification methods, a model structure is selected and the parameters of that model are calculated by optimizing an objective function. The methods typically used for optimization of the objective function are based on gradient descent techniques. On-line system identification used to date are based on recursive implementation of off-line methods such as least squares, maximum likely-hood or instrumental variable. Those recursive schemes are in essence local search techniques. They go from one point in the search point to another at every sampling instant, as a new input-output pair becomes available. This process usually requires a large set of input/output data from the system which is not always available. In addition the obtained parameters may be locally optimal.

Gradient – descent training algorithm are the most common form of training algorithms in signal processing today because they have a solid mathematical foundation . Gradient – descent training however leads to suboptimal performance under nonlinear conditions. Genetic algorithm has been used in many applications to produce a global optimal solution. Ths approach is a probablistic guided optimization process which simulates the genetic evolution. The algorithm cannot be trapped in the local minima as it employs a random mutation approach.In contrast to classical optimization algorithm, genetic algorithms are not guided in their search process by local derivatives. Trough coding the population with stronger fitness are identified and maintained while population with weaker fitness are removed. This process ensures that better offsprings are produced from their parents.This search process is stable and robust and can identify global optimal parameters of a system.GA has been applied tinto many diverse areas such as function optimization .image processing and system identification .

The identification of nonlinear systems is a topic which has received considerable attention over the last two decades. Generally speaking, when it is difficult to model practical systems by mathematical analysis method, system identification may be an efficient way to

overcome the shortage of mechanism analysis method. The goal of the modeling is to find a simple and efficient model which is in accord with the practical system. In many cases, linear models are not suitable to present these systems and nonlinear models have to be considered. Since there are nonlinear effects in practical systems, e.g. harmonic generation, intermediation, desensitization, gain expansion and chaos, we can infer that most control systems are nonlinear. Nonlinear models are more widely used in practice, because most phenomena are nonlinear in nature. Indeed, for many dynamic systems the use of nonlinear models is often of great interest and generally characterizes adequately physical processes over their whole operating range. Thus, accuracy and performance of the control law increase significantly. Therefore, nonlinear system identification is much more important than linear system identification. There is no common approach to nonlinear system identification, and some efficient methods of identification are only fit to specific nonlinear systems.

A simple and useful model of nonlinear systems is the Hammerstein model. Recently, Hammerstein model has been received great attention by researchers, because its structure is simple and it can effectively reflect nonlinearity of dynamic system. Several identification algorithms for the Hammerstein model have been investigated by using correlation theory, orthogonal functions, polynomials, neural networks, piecewise linear model, and so on.

Hammerstein models are composed of a static nonlinear gain and a linear dynamics part. In some situations, they may be a good approximation for nonlinear plants. The problem of identifying plants based on such a class of models has been given a great deal of interest over the last years. The basic approach is to suppose polynomial (or polygonal) for the nonlinear element of the model. Then, the identification problem turns out to be a parametric one since it consists in estimating the parameters of the model linear and nonlinear parts.

Here we will follow a different approach, we will use a FLANN structure to model the nonlinear structure as it is very useful in identifying nonlinearity, and then use genetic algorithm to identify the parameter of model linear and nonlinear part. In this chapter GA is used for simultaneously pruning of functional links and weight updation of the total parameters. While constructing an functional link artificial neural network the designer is often faced with the problem of choosing a network of the right size for the task to be carried out. The advantage of

using a reduced neural network is that it's less costly and faster in operation. However, a much reduced network cannot solve the required problem while a fully FLANN may lead to accurate solution. Choosing an appropriate FLANN architecture of a learning task is then an important issue in training neural networks. To achieve the cost and speed advantage, appropriate pruning of FLANN structure is required. Procedure for simultaneous pruning and training of weights have been carried out in subsequent sections to obtain a low complexity reduced structure

## 4.2. Genetic algorithm

In the case of deterministic search, algorithm methods such as steepest gradient methods are employed (using gradient concept), where as in stochastic approach, random variables are introduced. Whether the search is deterministic or stochastic, it is possible to improve the reliability of the results. GA's are stochastic search mechanisms that utilize a Darwin criterion of population evolution. The GA has robustness that allows its structural functionality to be applied to many different search problems. This effectively means that once the search variables are encoded into a suitable format, the GA scheme can be applied in many environments. The process of natural selection, described by Darwin, is used to raise the effectiveness of a group of possible solutions to meet an environmental optimum.

Genetic algorithms are very different from most of the traditional optimization methods. Genetic algorithms need design space to be converted into genetic space. So genetic algorithm works with coding variables. The advantage of working with a coding variable space is that coding discretizes the search space even though the function may be continuous. A more striking difference between genetic algorithms and most of the traditional optimization methods is that GA uses a population of points at one time in contrast to the single point approach by traditional optimization methods. This means that GA processes a number of designs at the same time.

### 4.2.1. GA Operations

The GA operates on the basis that a population of possible solutions, called chromosomes, is used to access the cost surface of the problem. The GA evolutionary process can be thought of as solution breeding in that it creates a new generation of solutions by crossing two chromosomes. The solution variables or genes that provide a positive contribution to the

population will multiply and be passed through each subsequent generation until an optimal combination is obtained.

The population is updated after each learning cycle through three evolutionary processes: selection, crossover and mutation. These create the new generation of solution variables. From the population a pool of individuals is randomly selected, some of these survive into the next iterations population. A mating pool is randomly created and each individual is paired off. These pairs undergo evolutionary operators to produce two new individuals that are added to the new population.

The selection function creates a mating pool of parent solution string based upon the "survival of the fittest" criterion. From the mating pool the crossover operator exchanges gene information. This essentially crosses the more productive genes from within the solution population to create an improved, more productive, generation. Mutation randomly alters selected genes, which helps prevent premature convergence by pulling the population into unexplored areas of the solution surface and add new gene information into the population.

### 4.2.2. Population Variable

A chromosome consists of the problem variables, where these can be arranged in a vector or a matrix. In the gene crossover process, corresponding genes are crossed so that there is no inter- variable crossing and therefore each chromosome uses the same fixed structure. An initial population that contains a diverse gene pool offers a better picture of the cost surface where each chromosome within the population is initialized independently by the same random process.

In the case of binary-genes each bit is generated randomly and the resulting bit-words are decoded into their real value equivalent .The binary number is used in the genetic search process and the real value is used in the problem evaluation. This type of initialization results in a normally distributed population of variables across a specific range. A GA population, P, consists of a set of N chromosomes $\{C_1 \ldots C_N\}$ and N fitness values $\{f_1 \ldots f_N\}$ where the fitness is some function of the error matrix.

$$P = \{(C_1, f_1)\ (C_2, f_2\ )\ (C_3, f_3)\ \ldots.\ (C_N, f_N)\}$$

The GA is an iterative update algorithm and each chromosome requires its fitness to be evaluated individually. Therefore, N separate solutions need to be assessed upon the same training set in each training iteration. This is a large evaluation overhead where population sizes can range between twenty and a hundred, but the GA is seen to have learning rates that evens this overhead out over the training convergence.

### 4.2.3 Chromosome selection.

The selection process is used to weed out the weaker chromosomes from the population so that the more productive genes may be used in the production of next generation. The chromosomes fitness are used to rank the population with each individual assigned a fitness value, f

$$E_i(n) = \frac{1}{M} * \sum_{j=1}^{M} e_{ji(n)}^2$$

The solution cost value $E_i$ of the ith chromosome in the population is calculated from a training block of M training signals and from this cost an associated fitness $f_i$ is assigned:

$$f_i(n) = \frac{1}{\left(1 + E_{i(n)}\right)}$$

The fitness can be considered to be the inverse of the cost but the fitness function in Eq ( ) is preferred for stability reasons, i.e. $E_i(n) = 0$

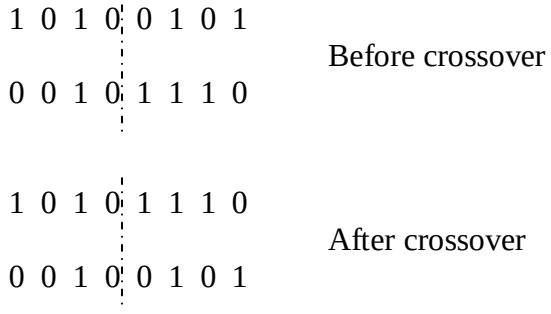
When the fitness of each chromosome in the population has been evaluated, two pools are generated, a survival pool and a mating pool. The chromosomes from the mating pool will be used to create a new set of chromosomes through the evolutional processes of natural selection and the survival pool allows a number of chromosomes to pass onto the next generation. The chromosomes are selected randomly from the two pools but biased towards the fittest. Each

chromosome may be chosen more than once and the fitter chromosomes are more likely to be chosen so that they will have a greater influence in the new generation of solutions.

### 4.2.4 Gene Crossover

The crossover operator exchanges gene information between two selected chromosomes. This operation aims to improve the diversity of the solution vectors. The pair of chromosomes, taken from the mating pool, becomes the parents of the two offspring chromosomes for the new generation.

In the case of a binary crossover operation the least significant bits are exchanged between corresponding genes of the two parents. For each gene- crossover a random position along the bit sequence is chosen and then all of the bits right of the crossover point is exchanged. In Figure 30 (a) , which shows a single point crossover , the fifth position is randomly chosen, where the first position corresponds to the left side. The bits from the right of the fourth bit will be exchanged. Figure 30(b) shows a two point crossover in which two points are randomly chosen and the bits in between them are exchanged. At the start of learning process the extent of crossing over the whole population can be decided allowing the evolutionary process to randomly select the individual genes. The probability of a gene crossing, P (crossing), provides a percentage estimate of the genes that will be affected within each parent. P (crossing) = 1 allows all the gene values to be crossed and P (crossing) = 0 leaves the parents unchanged, where a random gene selection value, $\omega \in \{1,0\}$ is governed by this probability of crossing.




**(a)**

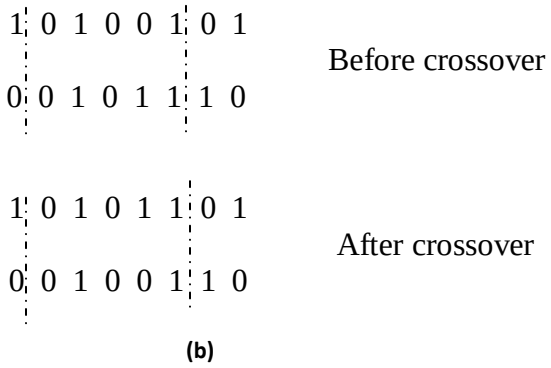



**Figure 30 Gene crossover (a) Single point crossover (b) Double point crossover**

The crossover does not have to be limited to this simple operation. The crossover operator can be applied to each chromosome independently, taking different random crossing points in each gene. This operation would be more like grafting parts of the original genes onto each other to create the new gene pair. All of a chromosome's genes are not altered within a single crossover. A probability of gene-crossover is used to randomly select a percentage of the genes and those genes that are not crossed remain the same as one of the parents.

### 4.2.5 Chromosome Mutation

The last operator within the breeding process is mutation. Each chromosome is considered for mutation with a probability that some of its genes will be muted after the crossover operation. A random number is generated for each gene, if this value is within the specified mutation selection probability, P(mutation), the gene will be mutated. The probability of mutation occurring tends to be low with around one percent of the population genes being affected in a single generation. In the case of a binary mutation operator, the state of the randomly selected gene-bits is changed, from zero to one or vice-versa.

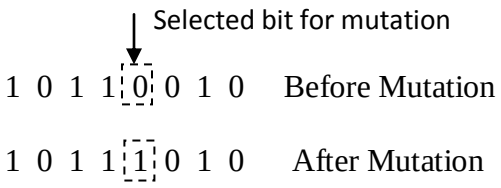



**Figure 31 Mutation operation in GA**

A simple genetic algorithm treats the mutation as a secondary operator with the role of restoring lost genetic materials. For example consider the following population having four eight-bit strings.

0 1 1 0 1 0 1 1
0 0 1 1 1 1 0 1
0 0 0 1 0 1 1 0
0 1 1 1 1 1 0 0

All the four strings have a zero in the left most bit position. If the true optimum solution requires a one in that position, then neither reproduction nor crossover operator will be able to create a one in that position. Only mutation operation can change that zero to one.

## 4.3. Parameters OF GA.

There are some parameters value required for GA. To get the desired result these parameters should be chosen properly.

(a) **Crossover and Mutation Probability**:

There are two basic parameters of GA - crossover probability and mutation probability.

**Crossover probability**: This probability controls the frequency at which the crossover occurs for every chromosome in the search process. This is a number between (0, l) which is determined according to the sensitivity of the variables of the search process. The crossover probability is chosen small for systems with sensitive variables. If there is crossover, offspring are made from parts of both parent's chromosome. Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old populations survive to next generation.

**Mutation probability**: This parameter decides how often parts of chromosome will be mutated. If there is no mutation, offspring are directly copied from crossovered ones without any change. If mutation is performed, one or more parts of a chromosome are changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed. Mutation generally prevents the GA from falling into local extremes. Mutation should not occur very often, because then GA will in fact change to random search.

(b) Other Parameters. There are also some other parameters in GA. One important parameter is population size.

**Population size**: How many chromosomes are in population in one generation. If there are too few chromosomes, GA has few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to use very large populations because it does not solve the problem faster than moderate sized populations.

## 4.4. Pruning of FLANN structure along with parameter estimation using GA.

In this Section a new algorithm for simultaneous training and pruning of weights using binary coded genetic algorithm is studied. Such a choice has lead to effective pruning of branch and update of weights. The pruning strategy is based on the idea of successive elimination of less productive paths (functional expansions) and elimination of weights from the FLANN structure. As a result the overall architecture of the FLANN based model is reduced which in turn reduces the corresponding computational cost associated with the model without sacrificing the performance. Various steps involved in this algorithm are dealt in this section.

**Step 1**- Initialization in GA:

A population of M chromosomes is selected in GA in which each chromosome constitutes (T×E)× (L+1) + L×W number of random binary bits where the first T×E number of bits are called Pruning bits (P) and the next T×E×L bits represent the weights associated with various branches (functional expansions) of the FLANN model and the last L×W bits represents the weight associated with the linear part of the model placed after the FLANN in the Hammerstein model .Again (T) represents the number of inputs and E represents the number of expansions specified for each input. Thus each chromosome can be schematically represented as shown in the Fig. (32).

A pruning bit (p) from the set P indicates the presence or absence of expansion branch which ultimately signifies the usefulness of a feature extracted from the time series. In other words a binary 1 will indicate that the corresponding branch contributes and thus establishes a physical connection whereas a 0-bit indicates that the effect of that path is insignificant and hence can be neglected.

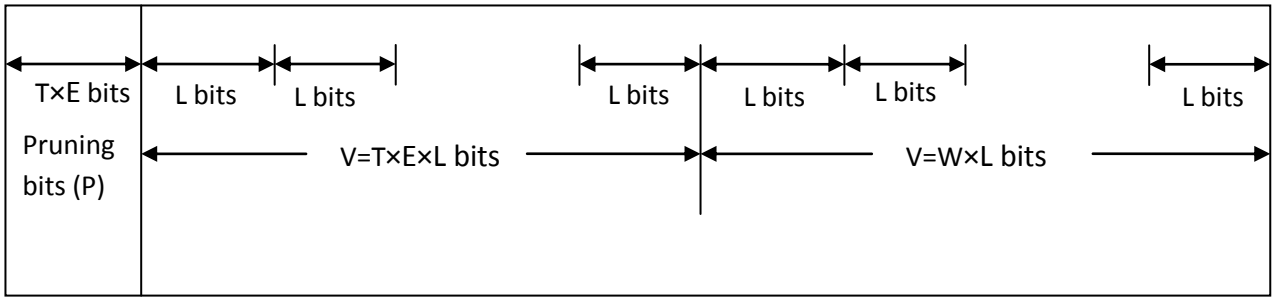



**Figure 32 Bit allocation scheme for pruning and weight updating**

**Step 2**- Generation of input training data:

K (≥500) number of signal samples is generated.

**Step 3**- Decoding:

Each chromosome in GA constitutes random binary bits. So these chromosomes need to be converted to decimal values lying between some ranges to compute the fitness function. The equation that converts the binary coded chromosome in to real numbers is given by:

$$RV = R_{min} + \left\{ (R_{max} - R_{min}) \Big/ (2^L - 1) \right\} * DV$$

where $R_{max}$, $R_{min}$,RV,DV represents the minimum range, maximum range, decimal and decoded value of an L bit coding scheme representation.The first T×E number of bits is not decoded since they represent pruning bits.

Step 4 – Compute the estimated output

At nth instant the estimated output of the neuron can be computed as

$$y(n) = \sum_{i=1}^{T} \sum_{j=1}^{E} \phi_{ij}(n) * W_{ij}^{m}(n) * P_{ij}^{m}(n) + b^{m}(n)$$

where $\phi_{ij}$(n) represents jth expansion of the ith signal sample at the nth instant. $W_{ij}^{m}(n)$ and $P_{ij}^{m}(n)$ represents the jth expansion weight and jth pruning weight of the ith signal sample for mth chromosome at kth instant. $b^{m}(n)$ corresponds to the bias value fed to the neuron.

This is then passed through the linear part of the model to get the estimated output.

Step 5 – Calculation of cost function:

Each of the desired output is compared with corresponding estimated output and K errors are produced. The mean square corresponding to m-th chromosome is determined by using the relation:

$$MSE(m) = \sum_{k=1}^{K} \frac{e_k^2}{K}$$

This is repeated for M times.

Step 6 – Operations of GA:

Here the GA is used to minimize the MSE. The crossover, mutation and selection operators are carried out sequentially to select the best M individuals which will be treated as parents in the next generation.

Step 7 – Stopping Criteria:

The training procedure will be ceased when the MSE settles to a desirable level. At this moment all the chromosomes attain the same genes. Then each gene in the chromosome represents an estimated weight.

## 4.5. Simulation & Results

a) In this example a static system is used. Nonlinearity given by:

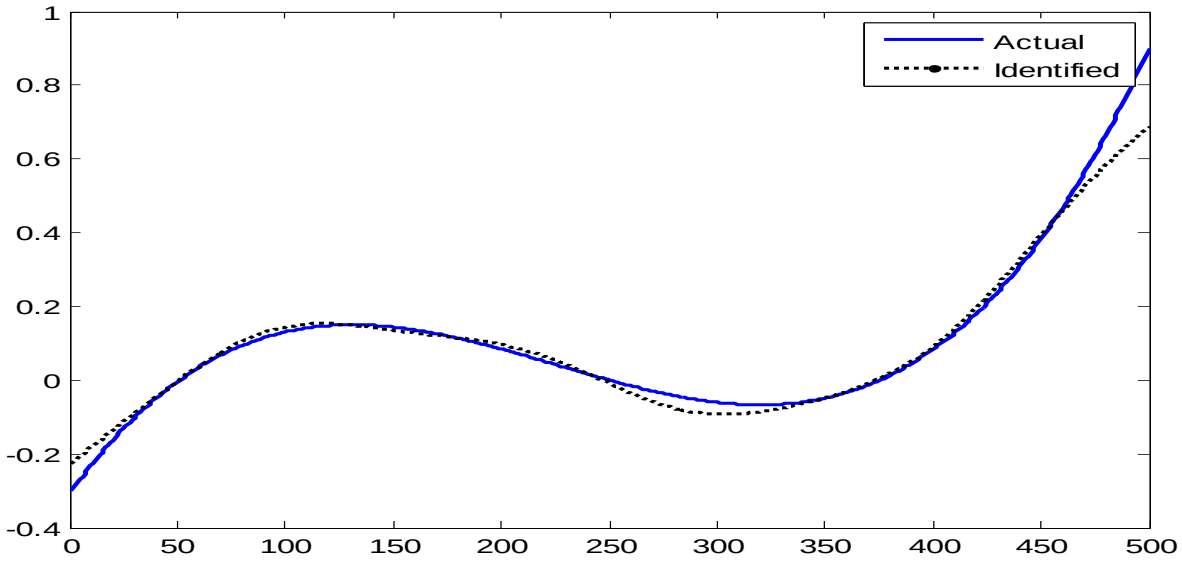
b=(a.^3)+0.3*(a.^2)-0.4*a;

In the FLANN structure the expansions used are x, sin (n*pi*x), cos (n*pi*x) where x is the input and

n =0, 1, 2, 3,4,5,6.

Probability of crossover used is pc=0.8 and that of mutation is pm=0.1.

The identification result using the structure as shown in Figure 33 is shown below:




Pruned weights come out to be:

1 0 1 1 1 0 1 0 0 1 0 0 0 0 0 0

The normalize mean square error plot given by NMSE= 10*log10 ( $\frac{\text{error}}{\max(\text{error})}$ ) is shown below:

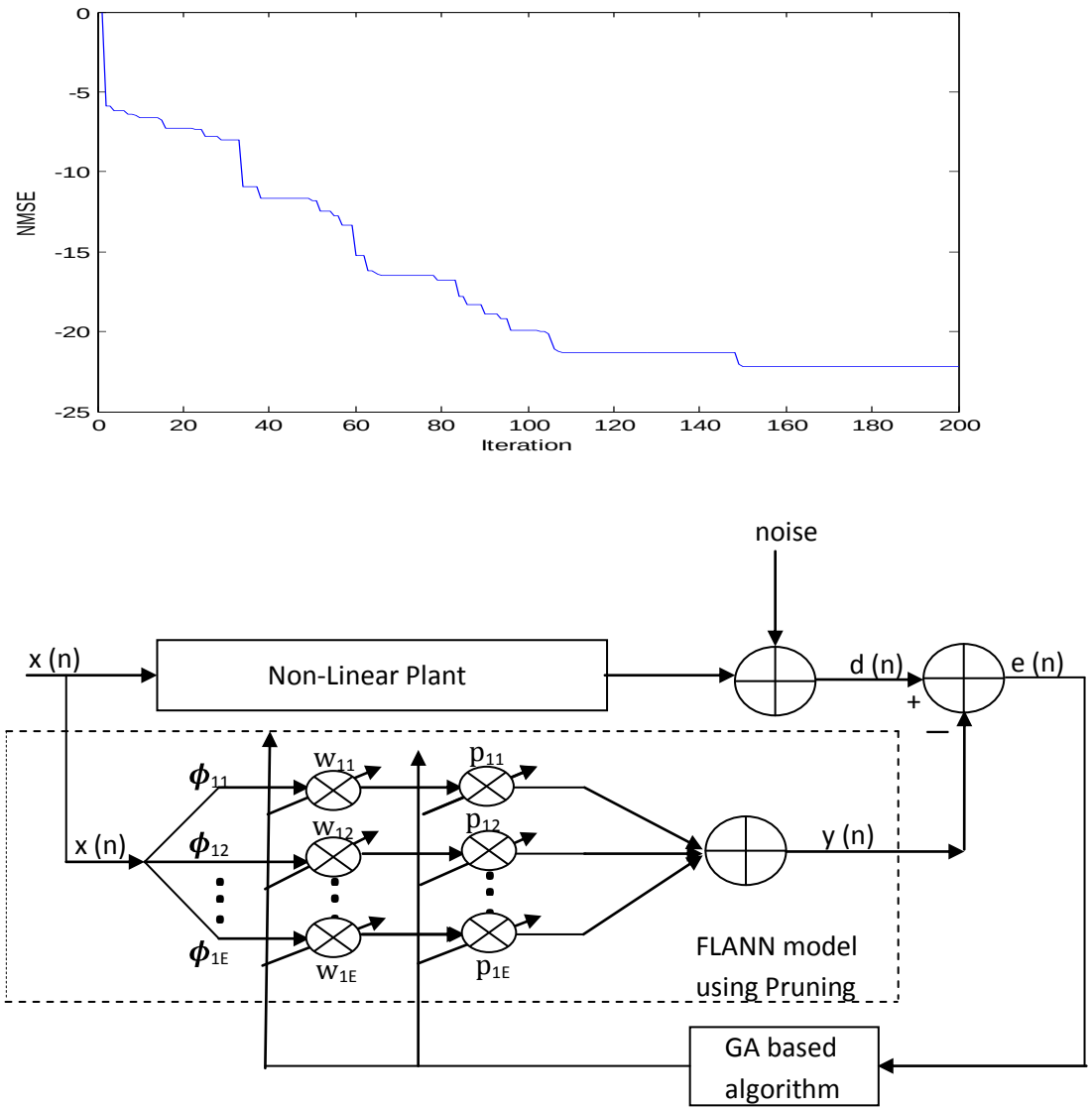



**Figure 33 FLANN based static nonlinear system identification model showing updating weight and pruning weights.**

b) In this example a dynamic system with static nonlinearity is identified. Nonlinearity is given by:

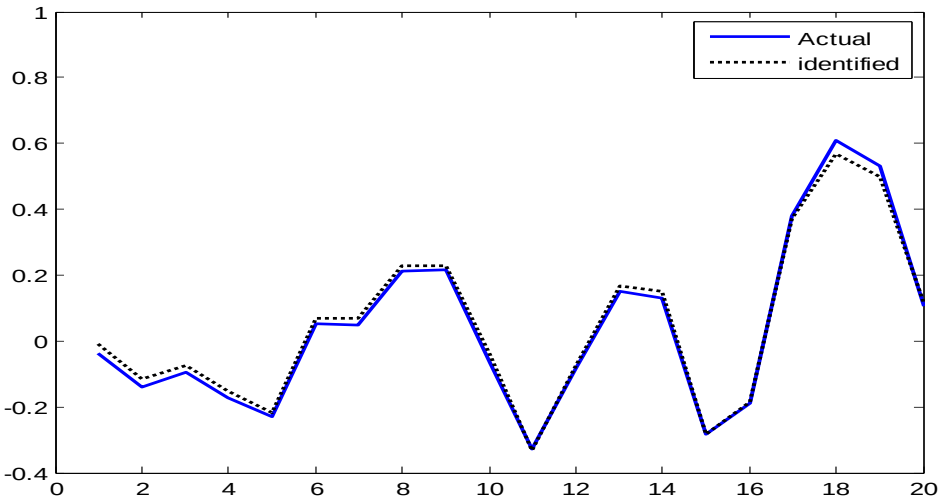
b = - 0.1*(a.^3) + 0.2*(a.^2) + a;

In the FLANN structure the expansions used are x, sin (n*pi*x), cos (n*pi*x) where x is the input and

n =0, 1, 2, 3.

Probability of crossover used is pc=0.8 and that of mutation is pm=0.1.

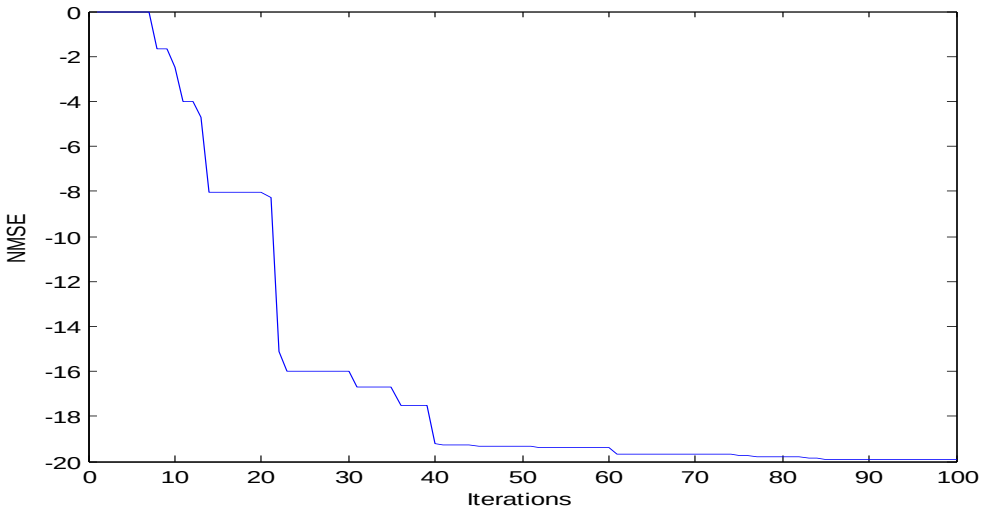
The identification result using the structure as shown in Figure 34 is shown below:




Pruned weights come out to be:

1 0 0 1 1 0 0 0 0 1

The normalize mean square error is shown below:



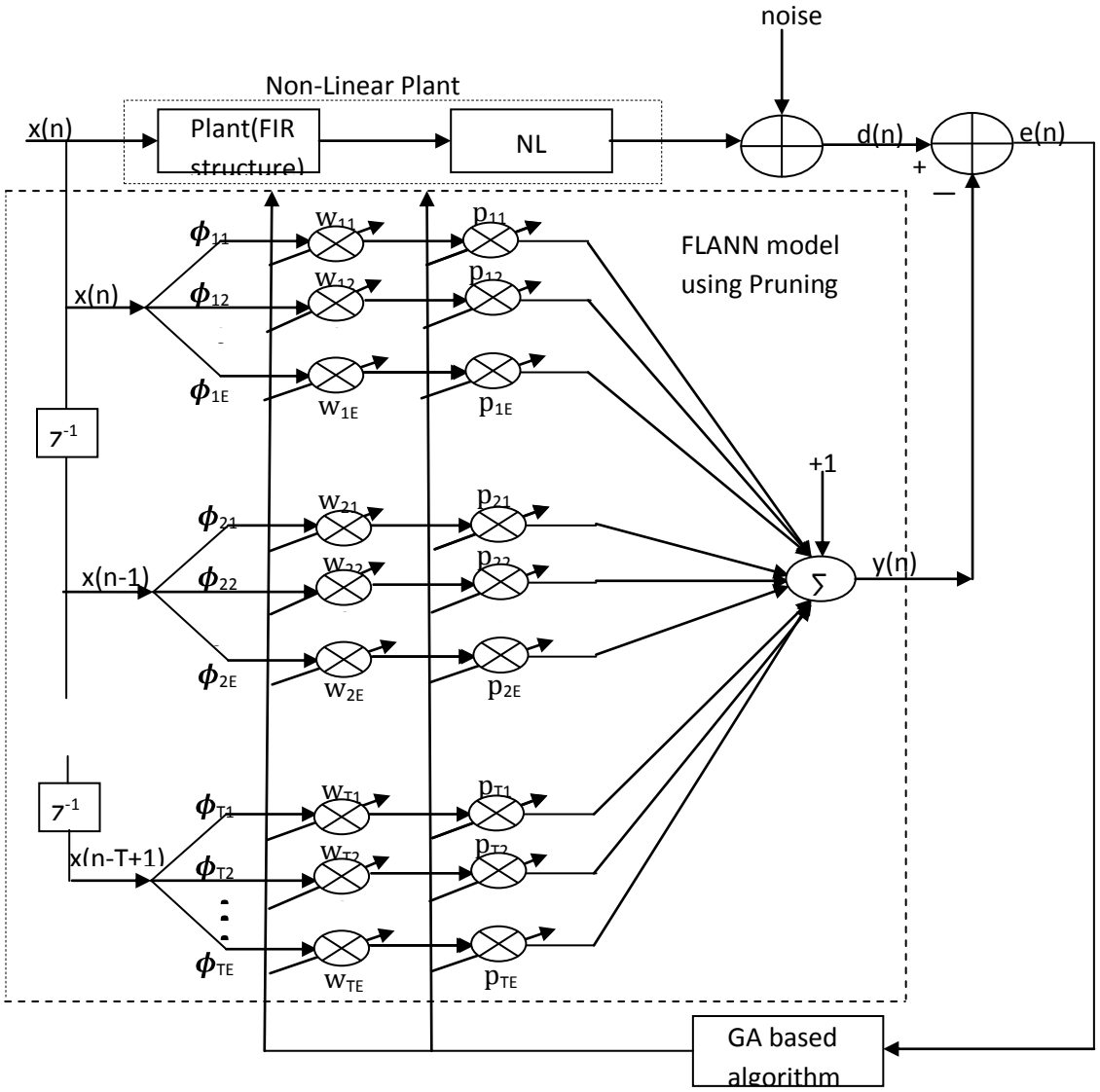



**Figure 34 GA used in identification and pruning of FLANN structure and identification of weights for dynamic plant**

3) In this example a Hammerstein type system with static nonlinearity and IIR linear part is used.

Nonlinearity is given by:

b = a + 0.5*(a. ^3);

Linear structure is given by:

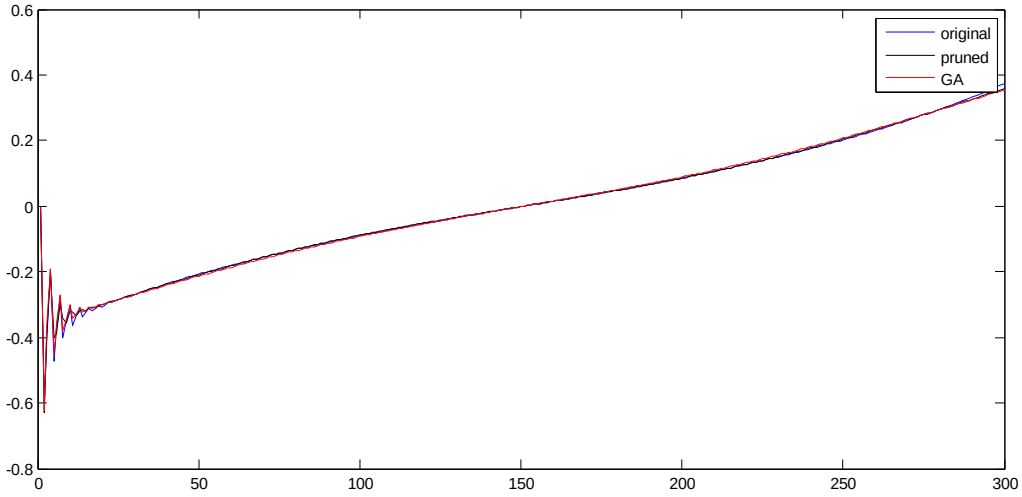
Forward network: B=[ 0.4 0.2 ];

and Reverse network: A=[ 0.8 0.6 ];

In the FLANN structure the expansions used are x, sin (n*pi*x), cos (n*pi*x) where x is the input and

n =0, 1,2,3,4, 5, 6.

The normal FLANN structure without pruning is also used and the results are compared. The pruning result is very close to the normal structure and reduces the hardware requirement to a great level. Probability of crossover used is pc=0.8 and that of mutation is pm=0.1.The identification using the pruned structure and normal structure as shown in Figure 35 is shown below:




Pruned weights come out to be:

1 0 0 1 0 0 0 0 0 0 0 0 0 0 0

4) This is same like the previous example but with different linear and nonlinear structure.

Nonlinearity is given by:

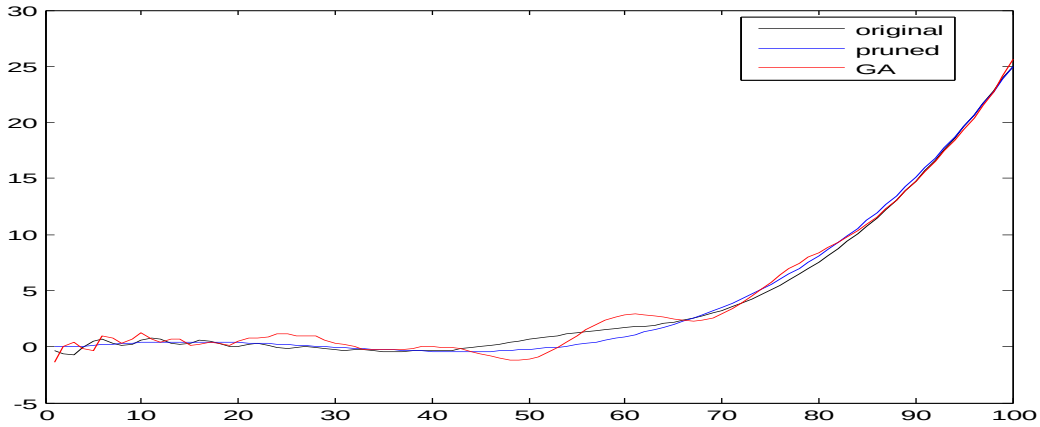
b = a + 3*(a.^2) + 2*(a.^3);

Linear structure is given by:

Forward network: B= [1 .5 .4 2];

and Reverse network: A=[.5 -.4 -.26 -.03];

Probability of crossover used is pc=0.85and that of mutation is pm=0.1.The identification using the pruned structure and normal structure as shown in Figure 35 is shown below:




Pruned weights comes out to be:

1 0 1 1 1 1 0 0 0 1 0 0 0 0 0

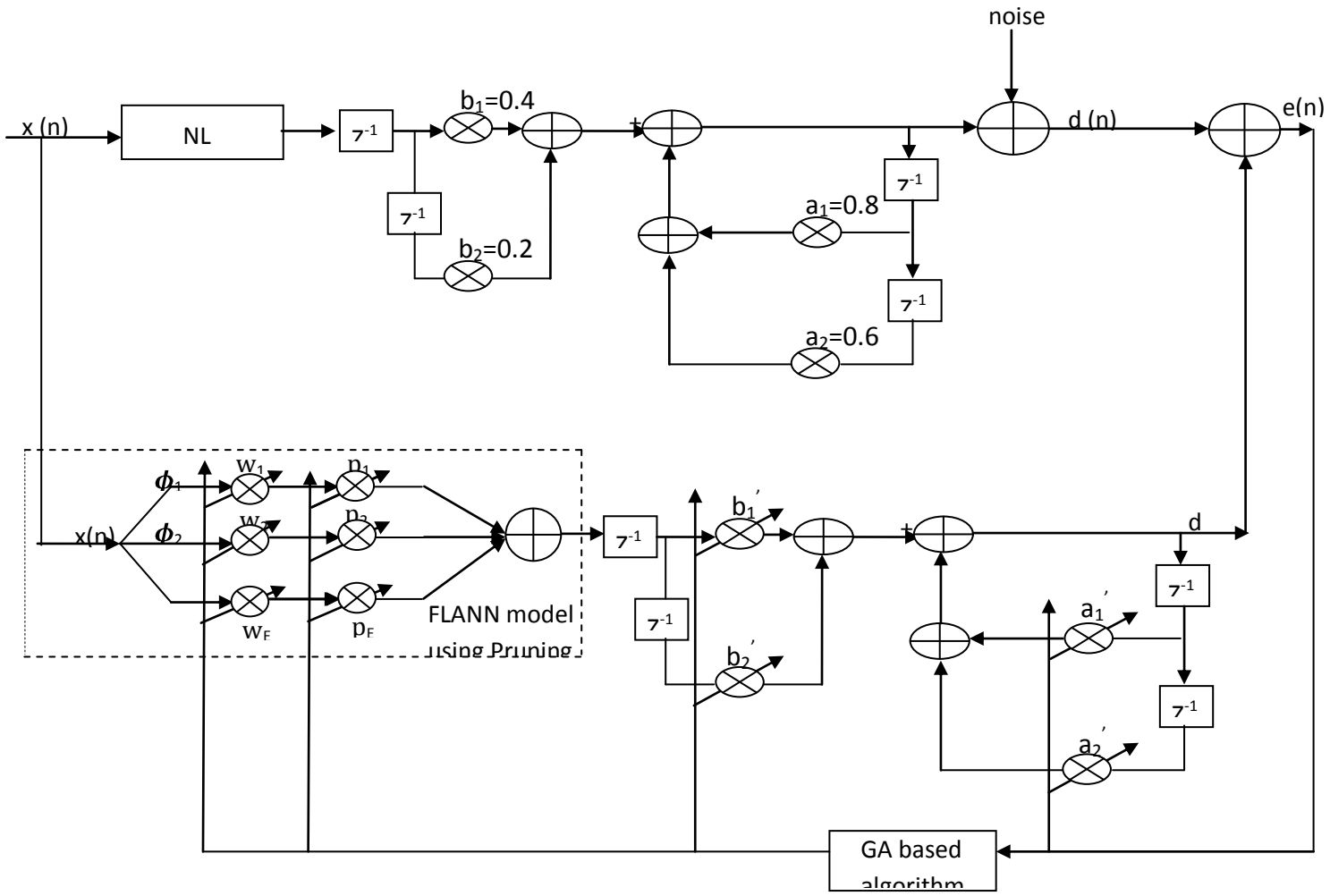



**Figure 35 FLANN based static nonlinear system identification model showing updating weight and pruning weights**

# CONCLUSIONS

1) In this a modified backpropogation learning algorithm for MLP and FLANN was discussed and the resulting network was called WNN and WFLANN respectively. These were then used in function approximation and channel equalization and results show its effectiveness in dealing with outliers present in training sequence.
2) Volterra series expansion was studied and then these models were applied in nonlinear system identification whose weights were applied using both LMS and RLS equation and results showed that RLS needs much smaller training pattern than LMS and thus is very useful.

   The resulting model parameters were then used to find the coefficients of the polynomial equation whose root are named as precompensator output, which when applied to nonlinear system gives a linear output. This method is very efficient as it's very simple and easy to design.
3) Two very useful block models namely Weiner model and Hammerstein model were studied and its parameters were derived by very simple LMS algorithm. Also there linearization was performed and results showed that the linear inverse is better when the number of taps in the inverse filter was increased.
4) Finally genetic algorithm was used for identification of Hammerstein model in which linear part is an IIR structure. Genetic algorithm could easily identify such complex structure. Pruning was also applied to the FLANN structure used for modeling nonlinearity and results proved that without lose in quality it reduces the number of expansions required to a great level and thus reduces the implementation cost and complexity.

Thus this work gives very good scope in various applications were nonlinearities are to be dealt with.

# REFERENCES


[1]. Jer-Guang Hsieh, Yih-Lon Lin, and Jyh-Horng Jeng, "*Preliminary Study on Wilcoxon Learning machines*", IEEE Trans. on neural networks, Vol. 19, No. 2, February 2008.

[2]. Jagdish C. Patra, Ranendra N. Pal, B. N. Chatterji, and Ganapati Panda, "*Identification of Nonlinear Dynamic Systems Using Functional Link Artificial Neural Networks*", IEEE Trans. on systems, Vol. 29, No. 2, April 1999.

[3]. Tomohiro Hachino, Katsuhisa Deguchi and Hitoshi Takata, "*Identification of Hammerstien Model Using Radial Basis Function Networks and Genetic Algorithm*", 5th Asian Control Conference 2004.

[4]. Khosrow Lashkari, Akshaya Puranik, "*Exact Linearization of Wiener and Hammerstien System*", IEEE 2005.

[5]. V.John Mathews, "*Polynomial Signal Processing*", Wiley Inter-Science,2000.

[6]. Hazem M. Abbas , Mohamed M. Bayoumi ,"*An adaptive evolutionary algorithm for Volterra system identification*",ELSEVIER 2005.

[7]. Khosrow Lashkari, "*High Quality Sound from Small Loudspeakers Using the Exact Inverse*", IEEE 2004.

[8]. Arthur J. Redfern and G. Tong Zhou,"A *Root Method for Volterra System Equalization*" ,IEEE Signal Processing Letters, Vol. 5, No. 11, November 1998.

[9]. John Tsimbinost JS and Kenneth V. Lever," *The Computational Complexity of Nonlinear Compensators based on the Volterra Inverse*", IEEE 1996.

[10]. Chandrakumar Bhumireddy and C. L. Philip Chen ,"*Genetic Learning of Functional Link Networks*", IEEE 2003.

[11]. Nader Sadegh," *A Perceptron Network for FunctionalIdentification and Control of Nonlinear Systems*", IEEE Transactions On Neural Networks, Vol. 4, No. 6 , November 1993.

[12]. Khosrow Lashkari, "*A Modified Volterra-Wiener-Hammerstein Model for Loudspeaker Precompensation*", IEEE 2005.

[13]. K.S.Narendra and P.G.Gallman," *An Iterative Method for the IdentScation of Nonlinear Systems Using a Hammerstein Model* ", IEEE Trans. on Automatic Control.

[14]. W. Lin and P.X. Liu,"*Hammerstein model identification based on bacterial foraging*", ELECTRONICS LETTERS 9th November 2006 Vol. 42 No. 23.

[15]. H.-X. Li," *Identification of Hammerstein models using genetic Algorithms*", IEE, 1999

[16]. Kristinn Kristinsson and Guy A. Dumont, "*System Identification and Control Using Genetic Algorithms*", IEEE Transactions On Systems, Man, And Cybernetics, Vol. 22, No. 5, September 2001.

[17]. V. J. Matthews, "*Adaptive Polynomial Filters*", IEEE SP magazine, Vol. 8, No. 3, pp. 10-26, July 1991.

[18]. M. Schetzen, "*Theory of pth-order Inverses of Nonlinear Systems*", IEEE Trans. On Circuits and Systems, CAS-23, No. 5, May 1976, pp. 285-291.

[19]. Martin T. Hagan, Howard B. Demuth and Mark Beale,"*Neural Network Design*".

[20]. Kumpati S. Narendra, and Kannan Parthasarathy," *Identification and Control of Dynamical Systems Using Neural Networks*", IEEE Transactions On Neural Networks. Vol. 1. No. 1. March 1990

[21]. Bernard Widrow and Samuel D.Stearns,"Adaptive Signal Processing",Pearson education.