1-31-1991

Comparative study of prediction gain based on neural network architecture

Prashant M. Shah
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

COMPARATIVE STUDY OF PREDICTION GAIN BASED ON NEURAL NETWORK ARCHITECTURE

Prashant M. Shah, M. S. E. E., New Jersey Institute of Technology
Thesis Advisor: Dr. C.N.Manikopoulos

This thesis describes the Neural Network approach to design predictor using Delta and Generalized Delta Rule. The predictor is designed by supervised training based on the typical sequence of pixel values. Neural Network is used to find the coefficients of the predictor. Both 1-D and 2-D scheme of the pixels as well as linear and non-linear correlations are used to find the coefficients by training. Different combinations of pixels are used to find the "best" combination among the order of the predictor.

Comparative Study of Prediction Gain based on Neural Network Architecture

by

Prashant M. Shah

Thesis submitted to the Faculty of the Graduate School
of the New Jersey Institute of Technology in partial
fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering
1991

Approval Sheet

Title of Thesis: **Comparative Study of Prediction Gain based on Neural Network Architecture**

Name of Candidate: Prashant M. Shah
Master of Science in Electrical Engineering. 1991

Thesis & Abstract Approved
by the Examining Committee:

Dr. Constantine N. Manikopoulos. Advisor — Date
Associate Professor
Department of Electrical and Computer Engineerig

Dr. Irving Wang — Date
Assistant Professor
Department of Electrical and Computer Engineering

Dr. George E. Antoniou — Date
Visiting Professor
Department of Electrical and Computer Engineering

New Jersey Institute of Technology, Newark, New Jersey.

VITA

Name: Prashant M. Shah

Permanent address:

Degree and date to be conferred: M.S.E.E., 1991.

Date of birth:

Place of birth:

Secondary education: Alembic Vidyalaya, Baroda, India

Collegiate institutions attended	Dates	Degree	Date of Degree
M. S. University, Baroda, India	1982-86	B.S.E.E.	December, 1986
New Jersey Institute of Technology	1988-91	M.S.E.E.	January, 1991

Major: Electrical and Computer Engineering

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There have been many impressive demonstrations of artificial Neural Network capabilities: a network has been trained to recognize human face, recognize handwritten characters and is used for voice and image compression[2][1]. Artificial neural net models have been studied for many years in the hope of achieving human-like performance in the fields of the speech and image recognition. Potential applications are those where human intelligence functions effortlessly and conventional computation has proven cumbersome or inadequate[1].

Adaptive Neural Network Predictor(ANNP) is designed using Delta Rule and Generalized Delta Rule[3]. It is used for both 1-D and 2-D Image data as well as for linear and non-linear inputs. The predictor used here uses the supervised training based on a typical sequence of pixel values. In predictive coding the dependence inherent in the data can be removed by good predictor and transforms the original data into a form such that successive data symbols are nearly independent of each other[6]. Predictive coding is applied for Image data compression. A fixed predictor is justified if the statistics of the data are known or available in advance and if the data from the source are stationary. The adaptive predictors provide solutions that are practical when data statistics are

[1]The numbers in square brackets indicate corresponding references in the bibliography.

unknown or nonstationary. The predictor at the receiver operates in exactly the same manner as the one at the transmitter side, and updating of its prediction rule can be done synchronously with that of the transmitter[5, 8, 10].

The approach here is to use non-linear terms in the inputs. Using the delta-rule to find the coefficients of the polynomial non-linearity in the inputs, assures that we can find the global minimum to the Least Mean Square problem. The training time for the back-propagation Neural Network is considerably higher than that of the Perceptron or Functional Link Neural Network. As the polynomial order is increased the weights associated with the higher order terms are insignificant. The Neural Network consider here in all cases is the feed-forward network[1, 3].

Using ANNP, the "best" input pixel combination for each order is found. The "best" combination is that which has the highest gain within that order, so minimum bits are required to transmit the image. Simulation results show that the vertical correlation is higher than the horizontal correlation, i.e. the weight associated with the pixel above the predicted one is higher than that with the pixel just before the predicted one.

The usefulness of data compression arises in storage and transmission of images, where the aim is the memory of storage and bandwidth for transmission. Image data compression methods can be classified in two basically different categories[13]. In the first category are those methods which exploit redundancy in the data. In the second category, compression is achieved by an energy preserving transformation of the given image into another array such that maximum information is packed into minimum number of samples. Here, we use the first category using predictive coding.

The main objective behind this work is to study and evaluate the per-

formance of the ANNP. While there is no specific method available to solve the non-linear prediction, neural net solve this problem easily. It has been shown that the ANNP gives much more higher PSNR than the simple linear-predictor. Since knowledge of the Neural Network Architecture is essential for an understanding of the rest of the work, for convenience of the reader, this thesis is organised into four parts. The first part, Chapter 2, reviews the architecture of the Neural Network used here[1, 2,11]. The second part, Chapter 3 describes the learning algorithm used here to developed the ANNP and it also describes the 1-D linear prediction algorithm[7, 9]. The third part, Chapter 4 describes the simulator and set of experiments performed with different ANNP. The last part, Chapter 5 presents conclusion based on the findings obtained and suggests directions for further research.

Chapter 2

ANNP Architecture

This chapter describes the parameters selected for ANNP. The learning of the Neural Networks depend on the Learning Algorithm, Training Rate, Initial Weights and Number of Hidden Neurons.

2.1 Input Selection

The training set consists of all the input and output data used by the Neural Network's learning algorithm. Depending on the problem one can select between discrete and continuous input. The range of values over which each input to a unit may vary is the same as the range of values over which the output of the activation function vary.

1. **Discrete Input:** The input here is binary. It is better to select input as -1 and 1 rather than 0 and 1. Because for the 0 input no training takes place. The discrete mean it uses the value only -1 and 1 or 0 and 1. The change in the weight ΔW_{ij} is given by,

$$\Delta W_{ij} = \eta \delta_i I_j$$

So, for the 0 input, there is no change in the weights and hence no training takes place.

2. **Continuous Input:** Mostly it is used for the Signal Processing Application. Some numerical problems could be avoided by having inputs and outputs scaled to some range as 0→ 1 or -1 → 1. The input range of -1→ 1 is used here to scale the illumination values of pixels from 0→ 255. The continuous mean it uses all the values between -1 and 1.

2.2 Activation Function

The activation function introduces a non-linearity in the Neural Network. The back-propagation algorithm requires only that activation function which is differentiable everywhere. All physical systems have a limited *dynamic range*, i.e. the response of the system cannot exceed a certain maximum response. This is mapped in all the activation function below.

1. **Sigmoid:** This activation function is usually used for the unipolar input for back-propagation. It is called *sigmoid.* The sigmoid has the additional advantage of providing a form of automatic gain control. For small signals the slope of the curve is steep producing high gain. As the magnitude of the signal becomes greater, the gain decreases. The activation function for the sigmoid is defined as,

$$y = \frac{1}{1+e^{-x}}$$

2. **Hyperbolic Tangent:** This activation function is used for the bipolar input for back-propagation. As input used here is -1→ 1, this is a right choice for the activation function. The derivative of this function, which

is used in back-propagation is also easier to implement. The activation function for the Hyperbolic Tangent is defined as,

$$y = \tanh(x)$$

2.3 Initial weights

Initial weights are selected using random number generator in the range of -1→1. For perceptron and functional-link(FL) there is no hidden layer so only one set of initial weights are selected. For back-propagation two sets of initial weights are selected, one from the input to hidden and other from hidden to output.

2.3.1 Perceptron/FL

The learning algorithm used for the both of these is Delta Rule. This rule always tend towards the global minimum of the objective function. So, the initial random weights make the convergence faster or slower but ultimately it finds the global solution. The LMS procedure finds the values of all of the weights that minimize the error function using a method called *gradient descent*[1].

2.3.2 Back-Propagation

The ANNP using back-propagation is sensitive to the random weights. The Generalized Delta Rule(*Back-Propagation*) tries to find the global minimum but it can stuck into local minimum. It is possible for the system to stuck into another minimum or to reach global minimum if it is started from different initial weights. Initial random weights also make convergence faster or slower.

2.4 Training Rate

The training process requires only that the change in weight is proportional to $\frac{\partial E}{\partial W}$, where E is prediction residual energy. The constant of proportionality is the **training rate**. The training rate makes the learning of the network faster or slower. If it is set too low, the network may take many trials to learn. If it is too large, the learning process is fast but it can lead to oscillation[17].

2.4.1 Delta Rule

It is recommended that training rate should be in the range of 0.01→0.25 and must not greater than 0.75. Training rate of 0.2 is selected for ANNP.

2.4.2 Generalized Delta Rule

The training rate of 0.1 is selected for ANNLP using backpropagation. As we increase the training rate the chances of oscillation also increases. Using *momentum* term we can increase the η without leading the ANNP into oscillation. The following change is made in weight change to include *momentum.*

$$\Delta W_{ij}(n+1) = \eta \delta_i I_j + \alpha \Delta W_{ij}(n)$$

$\Delta W_{ij}(n+1)$ = weight change for the *(n+1)* th sample

η = training rate

δ_i = error output for the i th output neuron

I_j = j th input

α = momentum coefficient

$\Delta W_{ij}(n)$ = weight change for the n th sample

The *momentum* filters out the high curvature and thus allows the effective weight steps to be bigger. The value of 0.05 is selected for η and 0.9 is selected for α. It has been found that network trains faster with η=0.05 and α=0.9(*with momentum*) than $\eta = 0.1$ and α=0.(*without momentum*)

2.5 Hidden Neurons

Hidden Neurons play an important role for the Neural Network. Some problems like XOR cannot be solved without hidden neurons. As the number of hidden neurons increases the weight vector associated with those neuron to the preceeding layer also increases and hence learning time increases. It is better to start with few hidden neurons and increase them till we get global minimum[1]. As we increase the number of hidden neuron the cost function becomes more smoother and chances to get global minimum also increases.

The back-propagation algorithm usually find the global minimum but sometimes it can stuck into local minima. Here, are the few suggestions to improve performance and reduce the occurrence of local minima:

1. Extra hidden neurons make the cost-function more smoother and network may come out of local minima.

2. Reducing the training rate also make the convergence of the network slower and make the cost-function smoother.

Chapter 3

Learning Algorithm and Prediction Techniques

This chapter describes the learning algorithm used with ANNP. It also describes the 1-D prediction algorithm based on the "autocorrelation" method. The main aim in prediction is to find out the *predictive coefficients*. The weights associated with the input-output layer, in delta rule, are like the prediction coefficients in a predictor without using Neural Network.

3.1 Delta Rule

This is a supervised learning algorithm. The main aim behind this rule is to adjust the strengths of the connections so that they will tend to reduce the difference between *target* and *output*. The *error* is the difference between those two values. This rule is also called as Widrow-Hoff learning rule or the Least Mean Square(LMS) rule.

3.1.1 Training Method for Delta Rule

The random weights are selected in the range of the -1→1. The rule for changing the weights are given by

$$\delta_n = (T_n - O_n) \tag{3.1}$$

$$\Delta_i = \eta \delta_n I_i \tag{3.2}$$

$$W_i(n+1) = W_i(n) + \Delta_i \tag{3.3}$$

where,

T_n = target output for the nth sample

O_n = actual output for the nth sample

δ_n = error output for the nth sample

η = training rate

Δ_i = the correction associated with the i th input I_i

$W_i(n+1)$ = the value of the weight i after adjustment

$W_i(n)$ = the value of weight i before adjustment

The training method for the delta rule can be summarized as follows:

1. Apply the input to the ANNP and calculate the actual output 'O'.
2. Compute the δ_n using equation (3.1).
3. Compute the Δ_i using equation (3.2).
4. Update the weights using equation (3.3).
5. Go to step 1. Repeat the above procedure till all the sets of the inputs are presented.
6. Go to step 1. Repeat the above procedure for the number of passes.

3.1.2 Delta Rule for Linear Prediction

The architecture of the ANNLP(Adaptive Neural Network Linear Predictor) is shown in Fig. 3.1. ANNLP has the number of neuron in the input layer as the order of the predictor. There is one neuron in the output layer.

The prediction residual energy for the nth sample is defined as

$$E_n = \frac{1}{2}(T_n - O_n)^2 \tag{3.4}$$

T_n = Target output for the nth sample

O_n = Actual output for the nth sample

When there are no hidden units it is straightforward to compute the relevant derivative. Here, we use derivative by parts: the derivative of the error with respect to the output of the unit times the derivative of the output with respect to the weight.

$$\frac{\partial E_n}{\partial W_i} = \frac{\partial E_n}{\partial O_n}\frac{\partial O_n}{\partial W_i} \tag{3.5}$$

From the equation (3.4),

$$\frac{\partial E_n}{\partial O_n} = -(T_n - O_n) = -\delta_n \tag{3.6}$$

This shows that the contribution of the output neuron to the error is simply proportional to δ_n. The actual output is defined as,

$$O_n = \sum W_i I_i \tag{3.7}$$

$$\frac{\partial O_n}{\partial W_i} = I_i \tag{3.8}$$

Thus, rewriting equation (3.5) using equation (3.6) and (3.8) we can find that,

$$-\frac{\partial E_n}{\partial W_i} = \delta_n I_i \tag{3.9}$$

$$\frac{\partial E}{\partial W_i} = \sum \frac{\partial E_n}{\partial W_i} \tag{3.10}$$

E is a total prediction residual energy. We can rewrite the equation (3.9) using the proportionality constant as,

$$\Delta W_i = \eta \delta I_i$$

This lead us to conclude that the net change in W_i after one complete cycle of pattern representations is proportional to this derivative and hence that the delta rule implements a gradient descent in E. Thus, with small enough learning rate, the delta rule will find a set of weights minimizing the error function.

3.1.3 Delta Rule for Non-linear Prediction

The architecture of the ANNNP(Adaptive Neural Network Non-linear Predictor) is shown in the Figure 3.2. This is also known as *Functional Link.* ANNNP has the number of neuron in the input as the all non-linear combination of the input pixel to the second order. To compute the prediction value at each pixel location, the value of the 'p' previous pixel samples plus the combination of those pixel are presented as input.

The actual output O_n is defined as:

$$O_n = \sum_i W_i I_{n-i} + \sum_i \sum_j W_{ij} I_{n-i} I_{n-j} + \sum_i \sum_j \sum_k W_{ijk} I_{n-i} I_{n-j} I_{n-k} + ... \quad (3.11)$$

where,

I_{n-i} is the first order input.

$I_{n-i}I_{n-j}$ is the second order nonlinear input.

O_n is the prediction value or *actual output* for the nth sample.

$W_i, W_{ij}, W_{ijk}, ...$ are the weights associated with ANNNP.

The prediction residual energy is written as,

$$E_n = \frac{1}{2}(T_n - O_n)^2 \quad (3.12)$$

Here, we have to show that the learning rule implements a gradient descent for the residual energy, E_n.

Using derivative by parts for the equation (3.12) we can write as,

$$\begin{aligned}
\frac{\partial E_n}{\partial W_i} &= \frac{\partial E_n}{\partial O_n}\frac{\partial O_n}{\partial W_i} \\
\frac{\partial E_n}{\partial W_{ij}} &= \frac{\partial E_n}{\partial O_n}\frac{\partial O_n}{\partial W_{ij}} \\
\frac{\partial E_n}{\partial W_{ijk}} &= \frac{\partial E_n}{\partial O_n}\frac{\partial O_n}{\partial W_{ijk}}
\end{aligned} \tag{3.13}$$

Using equation (3.6) and (3.8) we can rewrite the equation (3.13),

$$\begin{aligned}
-\frac{\partial E_n}{\partial W_i} &= \delta_n I_{n-i} \\
-\frac{\partial E_n}{\partial W_{ij}} &= \delta_n I_{n-i} I_{n-j} \\
-\frac{\partial E_n}{\partial W_i} &= \delta_n I_{n-i} I_{n-j} I_{n-k}
\end{aligned} \tag{3.14}$$

From (3.14) it is concluded that the if weights $W_i, W_{ij}, W_{ijk}, \ldots$ are changed in proportional to the corresponding derivative for the each input pattern then this process implements the gradient descent in E_n.

If we take total residual energy E = $\sum E_n$, then we can write as

$$\begin{aligned}
\frac{\partial E}{\partial W_i} &= \sum \frac{\partial E_n}{\partial W_i} \\
\frac{\partial E}{\partial W_{ij}} &= \sum \frac{\partial E_n}{\partial W_{ij}} \\
\frac{\partial E}{\partial W_{ijk}} &= \sum \frac{\partial E_n}{\partial W_{ijk}}
\end{aligned} \tag{3.15}$$

Above equations show that the total energy will be reduced if the weights are changed in proportional to the derivative. This show that the learning rule implement the gradient descent in E_n.

3.2 Generalized Delta Rule

This is also a supervised learning algorithm. . It is also known as the back-propagation algorithm. The architecture of the ANNP with Generalized Delta Rule is shown in the Figure 3.3.

There are i neurons in input layer, j in the hidden layer and k in the output layer. The weights associated with input-hidden layer is W_{ij} and that with hidden-output layer is W_{jk}.

3.2.1 Training Method for Generalized Delta Rule

The training method for the Generalized Delta Rule can be summarized as follows:

1. Compute the output at hidden layer before activation function:

$$HO_j = \sum_i W_{ij} I_i$$

2. Compute the output at hidden layer after activation function:

$$H_j = \tanh(HO_j)$$

3. Compute the output at output layer before the activation function:

$$OUT_k = \sum_j W_{jk} H_j$$

4. Compute the output at output layer after the activation function:

$$O_k = f(OUT_k)$$

5. Calculate the δ_k of the output layer:

$$\delta_k = f'(OUT_k)(T_k - O_k)$$

6. Compute the ΔW_{jk} and update the hidden-output weights W_{jk}:

$$\begin{aligned} \Delta W_{jk} &= \eta \delta_k H_j \\ W_{jk}(n+1) &= W_{jk}(n) + \Delta W_{jk} \end{aligned}$$

7. Compute the δ_j of the hidden layer:

$$\delta_j = \frac{1}{\cosh^2(HO_j)}(\sum_k \delta_k W_{jk})$$

8. Compute the Δw_{ij} and update the input-hidden weights:

$$\begin{aligned} \Delta W_{ij} &= \eta \delta_j I_i \\ W_{ij}(n+1) &= W_{ij}(n) + \Delta W_{ij} \end{aligned}$$

9. Go to step 1. Repeat the above procedure till all the sets of the inputs are presented.

10. Go to step 1. Repeat the above procedure for the number of passes.

3.2.2 Gradient Descent Implementation

If n input samples are presented during a training cycle, then

$$HO_{nj} = \sum_i W_{ji} I_{ni} \tag{3.16}$$

Thus, the hidden layer output after activation function is

$$H_{nj} = f_j(HO_{nj}) \tag{3.17}$$

For the generalized delta rule we must show that,

$$\Delta_n W_{ji} \ \alpha \ -\frac{\partial E_n}{\partial W_{ji}} \tag{3.18}$$

Here, E is the overall squared error function. It is useful to consider the derivative as resulting from the product of two parts. Thus, we can write as:

$$\frac{\partial E_n}{\partial W_{ji}} = \frac{\partial E_n}{\partial HO_{nj}} \frac{\partial HO_{nj}}{\partial W_{ji}} \tag{3.19}$$

Using Equation (3.16) we can rewrite the second partial derivative of the above equation as,

$$\frac{\partial HO_{nj}}{\partial W_{ji}} = \frac{\partial}{\partial W_{ji}} \sum_i W_{ji} I_{ni} = I_{ni} \tag{3.20}$$

Now, here we define,

$$\delta_{nj} = -\frac{\partial E_n}{\partial HO_{nj}} \tag{3.21}$$

So, we can rewrite the equation (3.19), using (3.20) and (3.21),

$$-\frac{\partial E_n}{\partial W_{ji}} = \delta_{nj} I_{ni} \tag{3.22}$$

This means, to implement the gradient descent in E, the weight change ΔW_{ji}, should be written as,

$$\Delta_n W_{ji} = \eta \, \delta_{nj} \, I_{ni} \tag{3.23}$$

To compute the δ_{nj} here, we use the method of derivative by parts for the equation(3.21),

$$\delta_{nj} = -\frac{\partial E_n}{\partial H_{nj}} \frac{\partial H_{nj}}{\partial HO_{nj}} \tag{3.24}$$

Using equation (3.17), we can rewrite the second part of equation (3.24) as

$$\frac{\partial H_{nj}}{\partial HO_{nj}} = f_j'(HO_{nj}) \tag{3.25}$$

which is simply the derivative of the activation function f_j for the jth unit. To compute the first factor in the equation (3.24), here, unit u_j represents the jth neurons in the output layer.

Considering u_j as output element we can rewrite the above equation(3.24) as,

$$\begin{aligned} E_n &= \tfrac{1}{2}(T_{nj} - H_{nj})^2 \\ \frac{\partial E_n}{\partial H_{nj}} &= -(T_{nj} - H_{nj}) \end{aligned} \tag{3.26}$$

Substituting equation (3.25) and (3.26) in equation (3.24),

$$\delta_{nj} = (T_{nj} - H_{nj}) f_j'(HO_{nj}) \tag{3.27}$$

Above equation is true for any output layer. Now, considering u_j as hidden element we can rewrite as,

$$\sum_k \frac{\partial E_n}{\partial OUT_{nk}} \frac{\partial OUT_{nk}}{\partial H_{nj}} = \sum_k \frac{\partial E_n}{\partial OUT_{nk}} \frac{\partial}{\partial H_{nj}} \sum W_{kj} H_{nj} = - \sum_k \delta_{nk} W_{kj} \tag{3.28}$$

Substituting equation (3.28) and (3.25) in equation (3.24),

$$\delta_{nj} = f_j'(HO_{nj}) \sum_k \delta_{nk} W_{kj} \tag{3.29}$$

From the above derivation it is conclude that the hidden layer δ_{nj} is given by, equation (3.29) and the output layer δ_{nk} is written as,

$$\delta_{nk} = (T_{nk} - O_{nk}) f_k'(OUT_{nk}) \tag{3.30}$$

3.3 1-D Prediction Technique

Let's say the order of the predictor is p and the input sample at time is n then, the predicted value $\hat{y}(n)$ is the linear combination of past values[7,8]. If we say *a(i)* as predictive coefficient then,

$$\hat{y}(n) = - \sum_{i=1}^{p} a(i)\, y(n-i) \tag{3.31}$$

The difference between actual value *y(n)* and predicted value $\hat{y}(n)$ is the difference signal,

$$\begin{aligned} e(n) &= y(n) - \hat{y}(n) \\ &= \sum_{i=0}^{p} a(i) y(n-i) \end{aligned} \tag{3.32}$$

Here, a(0) = 1. The main aim to use this algorithm is to find predictor coefficients *a(i)*. We have to compute the value of *a(i)* such that which will minimize mean-squared error. Here, we use *orthogonality principle* to find the coefficient which

make the error orthogonal to the samples *y(n-1), y(n-2), ... , y(n-p)*. Means, we require that,

$$< y(n-j)e(n) >= 0 \qquad j = 1, 2, ...p \tag{3.33}$$

We can rewrite the equation (3.32) using the above equation,

$$\sum_{i=0}^{p} a(i) \sum_{n} y(n-i)y(n-j) = 0 \quad j = 1, 2, ...p \tag{3.34}$$

The orthogonality principle says that resulting MSE is given by,

$$E = < e^2 > = < y(n)e(n) > \tag{3.35}$$

We can rewrite the above equation using (3.35) and (3.32),

$$E = \sum_{i=0}^{p} a(i) \sum_{n} y(n-i)\, y(n) \tag{3.36}$$

We can minimize the error over all time. We can rewrite the equation (3.34) as,

$$\sum_{i=0}^{p} a(i)\, r_{i-j} = 0 \quad where \quad j = 1, 2, ..., p \tag{3.37}$$

$$E = \sum_{i=0}^{p} a(i) r_i \tag{3.38}$$

where,

$$r_i = \sum_{n=-\infty}^{\infty} y(n)\, y(n-1) \tag{3.39}$$

The value of *y(n)* is defined for only finite number, N and it is zero elsewhere except (0, N - 1). So, we can write down the above equation as,

$$r_i = \sum_{n=i}^{N-1} y(n)\, y(n-i) \tag{3.40}$$

The equation (3.37), (3.38) and (3.39) together are known as *autocorrelation* method. The solution of these equation are given in Appendix A.

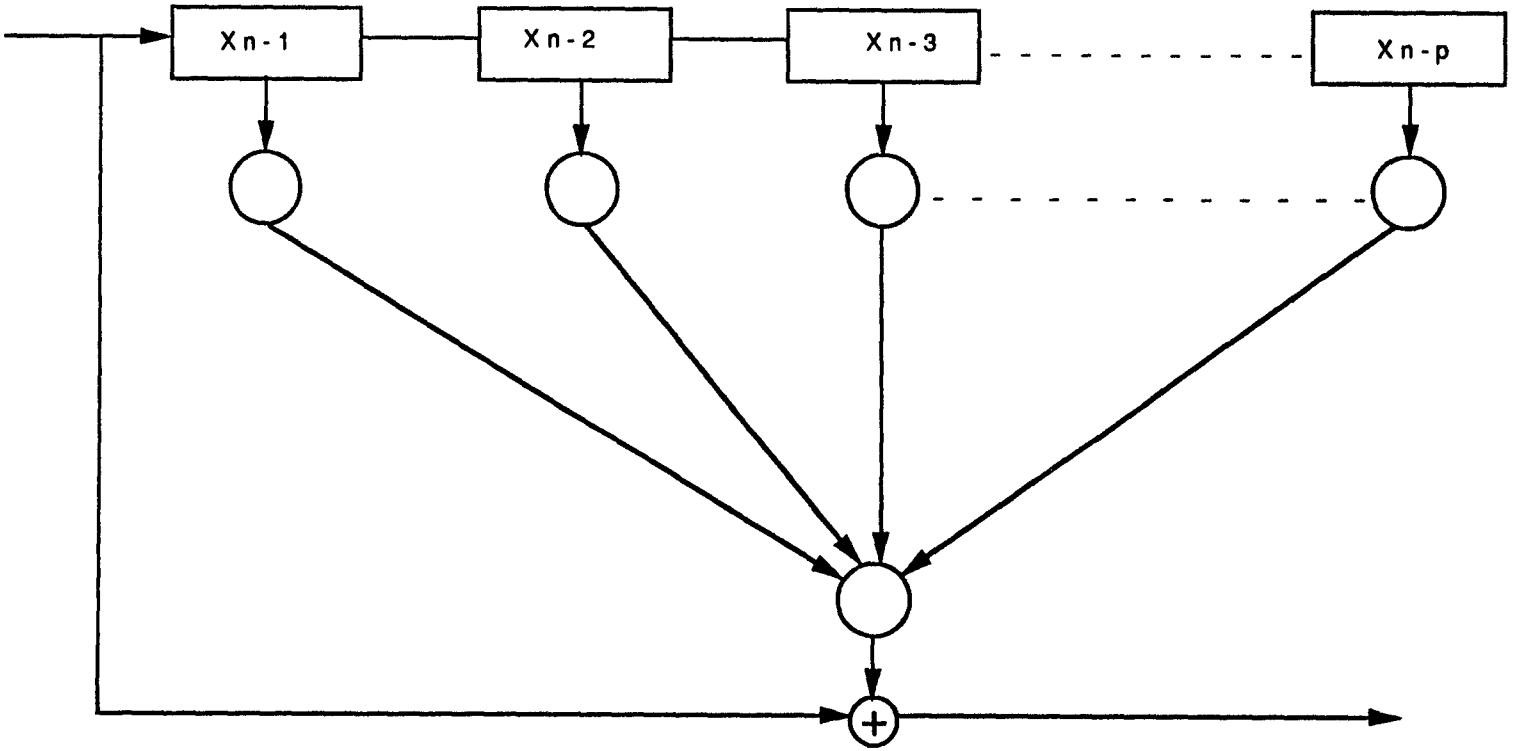

Figure 3.1: Adaptive Neural Network Linear Predictor (Perceptron)

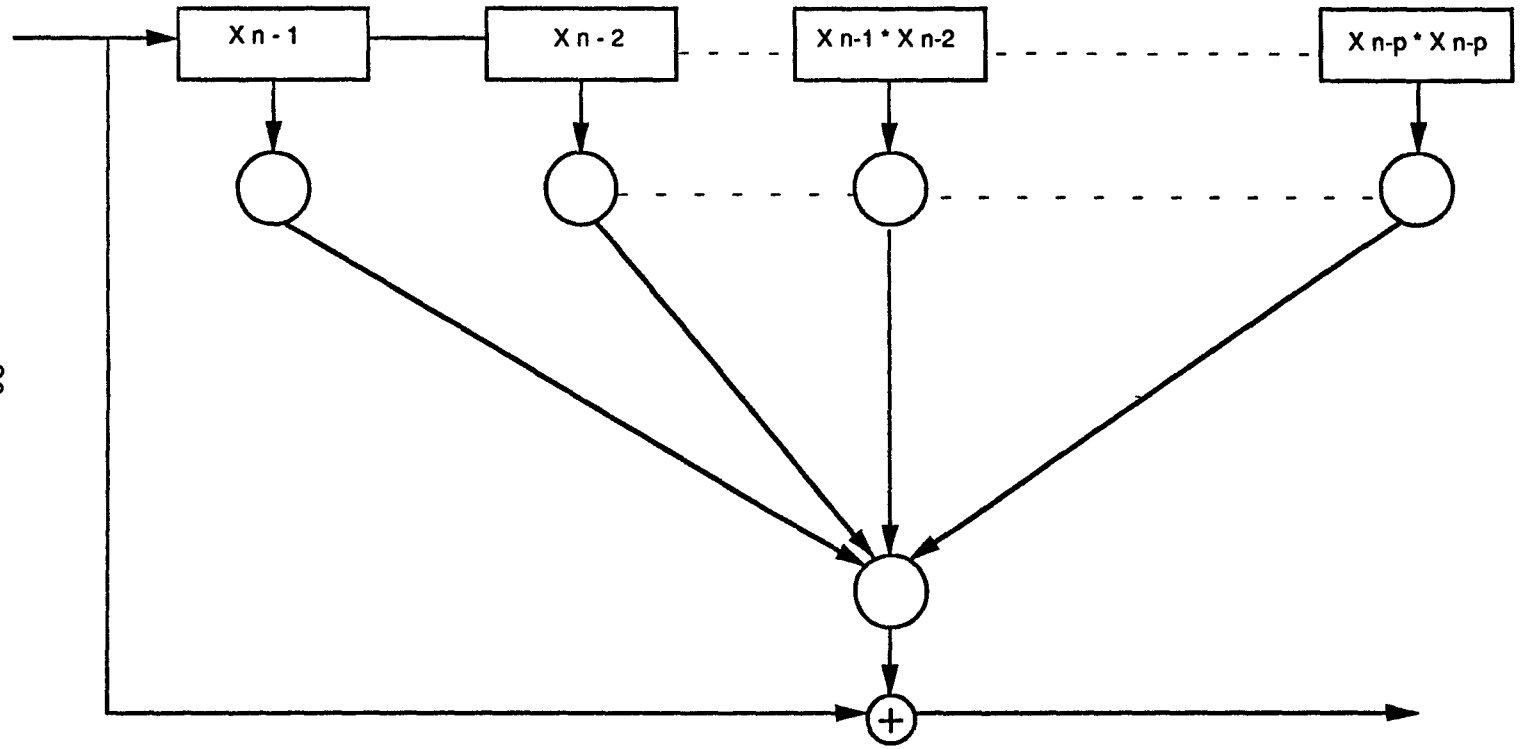

Figure 3.2: Adaptive Neural Network Non-linear Predictor (Functional Link)

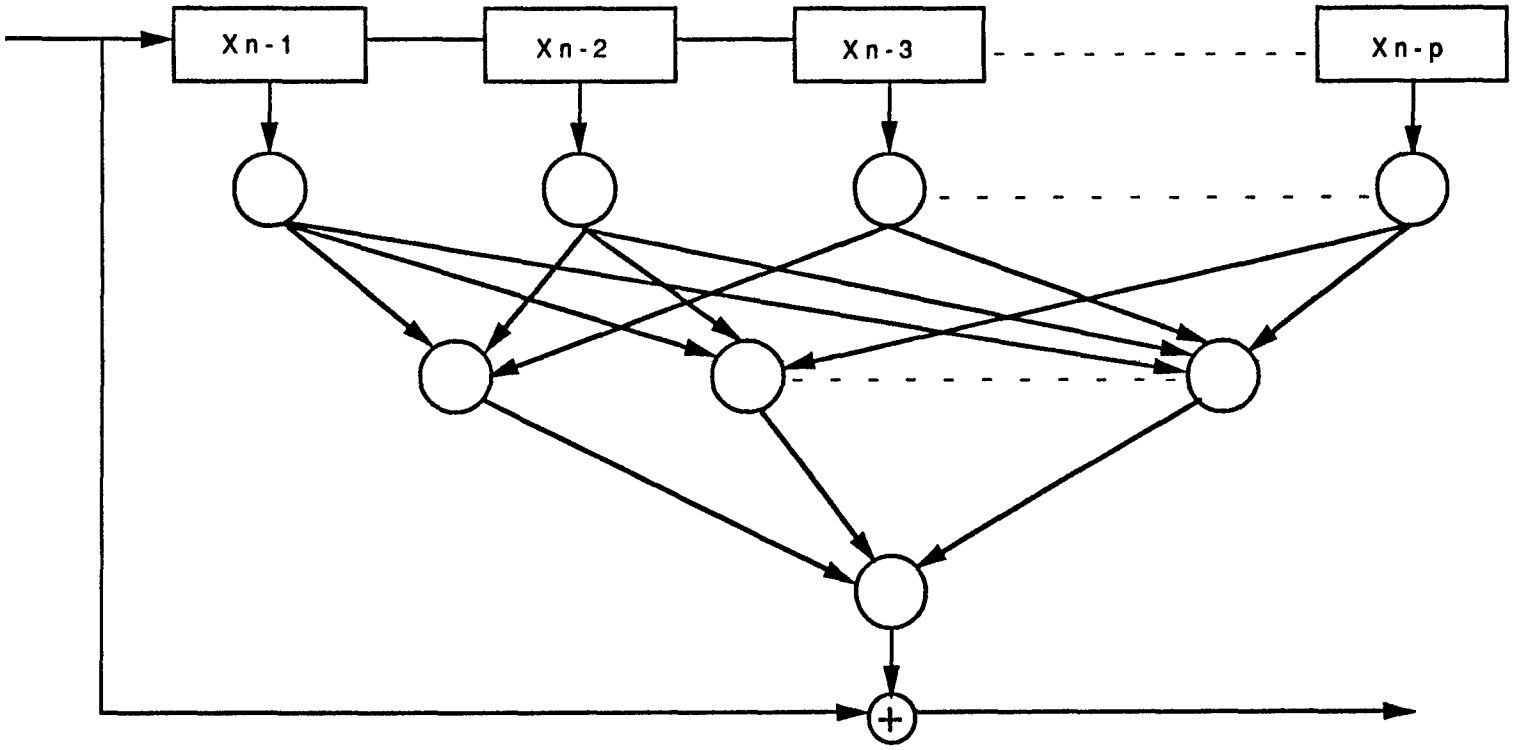

Figure 3.3: Adaptive Neural Network Linear Predictor (Backpropagation)

Chapter 4

Simulation of ANNP

Simulator for ANNP is designed using 'C' language. Simulation results give the comparative study of the different architecture for the ANNP. The image-file use here is LENA.

4.1 Selecting Memory Set

The choice of the memory set is very important for the design of predictor. For 1-D prediction the previous points(*pixels*) are selected as the order of the predictor. The points adjacent to the point to be predicted are the most useful and should be consider in the memory set. For 2-D prediction the optimum combination of the memory set is found and it is used for the different architecture of ANNP.

The different memory set are selected for the each order of the predictor to find out the optimum combination. (Figures 4.1, 4.2, 4.3, 4.4 and 4.5) The best combination among the each order is Fig.4.1(b) for the first order, Fig.4.2(c) for the second order, Fig.4.3(c) for the third order, Fig.4.4(e) for the fourth order and Fig.4.5(e) for the fifth order predictor. The optimum combination of pixel for each order includes the optimum combination of pixel for the previous order. For the predictor order the "best predictor" is defined as one that requires the minimum number of bits for the transmission of its error pattern.

The predictive coefficients are listed in the Appendix B.

4.2 Simulation Model

The image-file LENA which is 512 x 512 is divided into 256 blocks, each of 32 x 32. The aim here is to design a predictor for each of these blocks.

4.2.1 Model for Perceptron

The predictor is designed using delta rule has a number of input neurons as the order of the predictor. There are no hidden neurons and one output neuron. The memory set is presented to the input neuron as the input and the predicted point is presented to the output neuron as the target.

A set of random weights are selected for the first block. First point is selected as target and weights are updated using the delta rule. The next point is selected as the target and above procedure is repeated till all the points in that block are presented as target. The same procedure is repeated for 30 times for that block and it is assumed that the predictor for that block has required coefficients.

4.2.2 Model for Functional Link

The predictor is designed using delta rule has a number of input neurons as all non-linear combination of the input pixel to the second order. There are no hidden neurons and one output neuron. A combination of the memory set to the second order is presented to the input neurons as the input and the predicted point is presented to the output neuron as the target. It is called ANNNP *(Adaptive Neural Network Nonlinear Predictor).*

A set of random weights are selected for the first block. As the number of weights increases the training time also increases. First point is selected as

target and weights are updated using the delta rule. The next point is selected as the target and above procedure is repeated till all the points in that block are presented as target. The same procedure is repeated for 50 times for that block and it is assumed that the predictor for that block has required coefficients.

4.2.3 Model for Backpropagation

The predictor is designed using generalized delta rule has a number of input neurons as the order of the predictor. There are hidden neurons and one output neuron. The hidden neurons used here are such that which gives the optimum gain for that order. It is also called ANNLP(B) *(Adaptive Neural Network Linear Predictor using Backpropagation).*

Two sets of random weights are selected for the first block. First point is selected as target and weights are updated using the generalized delta rule. The next point is selected as the target and above procedure is repeated till all the points in that block are presented as target. The same procedure is repeated for 50 times for that block and it is assumed that the predictor for that block has required coefficients.

Following procedure is repeated for ANNLP(P), FL and ANNLP(B):
Now, with the fixed weights, inputs are given and the actual output is carried out which is compared with the desired output. It is repeated till each point of the block is presented as target.

Instead, of selecting random weights for the second block the final weights of the first block is selected which makes the convergence of the network faster. Thus, the initial weights of the each block except the first one is selected from the previous block.

4.3 Simulation Parameters

The measure of the predictor is defined by the prediction gain. If we consider $y(n)$ as desired output and $\hat{y}(n)$ as actual output then, difference signal $d(n)$ is,

$$d(n) = y(n) - \hat{y}(n) \tag{4.1}$$

The predictor gain G_p is defined as,

$$G_p(db) = 10\log_{10}(\frac{\sigma_i^2}{\sigma_d^2}) \tag{4.2}$$

Here,

σ_i^2 =Variance of the input signal

σ_d^2 = Variance of the different signal

Here, for each 32 x 32 block, prediction gain is carried out, using the variance of the input and difference signal for that block. The average gain is the sum of the gain of the individual block divided by 256. MSE(mean square error) is also known as the variance of the difference signal. MSE decreases as training passes are increased. (Fig. 4.9)

The prediction gain for each order of the predictor for the different architecture of Neural Network Predictor is shown in the table in the Appendix C. The effect of the training passes on MSE and G_p is also shown in Appendix C.

Huffman coding is a statistical data-compression technique whose employment reduce the average code length used to represent the symbols. Once the difference symbol values are found, probability of each symbol is found. Here, we want to transmit 511 symbols over the channel so using the probability we can find the Entropy, or the average bits/symbol.

The Entropy of 511 symbols is:

$$H(X) = -\sum_{i=0}^{510} P_i \log_2 P_i \tag{4.3}$$

The Huffman code is an optimum code of all statistical encoding techniques as it results in the shortest average code length. A key property of the Huffman code is that it can be instantaneously decoded as the coded bits in the compressed data stream are encoded[15].

4.4 Simulation Results

In 1-D prediction, the pixel which is nearest to the predicted one has a very high weight in the order of of 0.8 to 0.9. For 2-D prediction, pixel which has the most effect on the predicted one has a weight in the order of 0.5 to 0.9. As shown in the Table B.2 the second column represents the predictive coefficients of the pixel nearest to the predicted one for 1-D prediction. As shown in the Table B.2 and Table B.3 the weights of the most effective predictive coefficients for 2-D are decreased, compared to most effective predictive coefficients of the same order for 1-D. That means, in optimum 2-D predictor each pixels has considerable weightage compare to 1-D in which only the nearest one has very high weights.

As shown in the Table C.1, gain for the ANNLP(P) is about 3 db higher than that found using LPCA method for 1-D prediction (Figure 4.6). One of the reasons for that is, LPCA is "Direct Method"(Appendix A), and hence when we predict the first few pixels, as the order of the predictor, for next line we consider the input from the previous line. While in the case of ANNLP(P), as it is a "Iterative Method"(Appendix A), to predict the first few pixels, as the order of the predictor, for each line we consider extrapolated rows and columns.

Table C.2 represents prediction gain for ANNLP(P). The second column represents 1-D prediction gain while the third column represents the gain for the pixels vertically above the predicted one as the order of the predictor. This clearly shows that image LENA has higher vertical correlation than horizontal

correlation (Figure 4.7). The G_p is about 2 db higher for the same order.

The ANNLP(P) is used to find the optimum combination of the pixels for 2-D prediction. Table C.3 compares the 1-D with optimum 2-D for the same order. The G_p for the optimum 2-D is about 3.1 db higher than that of 1-D for the same order (Figure 4.8).

Table C.4 shows the value of G_p for the different pixel combination. The G_p correspond to Figure 4.1(a) is shown in second row and second column. The G_p corresponds to Figure 4.5(e) is shown in sixth row and sixth column. Comparing the gain of different pixels combination for the same order, the best combination of the pixels is found for the particular order.

Before the actual prediction takes place, neural network is trained for the number of times. It is assumed that it is trained after 30 passes for the 5th order ANNLP(P). As the number of weights increases, so as the training passes. Table C.5 shows the effect of training passes for the 3rd order ANNLP(P) on the G_p and MSE (Figure 4.9).

For the optimum pixel combination G_p for the ANNLP(P), FL and ANNLP(B) is found (Figure 4.10). There is not much difference in prediction gain which shows that predictor used with perceptron finds the global minimum. The results are tabulated in Table C.6.

Figure 4.1: First Order Pixel Combination

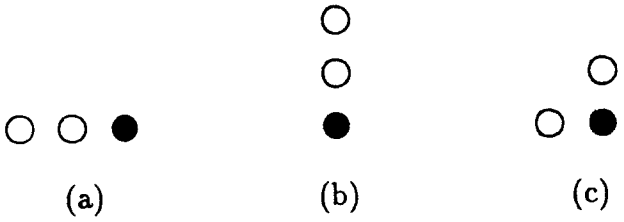

Figure 4.2: Second Order Pixel Combination

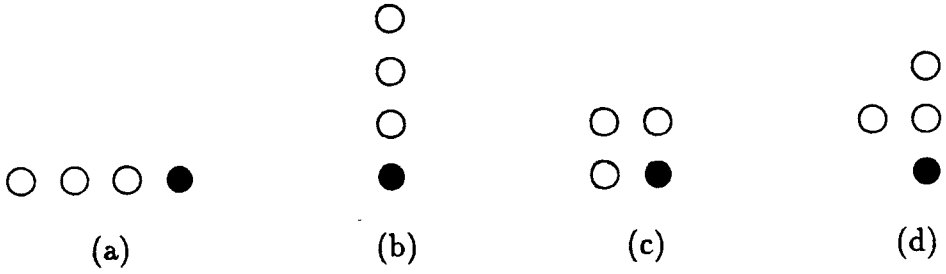

Figure 4.3: Third Order Pixel Combination

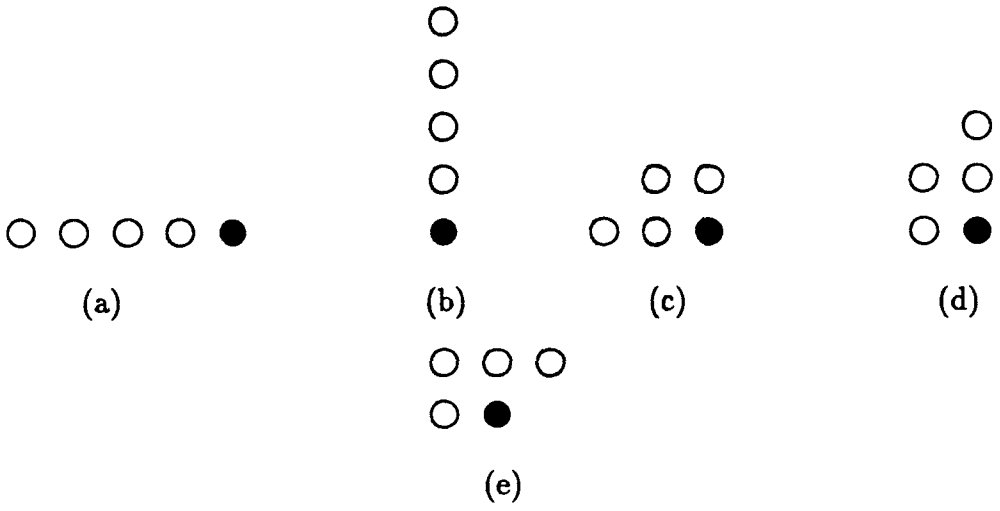

Figure 4.4: Fourth Order Pixel Combination

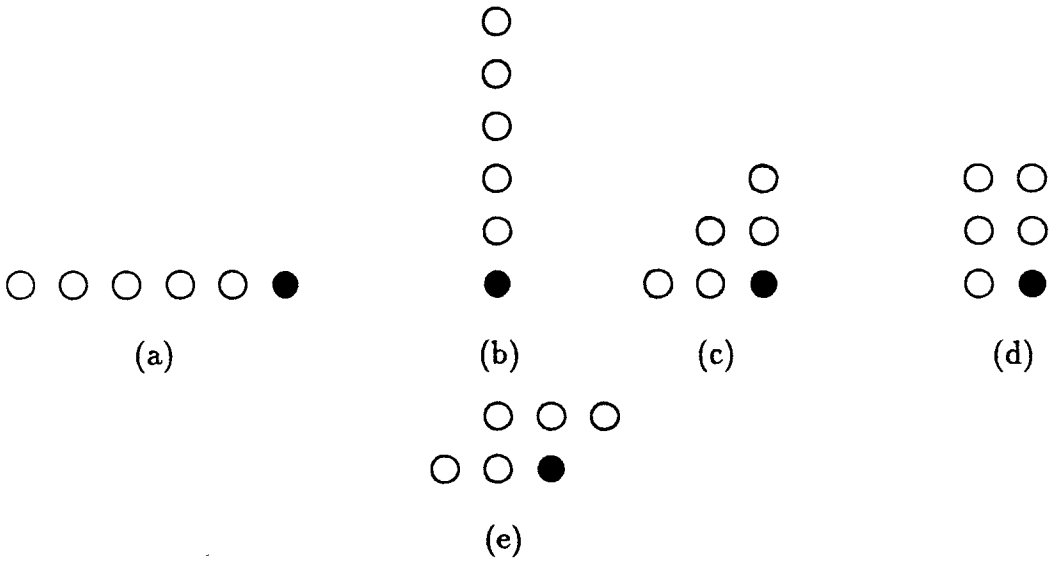

Figure 4.5: Fifth Order Pixel Combination

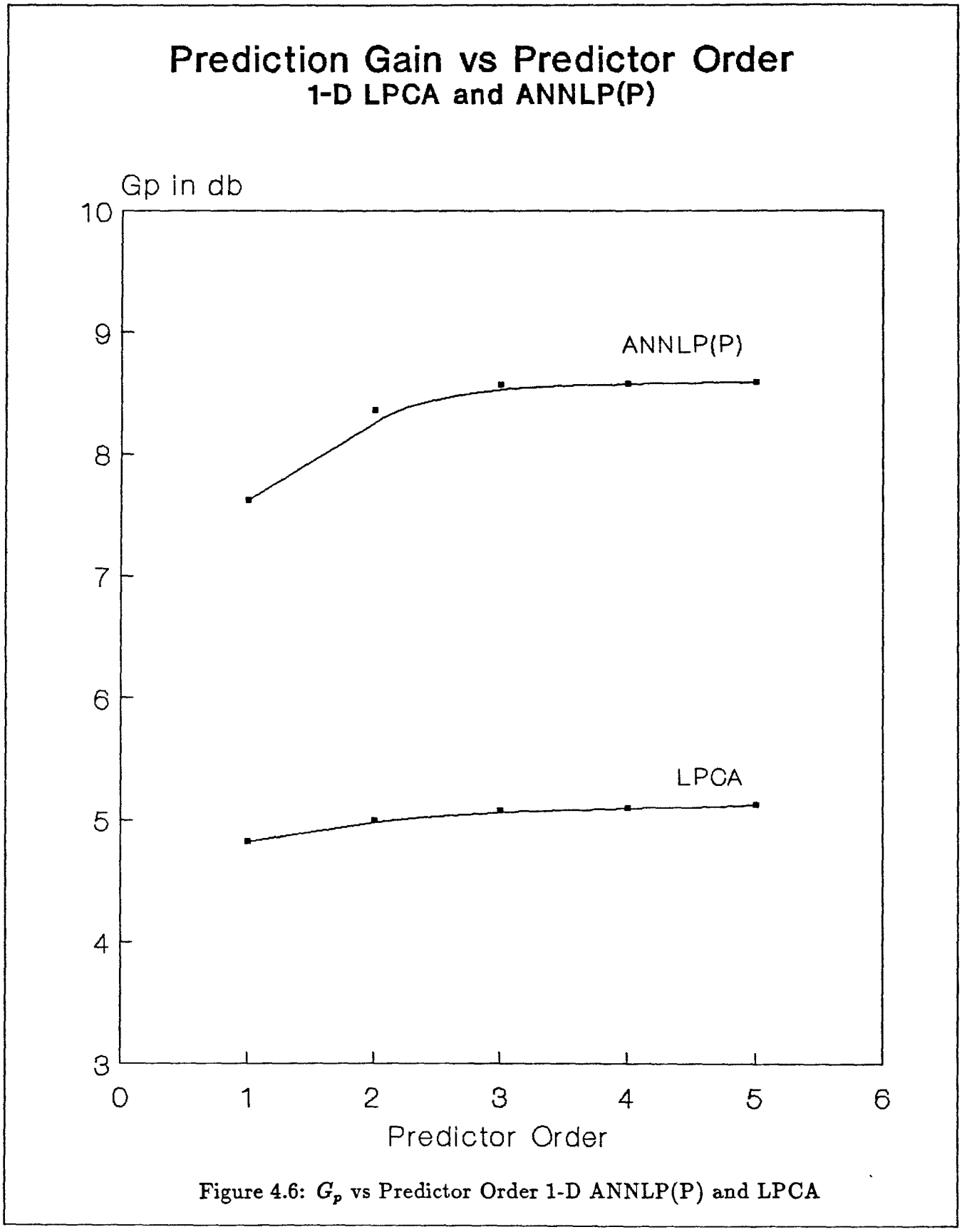

Figure 4.6: G_p vs Predictor Order 1-D ANNLP(P) and LPCA

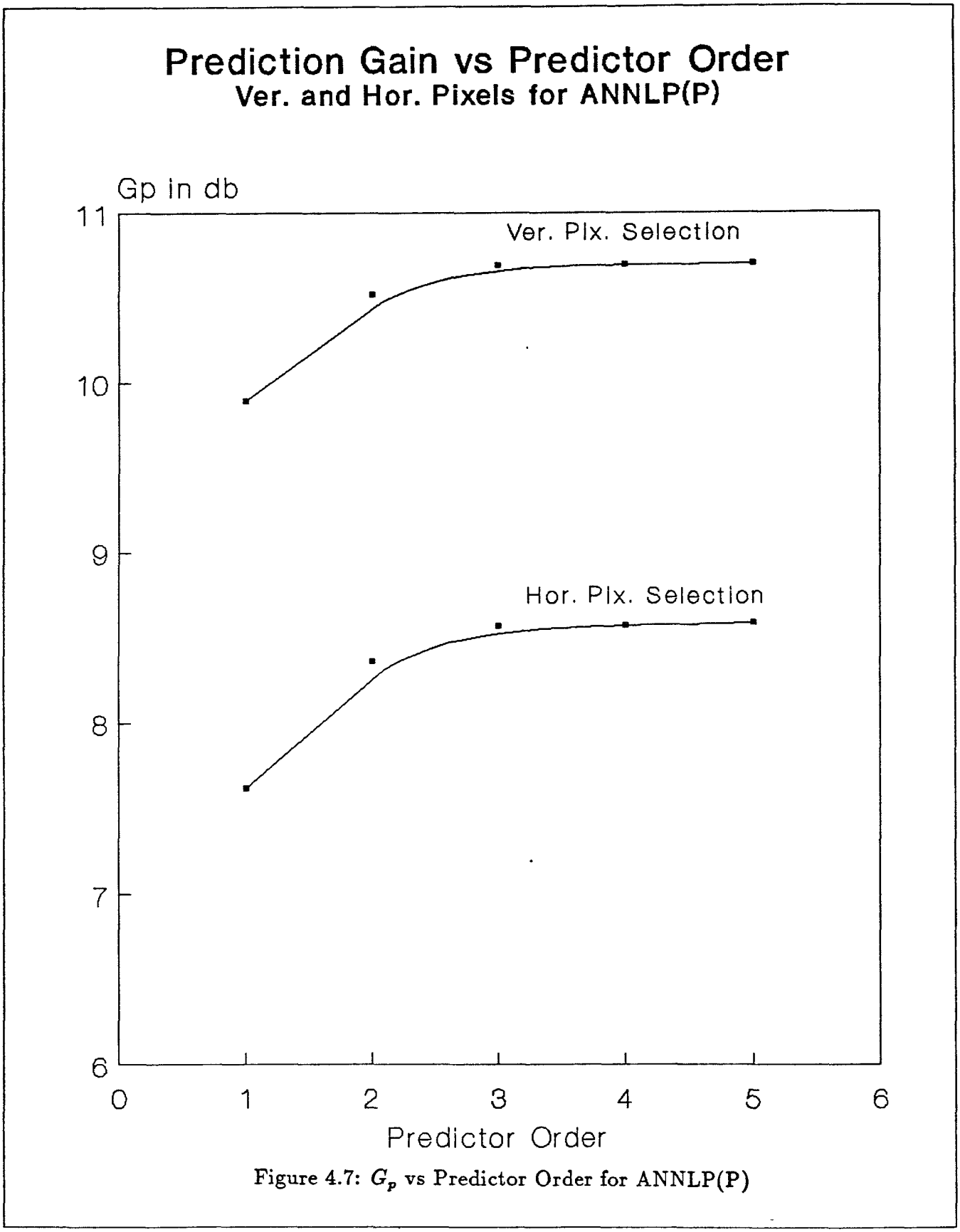

Figure 4.7: G_p vs Predictor Order for ANNLP(P)

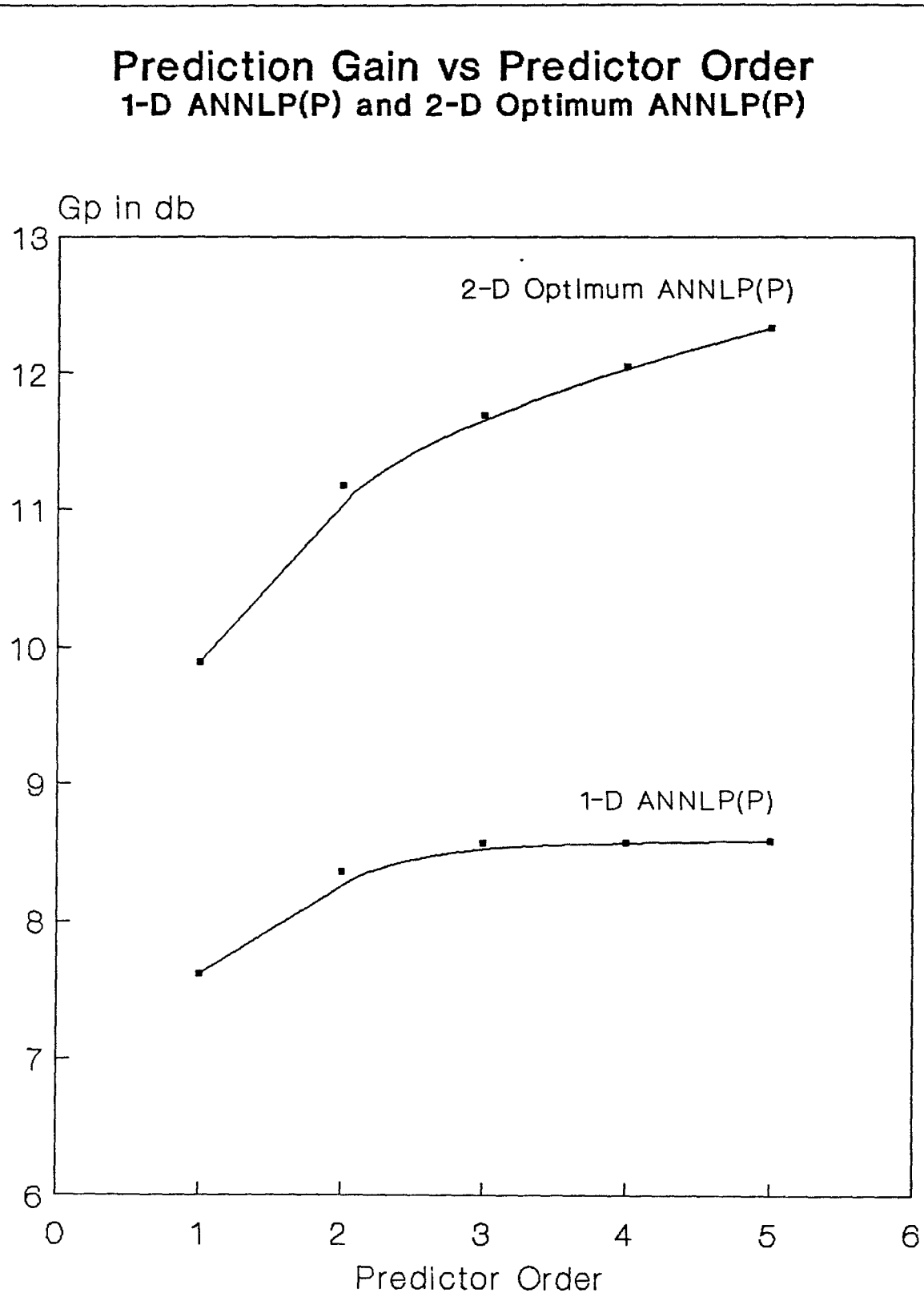

Figure 4.8: G_p vs Predictor Order for 1-D and 2-D Optimum for ANNLP(P)

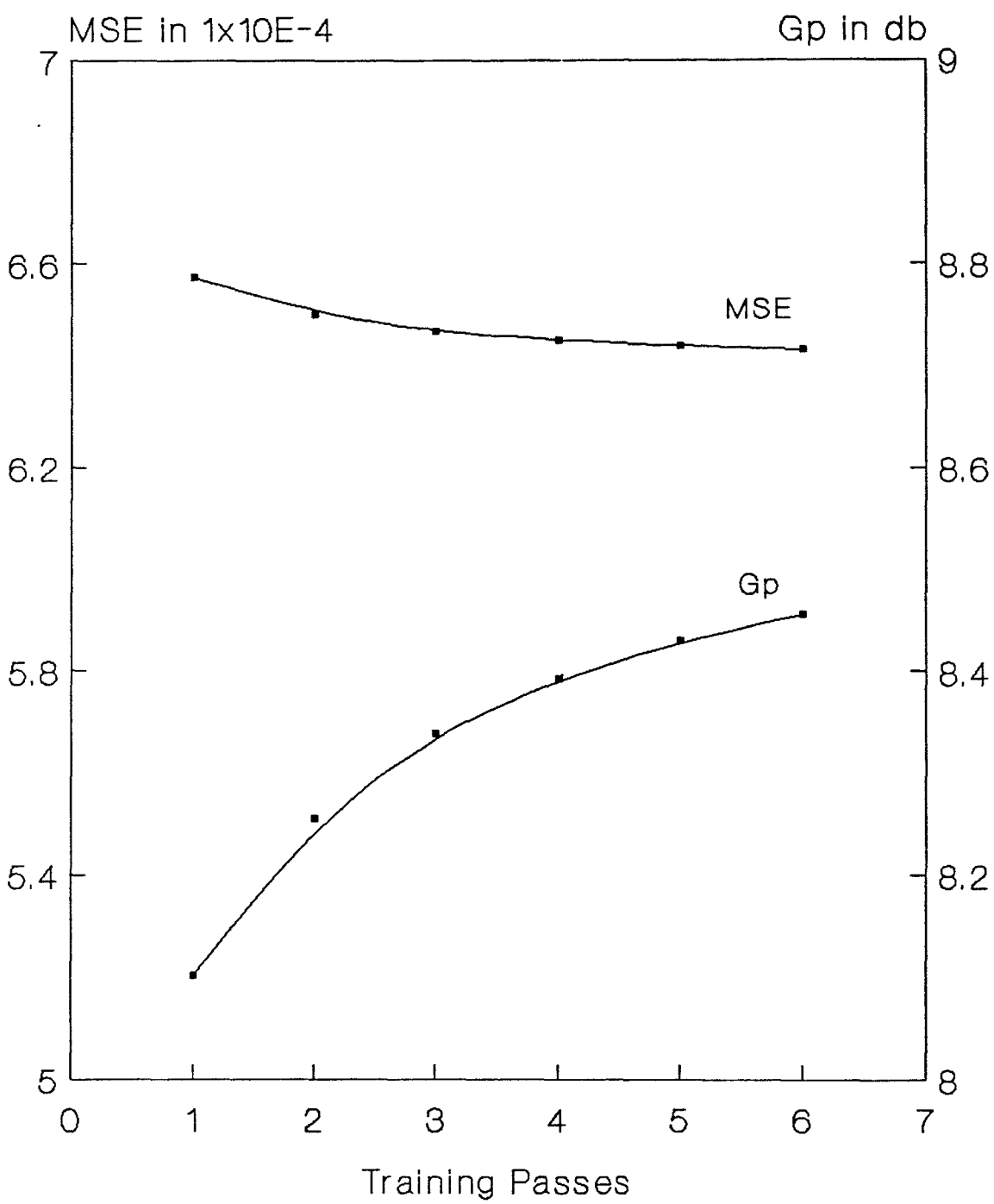

Figure 4.9: Effect of Training Passes on the MSE and G_p

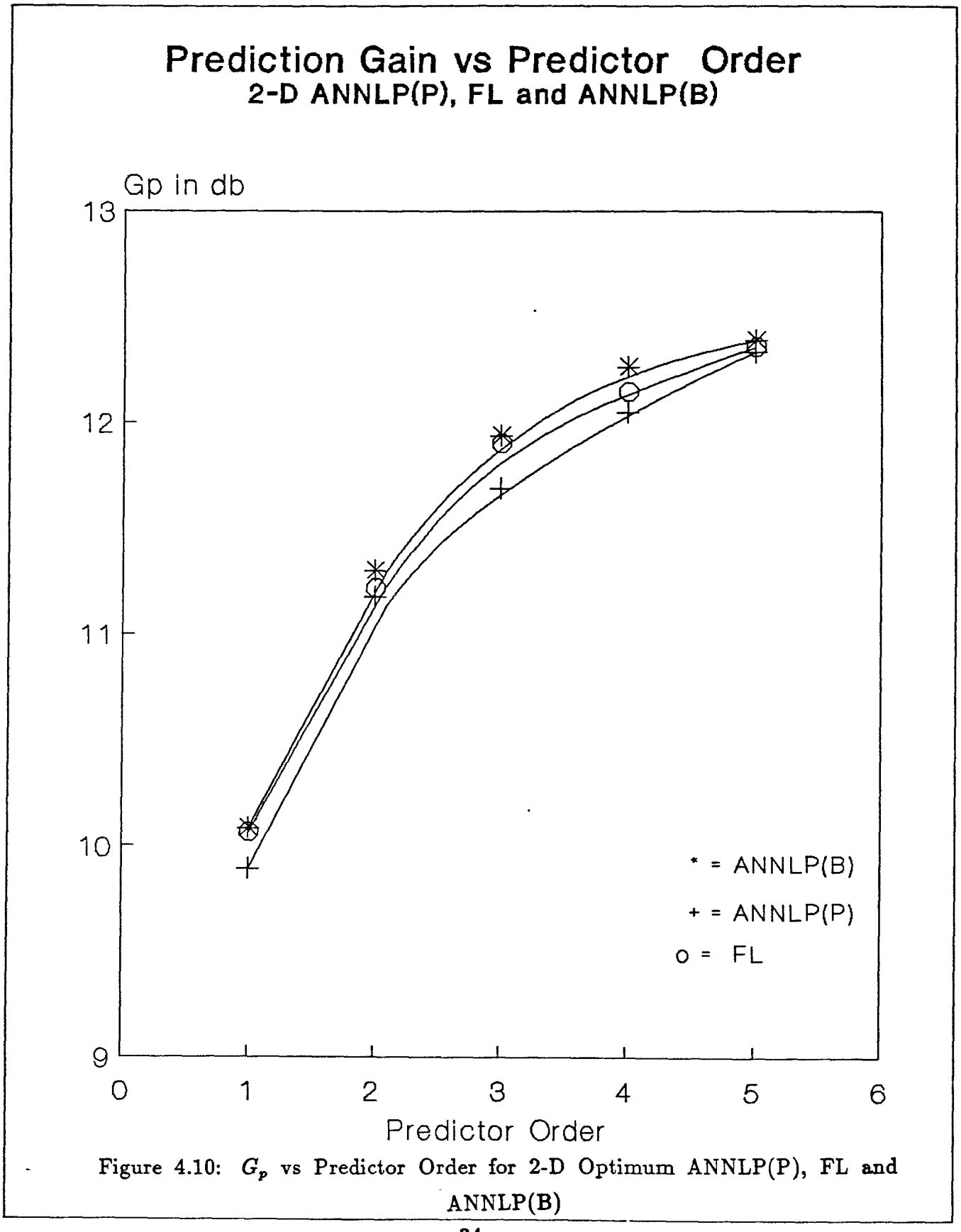

Figure 4.10: G_p vs Predictor Order for 2-D Optimum ANNLP(P), FL and ANNLP(B)

Chapter 5

Conclusion

This thesis describes the design of the predictor based on the Neural Network. The review of the literature indicate a limited activity in this field.

Different combinations of the pixels are used to find the best combination of pixels for the particular predictor order for the image LENA. The results show that the best combination of the pixels for the particular order includes the best combination of all previous order pixels combination.

The predictor using backpropagation algorithm (Generalized Delta Rule) gives marginally higher gain compare to the predictor using perceptron algorithm (Delta Rule). While the training time for the predictor using backpropagation is much higher compare to that with the perceptron, predictor using perceptron is the fair choice for image data compression, in the real world. The predictor using non-linear inputs upto second order also gives little bit higher gain but with the expense of more weights and more training time.

The obtained results clearly indicates the image LENA has higher vertical correlation than horizontal correlation. The 2-D prediction scheme gives much higher gain than 1-D scheme.

During last few years considerable research is going in the field of Neural Network. There are still many areas to be exploited like: 3-D prediction for the

moving images, better and faster prediction techniques and VLSI implementation of neural network for the prediction of images and speech.

Appendix A

Autocorrelation Method

The solution of the autocorrelation equation is given in this appendix. There are mainly two methods to compute the prediction parameters. The method used here is a *"autocorrelation"*. It is a part of *"Direct Method"*[8].

A.1 Iterative Method

In this methods, we start with the initial guess for the solution. There exist a number of iterative methods beside the direct method for the solution of simultaneous linear equation. The solution of the problem is updated by adding a correction term that is usually based on gradient descent. The solution of the problem depends much on the initial guess as for good guess it requires less iterations[8].

If the problem has many simultaneous equation then, it is easier to use this method. Some of the popular iterative methods are Gradient Descent, Newton's Method, Conjugate Gradient Method and Stochastic Approximation Method.

A.2 Direct Method

In this method, the predictive coefficient a(i), $1 \leq i \leq p$ is computed by solving a set of 'p' equations with 'p' unknown. Here, 'p' is the order of the predictor.

The equation (3.37) can be written as follow:

$$\begin{aligned} r_{-1}a(0) + r_0a(1) + r_1a(2) + \cdots + r_{p-1}a(p) &= 0 \\ r_{-2}a(0) + r_{-1}a(1) + r_0a(2) + \cdots + r_{p-2}a(p) &= 0 \\ r_{-3}a(0) + r_{-2}a(1) + r_1a(2) + \cdots + r_{p-3}a(p) &= 0 \\ \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots & \\ r_{-1}a(0) + r_0a(1) + r_1a(2) + \cdots + r_{p-1}a(p) &= 0 \end{aligned} \tag{A.1}$$

But, during derivation we assume that *a(0)=1*, so we can write down the above equation in matrix form. We also know that the, $r_{-i} = r_i$ for the real sequence,

$$\begin{bmatrix} r_1 & r_0 & r_1 & r_2 & \cdots & r_{p-1} \\ r_2 & r_1 & r_0 & r_1 & \cdots & r_{p-2} \\ r_3 & r_2 & r_1 & r_0 & \cdots & r_{p-3} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ r_p & r_{p-1} & r_{p-2} & r_{p-3} & \cdots & r_0 \end{bmatrix} \begin{bmatrix} 1 \\ a(1) \\ a(2) \\ \cdots \\ a(p) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cdots \\ 0 \end{bmatrix} \tag{A.2}$$

$$\mathbf{R} = \begin{bmatrix} r_1 & r_0 & r_1 & r_2 & \cdots & r_{p-1} \\ r_2 & r_1 & r_0 & r_1 & \cdots & r_{p-2} \\ r_3 & r_2 & r_1 & r_0 & \cdots & r_{p-3} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ r_p & r_{p-1} & r_{p-2} & r_{p-3} & \cdots & r_0 \end{bmatrix} \tag{A.3}$$

Here, we are trying to find the value of a(i) recursively, so multiplying $r_0, r_1, r_2, \cdots, r_{p-1}$ with '1' gives us a constant values. So, if we move the constant terms to the right-hand side, we will get p equations.

$$\begin{bmatrix} r_0 & r_1 & r_2 & r_3 & \cdots & r_{p-1} \\ r_1 & r_0 & r_1 & r_2 & \cdots & r_{p-2} \\ r_2 & r_1 & r_0 & r_1 & \cdots & r_{p-3} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{p-1} & r_{p-2} & r_{p-3} & r_{p-4} & \cdots & r_0 \end{bmatrix} \begin{bmatrix} a(1) \\ a(2) \\ a(3) \\ \cdots \\ a(p) \end{bmatrix} = - \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \cdots \\ r_p \end{bmatrix} \tag{A.4}$$

Here, $\mathbf{R}$ is Toeplitz and symmetric matrix. The recursive solution proceeds in steps. For each step, we have solution of $a(n-1)$ for the nth order predictor.

Levinson dived an elegant recursive procedure for solving this type of equation. This procedure was later updated by Robinson. Durbin's recursive procedure is to be consider twice as fast as Levinson's.

This recursive procedure can be specified as follows:

$$E_0 = r_0 \tag{A.5}$$

For step n, we can write as,

$$k_n = -\frac{1}{E_{n-1}} \sum_{i=0}^{n-1} a_{n-1}(i) r_{n-i} \tag{A.6}$$

$$a_n = k_n \tag{A.7}$$

For i=1 to n-1, we can write as,

$$a_n(i) = a_{n-1}(i) + k_n a_{n-1}(n-i) \tag{A.8}$$

$$E_n = E_{n-1}(1 - k_n^2) \tag{A.9}$$

The absolute value of k never exceed 1 and so each new prediction error tends to be less than the preceding one but greater than zero[7]. The above recursion procedure starts at n=0 and continue for the number of samples.

Appendix B

Predictive Coefficients

This appendix list the predictive coefficients associated with the 1-D LPCA, 1-D ANNLP(Perceptron) and 2-D optimum ANNLP(Perceptron).

The sum of the predictive coefficients for the particular order should be around 1[4]. Here, Table B.1 list the values of the coefficients for 1-D LPCA using Autocorrelation Method. The coefficients corresponds to the pixel sequence of Figure 4.1(a), 4.2(a), 4.3(a), 4.4(a) and 4.5(a) for the respective order.

ANNLP(Perceptron) has no hidden layer and hence the linkage weights from the input to the output directly represents the predictive coefficients. Table B.2 list the values of the predictive coefficients for 1-D ANNLP. Here, also coefficients corresponds to the pixel sequence of Figure 4.1(a), 4.2(a), 4.3(a), 4.4(a) and 4.5(a) for the respective order. Table B.3 list the values of the predictive coefficients for the Optimum 2-D ANNLP. The coefficients corresponds to the the pixel sequence of Figure 4.1(b), 4.2(c), 4.3(c), 4.4(e) and 4.5(e).

ORDER	*Predictive Coefficients*				
1	0.991827	-	-	-	-
2	1.045010	-0.053621	-	-	-
3	1.043368	-0.021622	-0.030621	-	-
4	1.041428	-0.022992	0.035491	0.00897	-
5	1.037027	-0.020527	0.33894	0.008970	-0.069456

Table B.1: **1-D LPCA**

ORDER	*Predictive Coefficients*				
1	0.967145	-	-	-	-
2	0.997648	-0.053532	-	-	-
3	0.975954	-0.163984	0.199464	-	-
4	0.921486	0.212866	0.032298	-0.220589	-
5	0.869829	0.226414	0.085533	-0.044182	0.180551

Table B.2: **1-D ANNLP(Perceptron)**

ORDER	*Predictive Coefficients*				
1	0.987083	-	-	-	-
2	0.681683	0.298295	-	-	-
3	0.749529	0.376215	-0.144427	-	-
4	0.569940	0.253739	-0.096784	0.248346	-
5	0.591839	0.280060	0.018771	0.243267	-0.149125

Table B.3: **2-D Optimum ANNLP(Perceptron)**

Appendix C

Prediction Gain

This appendix list the prediction gain for the different architecture of the Neural Network Predictor for the different combination pixels.

Table C.1 list the G_p for the LPCA *(without neural network)* and ANNLP *(with neural network).* The pixel combination used are as in the Figure 4.1(a), 4.2(a), 4.3(a), 4.4(a) and 4.5(a) for the corresponding order of the predictor.

Table C.2 list the G_p for the ANNLP. The second column corresponds to the the pixel combination of Figure 4.1(a), 4.2(a), 4.3(a), 4.4(a) and 4.5(a) for the particular order. The third column corresponds to the pixel combination of Figure 4.1(b), 4.2(b), 4.3(b), 4.4(b) and 4.5(b) for the particular order.

Table C.3 list the G_p for the ANNLP. The second column corresponds to the the pixel combination of Figure 4.1(a), 4.2(a), 4.3(a), 4.4(a) and 4.5(a) for the particular order. The third column corresponds to the pixel combination of Figure 4.1(b), 4.2(c), 4.3(c), 4.4(e) and 4.5(e) for the particular order.

Table C.4 list the G_p for the different combination of the pixels for the ANNLP(P). The G_p for each order corresponds to the Figure 4.

Table C.5 shows the effect of training passes on the G_p and MSE. This table corresponds to 1-D 3rd order ANNLP(P).

Table C.6 list the G_p for the optimum combination of the pixels correspond-

ing to Figure 4 for the particular order for the ANNLP(P), FL and ANNLP(B). For the FL, combination of the optimum pixels upto the second order is selected as the inputs.

ORDER	*LPCA*	*ANNLP*
1	4.824453	7.620573
2	4.996026	8.361572
3	5.074013	8.570030
4	5.099560	8.576096
5	5.123941	8.593728

Table C.1: G_p **in db 1-D LPCA and ANNLP(Perceptron)**

ORDER	*Horizontal Pixels*	*Vertical Pixels*
1	7.620573	9.892055
2	8.361572	10.518970
3	8.570030	10.690358
4	8.576096	10.691872
5	8.593728	10.69837

Table C.2: G_p **in db for ANNLP(Perceptron)**

ORDER	*1-D ANNLP*	*2-D Optimum ANNLP*
1	7.620573	9.892055
2	8.361572	11.1741
3	8.570030	11.6874
4	8.576096	12.046122
5	8.593728	12.331872

Table C.3: G_p **in db for ANNLP(Perceptron)**

ORDER	*Figure(a)*	*Figure(b)*	*Figure(c)*	*Figure(d)*	*Figure(e)*
1	7.620573	9.892055	-	-	-
2	8.361572	10.518970	11.1741	-	-
3	8.570030	10.690358	11.687	10.827391	-
4	8.576096	10.691872	11.770004	11.847797	12.046122
5	8.593728	10.69837	12.001704	11.87438	12.331872

Table C.4: G_p **in db for Different Pixel Combination**

TRAINING PASSES	*MSE*	G_p
1	0.006573	8.102315
2	0.006499	8.255650
3	0.006467	8.338809
4	0.006449	8.392299
5	0.006439	8.429673
6	0.006431	8.455760

Table C.5: **Effect of Training Passes on MSE and** G_p

ORDER	*ANNLP(P)*	*ANNNP(or FL)*	*ANNLP(B)*
1	9.892055	10.062193	10.085311
2	11.1741	11.21163	11.29942
3	11.6874	11.89523	11.93411
4	12.046122	12.13892	12.25913
5	12.331872	12.35294	12.38531

Table C.6: G_p **in db for ANNLP(P), FL and ANNLP(B)**

Bibliography

[1] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, "*Parallel Distributed Processing*," Cambridge, MA:MIT Press, 1986.

[2] P. D. Wasserman, "*Neural Computing: Theory and Practice*," Van Nostrand Reinhold Press, 1989.

[3] J. Li, C. Manikopoulos, "A Neural Network Approach to DPCM System Design". Submitted for Publication.

[4] Ali Habibi, "Comparision of Nth-Order DPCM Encoder With Linear Transformations and Block Quantization Techniques,"*IEEE Transactions on Communications*, VOL. COM-19, No.6 December 1971, pp 948-956.

[5] N. S. Jayant, Peter Noll,"*Digital Coding of Waveforms*," Prentice-Hall Press, 1984.

[6] H. Kobayashi, L.R. Bahl, "Image Data Compression by Predictive Coding: Prediction Algorithm", *IBM Journal of Research and Development*, March 1974, pp 164-171.

[7] Thomas Parsons, "*Voice and Speech Processing*," McGraw-Hill Press, 1987.

[8] J. Makhoul, "Linear Prediction: A Tutorial Review," *Proceeding of IEEE*, VOL-63, NO.4, April 1975, pp 561-580.

[9] E. Rietman, Robert C. Frye, "The Back Propagation Algorithm for Neural Networks," *AT&T Bell Laboratories.*

[10] A. N. Netravali, B.G. Haskell, "*Digital Pictures: Representation and Compression*", Plenum Press, 1988.

[11] R. Lippmann, "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, April 1987, pp 4-22.

[12] Paul J. Werbos, "Backpropagation Through Time: What It Does and How To Do It", *IEEE Proceedings*, October 1990

[13] Anil K. Jain, "Image Data Compression: A Review", *Proceedings of the IEEE* Vol 69. No.3 March 1981, pp 349-386.

[14] Akio Mukai, Tsuneo Saito, Yukio Hoshiko, "DPCM Picture Signal Coding with Adaptive Linear Prediction", *Electronics and Communications in Japan*, Vol. 62-B, No. 11, 1979.

[15] Gilbert Held, "*Data Compression: Techniques and Applications Hardware and Software Considerations*", John Wiley & Sons, 1984.