IMPLEMENTATION OF VARIABLE STEP-SIZE LMS FILTER IN NEURAL NETWORK

A dissertation submitted in the partial fulfilment of requirement

for the award of degree of

Master of Technology

In

VLSI DESIGN

Submitted by:

DEEPAK GUPTA

Roll No: 601261012

Under the guidance of:

Dr. Ravi Kumar

Assistant Professor, ECED

Thapar University, Patiala

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR UNIVERSITY, Patiala

(Established under the section 3 of UGC Act, 1956)

PATIALA – 147004 (PUNJAB)

DECLARATION

I hereby declare that the work which is being presented in this dissertation entitled, **"Implementation of variable step-size LMS Filter in neural network"** in partial fulfilment of the requirement for the award of degree of **Master of Technology** in **VLSI Design** submitted in Electronics and Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Ravi Kumar (Assistant Professor), ECED and refers other researcher's work which are duly listed in the reference section.

The matter presented in this dissertation has not been submitted in any other University / Institute for the award of any other degree.

Date: 10/07/2014

deepakgupta

Deepak Gupta

Roll. No. 601261012

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Date: 10/7/2014

Ravikumar

Dr. Ravi Kumar

Assistant Professor, ECED

Thapar University, Patiala

Countersigned by:

(Dr. Sanjay Sharma)

Professor and Head ECED

ECED, Thapar University, Patiala

(Dr. S. K. Mohapatra)

Dean of Academic Affairs

Thapar University, Patiala

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to **Dr. Ravi Kumar, Assistant Professor,** Electronics and Communication Engineering Department, Thapar University, Patiala for his patient guidance and support throughout this report. I am really very fortunate to have the opportunity to work with him. I found this guidance to be extremely valuable.

I am also thankful to **Head of the Department, Dr. Sanjay Sharma** as well as our P.G. co-ordinator **Dr. Kulbir Singh (Associate Professor)** of Electronics & Communication Engineering Department for their encouragement and inspiration for the execution of the dissertation work.

Further, I would like to thank entire faculty member, staff of Electronics and Communication Engineering Department for the help and moral support which went along the way for the successful completion of this work. I thank all those who have contributed directly or indirectly to this work.

Lastly, I would like to thank my parents and grandparents for their years of unyielding love and for constant support and encouragement. They have always wanted the best for me and I admire their determination and sacrifice.

Deepak Gupta

ABSTRACT

This dissertation is an effort towards the implementation of an integrated ANN trained with backpropagation algorithm that can also function as a variable step size LMS filter. An artificial neural network is an emulation of biological neural system. An artificial neural network is an adaptive system. Learning rule is required to make the neural network adaptive. The implementation of the neural network suffers from various bottlenecks including massive consumption of computational resources and difficulty to determining the parameters of the network and training algorithm.

This work discusses the effect of the step size on the training of neural network. A novel variable step size algorithm based on Principal Component Analysis (PCA) that derived from statistical analysis has been proposed.

Furthermore, a novel approach is proposed to implement backpropagation algorithm in FPGA for effective resource utilization. Simulation and implementation results confirm the efficacy of the proposed techniques both in terms of generalization performance and hardware resource utilization.

CONTENTS

\

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

ANN	Artificial Neural Network
LMS	Least Mean Square
BP	Backpropagation
PCA	Principal Component Analysis
ASIC	Application Specific Integrated Circuit
LM	Levenberg-Marquardt
SIMD	Single Instruction Multiple Data
MIMD	Multiple Instruction Multiple Data
FPGA	Field Programmable Gate Array
CPS	Connection per Second
CUPS	Connection Update Per Second
SP	Serial Processing
PPP	Partial Parallel Processing
FPP	Full Parallel Processing
FXP	Fixed Point
FLP	Floating Point
DA	Direct Approximation
LUT	Look up Table
PWL	Piece Wise Linear
MSE	Mean Square Error
MAC	Multiply and Accumulate
HDL	Hardware Description Language
XPE	Power Estimator
PAR	Place and Route
NCD	Native circuit Description
RAM	Read Only Memory
APEX	Adaptive Principal Component Extraction Algorithm
MLP	Multilayer Perceptron
VHDL	Very High Speed Integrated Circuit Hardware Description Language

CHAPTER 1

INTRODUCTION AND LITERATURE REVIEW

Artificial neural networks are data processing models that inspired by the principles of computations performed by the biological neural networks of the brain. Neural networks posses many attractive characteristics that may ultimately surmount some of the limitations in canonical computational systems. The processing in the brain is mainly parallel and distributed. The information store in connections, mostly in myelin layers of axons of neurons so distributed over the network. The store information is process by the large number of parallel neurons. Neural networks have ability to learn the rules that describing training data, previously learnt information and how to respond to novel patterns. Neural networks are fault-tolerant, in the sense that the loss of a few neurons or connections does not affect their behaviour so much because information processing involves a large number of neurons and connections.

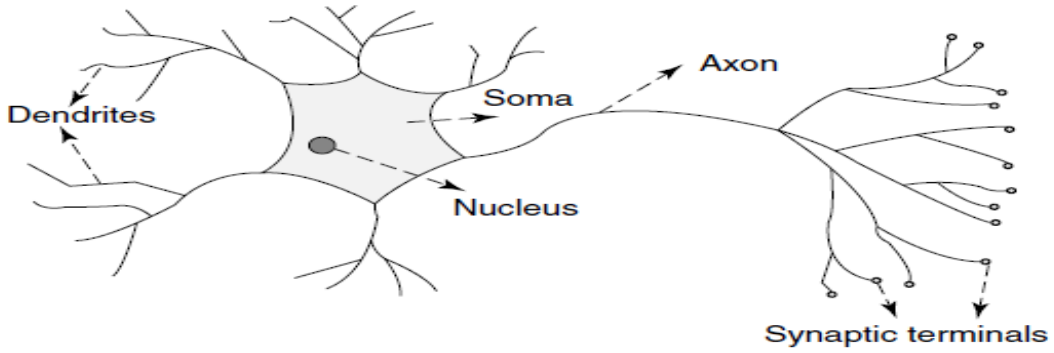


Figure 1.1 Biological neuron [1]

1.1 Properties and Capabilities of Artificial Neural Network

The neural has a powerful computational power. Computational power of the neural network based on two factors. The first massively parallel structure and second its ability to learn. These properties make the neural network to solve complex problems. The neural network offers the following useful properties and capabilities:

1. **Linearity and Nonlinearity**: Artificial neuron can be linear and nonlinear. The Neural network is interconnection of these neurons. Nonlinearity is distributed throughout the network. Nonlinearity is a very important property and it is responsible for the generation of the input signal.

2. **Input output Mapping:** Synaptic weights of the neural network adjust by training examples. Training example consist a unique input signal and a corresponding desired response. Synaptic weights of the network modified to minimize the difference between actual response and desired response in accordance with statistical criterion. The training of the network repeated until no further significant changes in the synaptic weights. Ex: Pattern classification tasks.
3. **Adaptability:** Neural networks have ability to adapt their synaptic weights according to changes in the surrounding environment. Neural network designed to change the weight in real-time. The Neural network is a very useful tool in adaptive pattern classification, signal processing and control application.
4. **Uniformity of Analysis and Design:** Neural networks have universality as information processor because same notation is used in all domains that use the application of the neural network. For example Neuron is the common ingredient to all neural networks that make possible to share learning algorithm and theories in different application of the neural network.
5. **Evidential response:** In pattern classification application a neural network has the capability to provide which particular pattern to select and a give confidence in the decision made.
6. **Contextual Information:** Information is presented by the every structure and activation state of a neural network. Every neuron in the network affects the activity of the all other neurons in the network.
7. **Fault Tolerance:** Neural network stored the information in the distributed form so performance degrades gracefully under adverse operating condition. Neural network implemented in hardware form has the potential to be inherently fault tolerant or capabilities of robust computation. Neural network exhibits a graceful degradation in performance rather than catastrophic failure.
8. **VLSI Implementability:** The neural network has massively parallel architecture. The massively parallel architecture makes computation fast for certain tasks. This feature makes the neural network well suited for implementation using VLSI technology**.**
9. **Neurobiological analogy:** The design of the neural network inspires by the human brain that tells us fault tolerant parallel processing is not only physically possible but also fast and powerful. Neurobiologists use the neural networks as a research tool for interpretation of neurobiological phenomena.

1.2 Applications of Neural Networks

Artificial neural networks have many applications in several domains. Some of them are given below:

1. **Speech:** speech recognition, vowel classification and speech compression.
2. **Telecommunications:** Image and data compression, real-time conversion of spoken language
3. **Electronic:** integrated circuit chip layout, control of the process, chip failure analysis.
4. **Robotics:** manipulator controllers, and vision systems.
5. **Defence:** Target tracking, facial recognition, new kinds of sensors, image signal processing, including data compression, feature extraction and noise suppression.
6. **Medical:** Breast cancer analysis, EEG and ECG analysis.

1.3 Least Mean Square Algorithm [32]

A learning system behaves like an operator on signals, sounds and images etc. The learning system has a free parameter and adaptive algorithm. An adaptive algorithm is used to automatically adjust these parameters in order to improve the efficiency of the learning system. These adjustable parameters are weights indicated in fig. 1.2.The Adaptive algorithm uses stochastic input signal to adjust the parameter of the learning system. An efficient algorithm minimizes the use of input data for weight adjustment and quality of the solution is also maximized. Minimizing data usage and solution quality is inversely proportional to each other. Fast adaptive convergence is related to minimizing data usage, but fast adaptive convergence may be providing poor quality of solution. This trade off is present in every learning system.

The Least mean square algorithm is the adaptive algorithm. Least mean square algorithm (LMS) or delta rule was discovered by Widrow and Hoff in 1960.LMS algorithm is widely used in linear adaptive filtering. In linear adaptive filtering neuron operates in linear mode. LMS algorithm is used in adaptive signal processing such as adaptive antennas, noise cancelling, inverse controls, seismic signal processing and adaptive equalization in high speed modems. The LMS and perceptron are closely related to each other so study of these two algorithms can be done simultaneously. The perceptron is used for classification of patterns. Perceptron classifies linearly

separable patterns. Perceptron is a simple form of the neural network, which basically consists of a signal neuron with adjustable synaptic weights and bias.

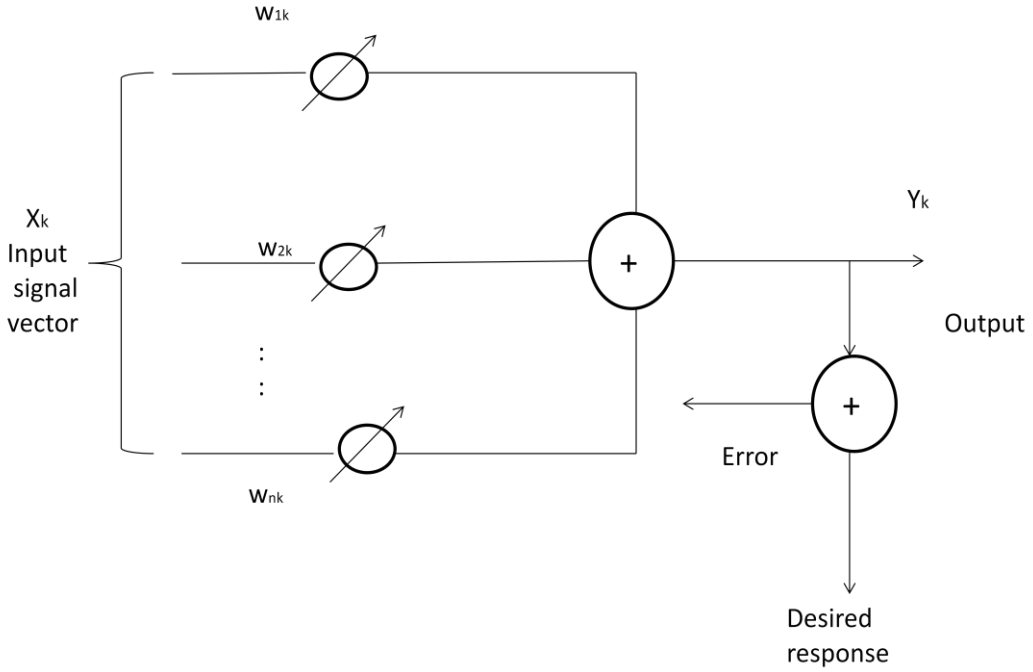


Figure 1.2 Adaptive linear combiner [14]

A Single linear neuron works as an adaptive filter when many inputs and one output are used in the model Weights are adjusted using the steepest descent method, Newton's Method and Gauss-Newton Method. These algorithms are relevant in the study of adaptive filters. These Methods are described in the following section.

1.3.1 Steepest Descent

In the steepest descent method synaptic weights are adjusted in direction opposite to the gradient vector. The gradient vector is denoted by $\nabla \boldsymbol{€}(\boldsymbol{w})$. **€ (w)** is the cost function and **∇€(w)** is the gradient of the cost function. Cost function must be reduced at the each iteration by using the following equation:

$$€\big(w(n+1)\big) < €(w(n) \tag{1.1}$$

Where w (n) is the old value of the weight vector and w (n+1) updated value of weight vector. The steepest descent algorithm is described by the following equation:

$$w(n+1) = w(n) - \eta \nabla € \big(\mathrm{w(n)}\big) \tag{1.2}$$

η is a positive step size or also called learning rate parameter. It is evident from equation 1.2 for positive learning rate parameter the cost function is decreased when

the algorithm undergoes present iteration to the next iteration. η has a great influence on the convergence behaviour and the effect of η on the convergence are given below:

a) When η is small, the transient response of the algorithm is over damped.
b) When η is large, the transient response of the algorithm is under damped.
c) When η is above some critical value, the algorithms become unstable.

1.3.2 Newton's Method

Newton's method minimizes the quadratic approximation of the cost function €($\boldsymbol{w}$) and minimization is performed at the every iteration of the algorithm. In the Newton's method weights are updated using equations that are given below:

$$w(n+1) = w(n) + \Delta w(n)$$

$$= w(n) - H^{-1}(n)g(n) \qquad (1.3)$$

Where $H^{-1}(n)$ is the inverse of the hessian of € (w). g (n) is the gradient vector of the cost function. H (n) is the hessian matrix of € (w) and hessian matrix is defined as $H = \nabla^2 €(w)$.

It is evident from the definition of the hessian matrix; cost function should be double differential with respect to elements of weight vector. Newton's method converges quickly, alike steepest descent, it does not permit the zigzagging behaviour vector. Hessian matrix should be positive definite matrix for all iteration otherwise the Newton method cannot work properly.

1.3.3 Gauss Newton Method

Gauss – Newton method is used for those cost functions that can be expressed as the sum of squares error. The mathematical forms of this cost function are given below:

$$€(w) = \frac{1}{2} \sum_{i=1}^{n} e^2\,(i) \qquad (1.4)$$

Where ½ is the scaling factor and used to simplify the analysis. All the error terms in the formula are calculated for weight vector w. This weight vector is fixed for the entire observation interval. The weights are updated in the gauss Newton method by the following equation:

$$w(n+1) = w(n) - (J^T(n)\,J(n))^{-1} J^T(n) e(n) \qquad (1.5)$$

J (n) is the Jacobian matrix of the error vector e (n), $J^T(n)\,J(n)$ matrix product must be non-singular for the Gauss –Newton iteration to be computable. J (n) should be row rank to satisfy the above condition. J(n) is the N by M Jacobian matrix of e(n). N is number of iteration and m is the number of weights

$$j(n) = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \frac{\partial e(1)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_m} \\ \frac{\partial e(2)}{\partial w_1} & \frac{\partial e(2)}{\partial w_2} & \cdots & \frac{\partial e(2)}{\partial w_m} \\ \frac{\partial e(n)}{\partial w_1} & \frac{\partial e(n)}{\partial w_2} & \cdots & \frac{\partial e(n)}{\partial w_m} \end{bmatrix} \tag{1.6}$$

1.4 Linear Least-Squares Filter

Steepest descent, Newton's method and Gauss Newton method are the tools for building linear least squares filter. Linear least-squares filter has two properties. First a linear single neuron must be used. Second, cost function € (w) used to design the filter consists of the sum of error squares. The formulas for linear least-square filters are given below:

$$w(n+1) = X^+(n)d(n) \tag{1.7}$$

$$X^+(n) = \left(X^T(n)X(n)\right)^{-1}X^T(n) \tag{1.8}$$

$X^+(n)$ is the pseudoinverse of the data matrix x(n). It is evident from equation (1.8) weight vector solve the linear least square problem over an observation interval of duration n. Wiener filter is working as a limiting form of linear least square filter for an ergodic environment. Ergodic environment is described by the second order statistics. These second order statistics mainly use two parameters, these two are given below:

a) Correlation matrix of the input vector that are denoted by R_X.
b) Cross- Correlation vector between the input vector x (i) and desired response d (i) that are denoted by r_{Xd}

$$R_X = lim_{n\to\infty}\frac{1}{n}\,X^T(n)X(n) \tag{1.9}$$

$$r_{Xd} = lim_{n\to\infty}\frac{1}{n}\,X^T(n)d(n) \tag{1.10}$$

$$w_0 = \sum_{n\to\infty} w(n+1) = R_X^{-1}\,r_{xd} \tag{1.11}$$

The weight vector W_o is called the wiener solution to the linear optimum filtering problem. Linear least-square filter works like a wiener filter when the number of observations approaches infinity. Wiener filter is required second order statistics parameter and desired response. This information is not easily available in many real applications. A Linear adaptive filter is used to deal with unknown environmental. Least–mean–square filter is used to adjust the free parameters of filters. Least- mean-square filter is very closely related to the wiener filter.

1.4.1 LMS Algorithm

The instantaneous value of the cost function is used for the least mean square filter. The cost function is given by

$$€(w) = \frac{1}{2}\, e^2(n) \tag{1.12}$$

Linear neuron is used with the LMS algorithm as like a linear least square filter. The equation of the error signal is given by the following equation:

$$e(n) = d(n) - X^T(n)w(n) \tag{1.13}$$

The weight update equation of the LMS algorithm is given by equation 1.14

$$\widehat{w}(n+1) = \widehat{w}(n) + \eta\ x(n)e(n) \tag{1.14}$$

There η is the learning rate parameter that control the stability and the rate of convergence. The Feedback loop in LMS algorithm behaves as a low pass filter. Low pass filter passes the low frequency component of the error signal and suppress the high frequency components. The filtering operation is directly proportional to the learning rate parameter η. The learning rate parameter can be considered memory of the LMS algorithm. The small value of the learning rate parameter means a slow filtering operation, but more past data remembered by the algorithm. Learning rate parameter and memory have an inversely proportional relation in LMS algorithm. LMS algorithm is also called a stochastic gradient algorithm because weight vector traces a random trajectory in weight space.

1.4.2 LMS/Newton Algorithm

LMS/Newton is a gradient descent adaptive algorithm. This algorithm is based on the Newton method and optimal in the least square sense. LMS/Newton algorithm is not

simple like LMS algorithm so cannot be implemented in most practical applications. LMS/Newton acts as a benchmark for all least square adaptive algorithm due to its optimality. Convergence rate cannot be estimated in the steepest descent based least square adaptive algorithm, but in LMS/Newton method rate of convergence is predictable. This algorithm uses Input data very efficient. The weight update equation in the LMS / Newton algorithm can be written as

$$w_{k+1} = w_k + 2\mu\lambda_{avg}R^{-1}e_k x_k \tag{1.15}$$

The above equation is equivalent to the equation (1.14) when eigenvalues of the R are same. The time constant of this algorithm is independent of the weight's vector initial conditions, but generally unknown, so cannot be implemented in practice.

1.4.3 Learning Mechanism of the LMS and LMS/Newton Algorithm

Learning of the gradient descent algorithm can be explained by training the linear combiner shown in the figure 1.2 with a finite number of the data samples. For example, one data sample consists of an x vector and its associated desired response. The aim is to find a set of weights that will minimize the sum of the squares of the error for the training sample. The true wiener solution gives the minimum MSE. The solution based on the N training samples produces more MSE than the wiener solution. This is an excess mean square error if different set of N data samples selected from the same distribution for training then different MSE is obtained. The ratio of the average excess MSE and minimum MSE is called the misadjustment. The average excess MSE is obtained from training with N data samples.

$$M \triangleq \frac{(average\ excess\ MSE)}{(minimum\ MSE)}$$

$$=\frac{(number\ of\ weights)}{(number\ of\ training\ samples)} =\frac{n}{N} \tag{1.16}$$

In gradient descent algorithm exponentially relax towards the wiener solution with noise superposed. The noise is originated because estimating the gradient with signal data sample produce the noisy gradient. Due to the gradient noise the weights do not converge to the wiener solution. Mean square error is always greater than the minimum mean square error and never goes below it.

$$M = \mu\, Tr(R) = \frac{n}{4\tau_{MSE}} \tag{1.17}$$

Where Tr (R) is the trace of *R,* and *is* the time constant of the mean square error learning curve. The unit of time is the iteration cycle**.** The learning curve shown in the fig.1.3 where ε_{fin} is the asymptotic MSE, ε_{excess} is the difference between the ε_{fin} and minimum MSE ε_{min} .Widrow and hoff proved that training a neural network having only one neuron, the number of training patterns would be equal to 10 times the number of weights. There are no such rules for multilayer neural network trained with backpropagation but one may assume that for a network with many inputs and a single output, regardless of the number of neurons and layers, training with a number of patterns equal to 10 times the number of inputs should give good performance.

The learning curve is exponential for LMS/Newton algorithm with a single time constant. For practical purpose, its convergence time is of the order of four time constants. In initial learning transient the MSE is excessive so fast convergence is desirable. The LMS algorithm has a learning curve, which is the sum of exponentials but when all the eigenvalues of the R matrix are equal, LMS has a single exponential curve so LMS is optimal in this case. Generally eigenvalues are not equal and then LMS has a different kind of the learning curve than LMS/Newton. After learning transients dies out the steady state adjustment for the LMS / Newton and LMS is given by

$$M = \mu\, Tr(R) \tag{1.18}$$

The above equation is true for stationary input data. The figure 1.3 shows the influence of the initial condition for convergence of the algorithm. In the Newton method all the learning curves are identical, but in the steepest descent learning curve depend on the initial condition. For some initial conditions LMS converges faster than LMS/Newton. For other initial condition LMS is slower than the LMS/Newton. Learning time can be estimated using the Excess error energy. Excess error energy is proportional to the learning time. The below figure shows that during the transient the MSE is high. The area under the curve and above ε_{min} is defined as excess error energy. This area is a sum of MSE over time. The fig 1.5 has a single time constant. The area under this curve is equal to the amplitude of the exponential multiplied by the time constant. The learning time for the single exponential is defined as

$$learning\ time\ \triangleq 4\ \times \frac{(excess\ \ error\ energy)}{(Initial\ excess\ MSE)} \tag{1.19}$$

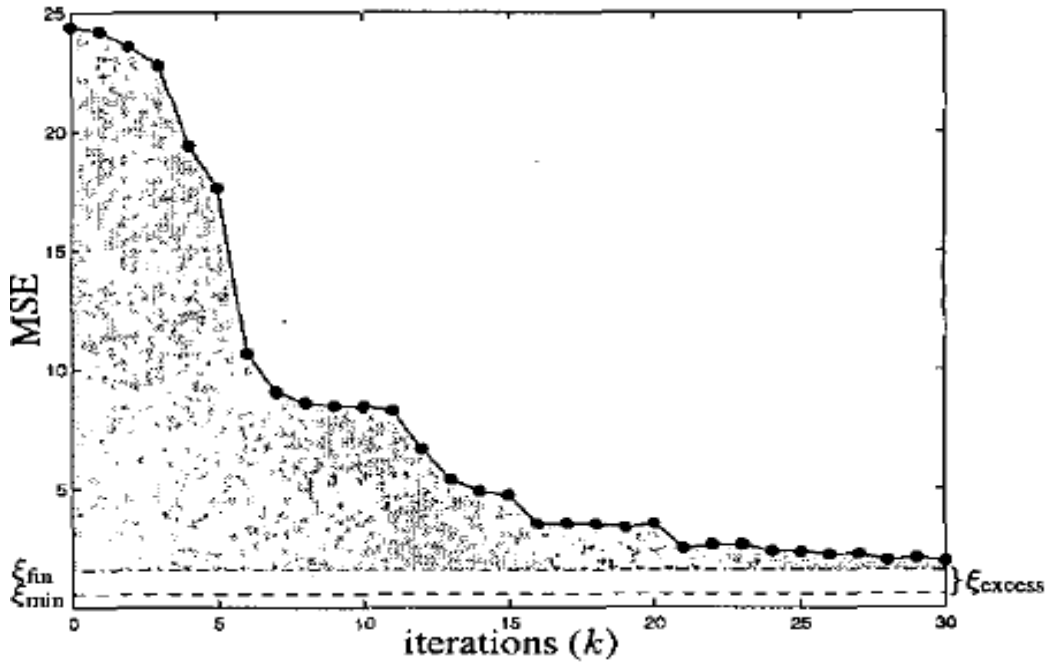


Figure 1.3 Simple learning curve with gradient noise [24]

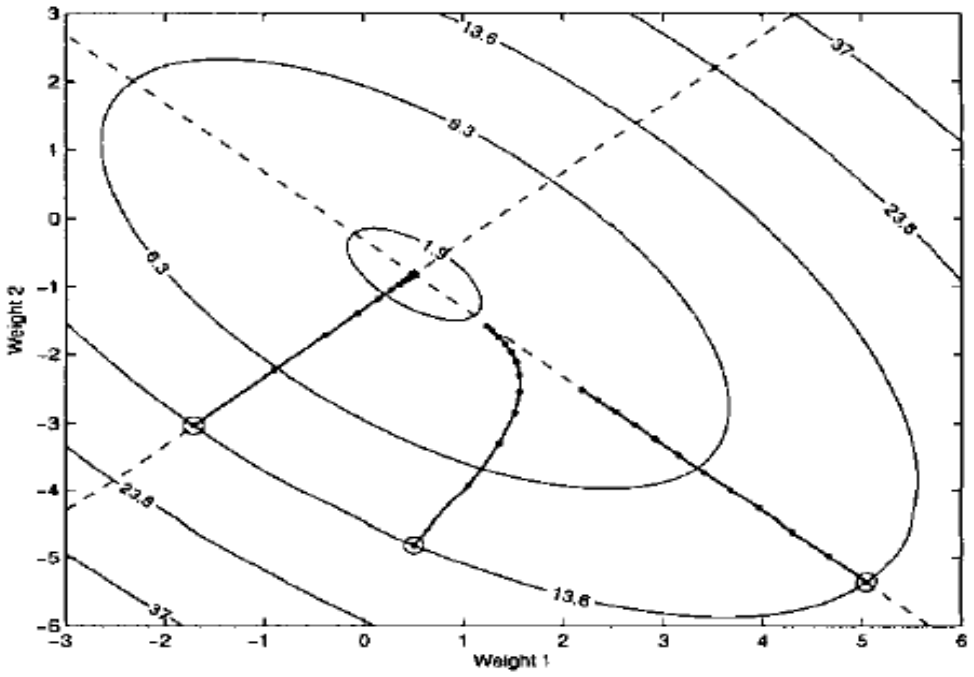


Figure 1.4 Effect of the initial condition on the LMS algorithm [24]

In general learning time of the learning curve is the sum of the any number of the exponentials. The figure 1.5 (a) and 1.5 (b) shows that average initial convergence of the LMS is faster than then LMS/Newton, whereas the average final convergence of LMS is slower than of LMS/Newton. However, their average learning times and excess error energies same. The total misadjustment for adaption with the LMS algorithm in the non stationary environment is given by

$$M_{sum} = \begin{pmatrix} Misadjustment \\ due\ to\ the\ gradient\ noise \end{pmatrix} + (Misadjustment\ due\ to\ lag)$$

$$= \mu\, Tr(R) + n\frac{\sigma^2}{4\mu\varepsilon_{min}} \tag{1.20}$$

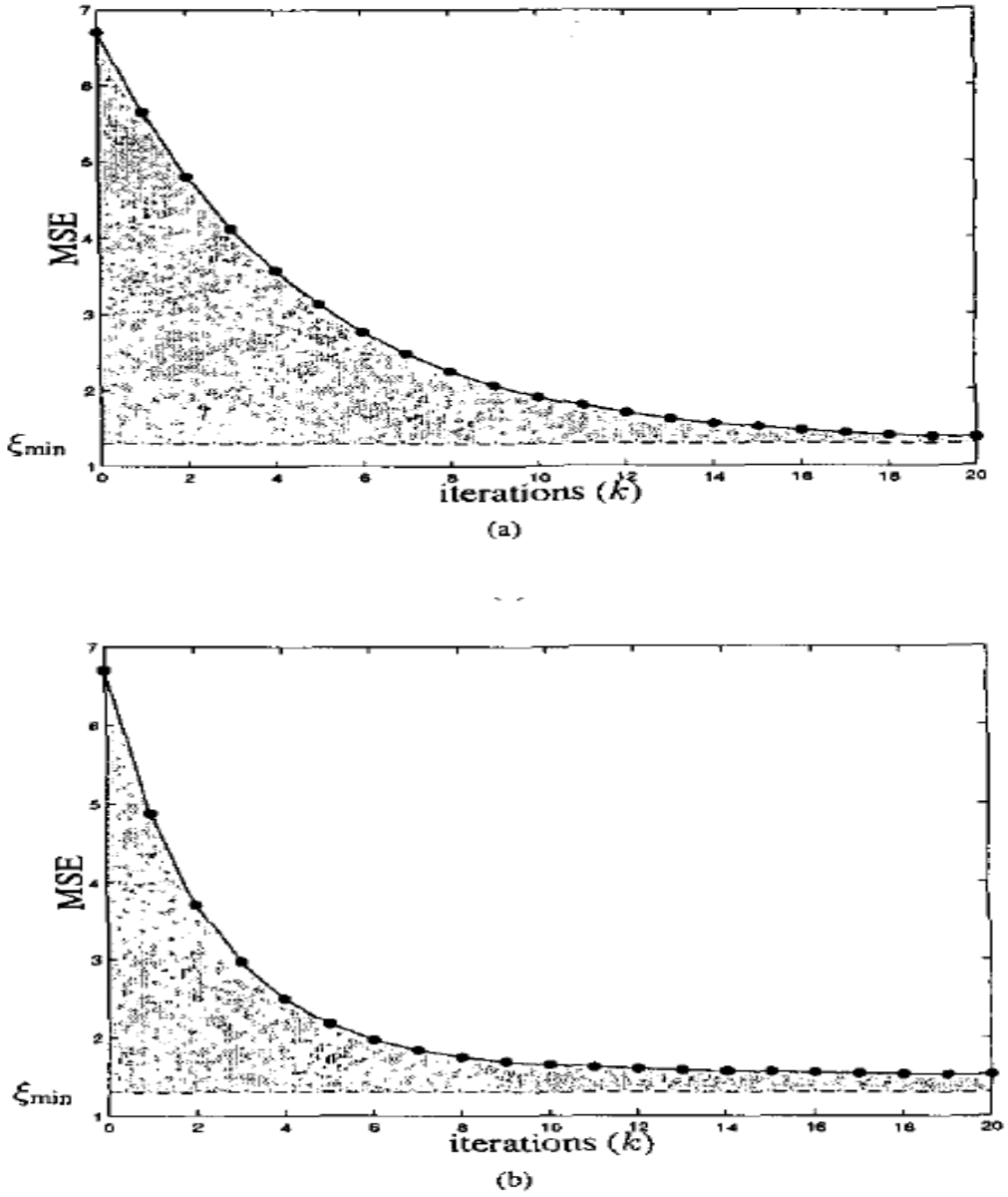


Figure 1.5 Idealized learning curves (a) LMS/Newton algorithm (b) LMS algorithm [24]

The above equation shows that the optimal choice of μ for minimize the M_{sum}. Optimization take place when these two terms are equal, that happens when the loss of performance from adapting too rapidly is equal to the loss in performance from adapting too slowly. The above equation show that M_{sum} is depends on the choice of the parameter μ and statistical properties of the non stationary environment. M_{sum} does not depend on the spread of the eigen values of the R matrix. The same expression is used with the LMS/Newton method. The performance of the both algorithms is same for the same μ in the non stationary environment and tracking the first order Markov target.

1.4.4 Virtues and Limitations of LMS Algorithm

LMS algorithm has some important properties so it is useful in many applications. Some of them are given below**:**

1. LMS algorithm is model independent and robust that means small model uncertainty and small disturbances result in small estimation errors.
2. LMS algorithm can work in stationary and non stationary environment.
3. LMS algorithm is computationally being very simple and also has a regular structure. VLSI implementation can be possible for the LMS algorithm.

LMS algorithm has many limitations some of them are listed below:

1. The primary limitations of the LMS algorithm are slow rate of convergence and sensitivity to variations in the Eigen structure of the input. Convergence of the LMS is very slow when dimensionalities of the input space become high.
2. LMS algorithm is particularly sensitive to the variations in the eigenvalue spread of the correlation matrix R_X of the input vector x .The eigenvalue spread is defined by $\chi(R_X) = \frac{\lambda_{max}}{\lambda_{min}}$,where λ_{max} and λ_{min} are the maximum and minimum eigenvalues of the matrix R_X.
3. The orientation of an Initial weight vector with respect to eigenvectors of the R matrix is not known in advanced. It is difficult to estimate the rate of convergence of the LMS algorithm. Under worse case initial condition, the learning time of the LMS is higher and this is the principal disadvantage of the LMS algorithm.

1.5 Variable Step- Size LMS Algorithm

The most important parameter in the design of the LMS algorithm is the learning rate parameter (step-size). In the fixed step size LMS algorithm the learning rate parameter is constant throughout the computation

$$\eta(n) = n_o \tag{1.21}$$

In the standard LMS algorithm convergence speed and steady-state error cannot meet at the same time. To avoid this variable step size LMS algorithm is used. In the stochastic approximation the time varying learning rate parameter is given by Robbins and Monoro .

$$\eta(n) = \frac{C}{n} \tag{1.22}$$

This equation is sufficient to guarantee convergence of the stochastic approximation algorithm. The problem from the above formula is that when the constant c is larger for small n then learning rate parameter can be out of bound. Darken and Moody had been proposed a search-then-converge schedule to overcome this problem. The new time varying learning rate parameter is given by

$$\eta(n) = \frac{n_o}{1+\left(\frac{n}{k}\right)} \tag{1.23}$$

Where n_o and k are user selected constants. Here, k is the search time constant. In the initial stage number of the iteration is small compared to the search time constant k so $\eta(n) = n_o$. The algorithm operates initially as the standard LMS algorithm if the number of iterations n large compared to the search time constant k, the learning rate parameter $\eta(n)$ can be approximated as $\frac{C}{n}$ where c=kn_o so algorithm operates as a traditional stochastic approximation algorithm. Search-then-converge method has the potential to take the advantage of standard LMS and traditional stochastic theory. In recent years, many variable step size algorithms are proposed to avoid the contradiction between the convergence speed and steady state error. Sigmoid function based variable step-size algorithm have a fast tracking speed and smaller steady state error, but the function changes fast when error close to zero. Hyperbolic tangent function based step size variation is used mean square error to track changes in the system and eliminate the effect of noise. This type of algorithm is used nonlinear relation between the step-size and the error. Some researchers use self correlation to

control the step-size. Self-correlation eliminates the effect of the noise, but this type of algorithm is sensitive to the initial step-size and slow convergence speed. The normalized LMS algorithm is useful in the stable environment, but performance in the time varying system is not so good.

1.6 Literature Review

T. Houya et al. [5] had proposed a new approach to design an adaptive filter using neural networks with symmetric weights trained by the modified momentum method, which is based on the backpropagation learning algorithm. It was found that the proposed method performs 25% better than the conventional LMS algorithm.

S. Dixit et al. [6] had proposed the adaptive noise canceller using neural network that uses least mean square adaptive algorithm. The Analog neural network had used to change the coefficients of the neural network. The neural network can optimize the coefficients of the adaptive filter at each new received sample that is useful in a non stationary environment. Due to the parallel and the analog nature of the processing, time required by the neural network for computation of coefficients is small. It was found that performance of the neural network based ANC better than LMS-ANC direct method.

Qian Xiao et al. [7] had proposed an adaptive filter based on wavelet transform method. The Hopfield neural network had used to implement the adaptive filtering algorithm LMS. After simulation, it was found that wavelet transform based neural network adaptive filter can achieve the best denoising effect.

M. Ibnkahla [8] had implemented a simplified multilayer neural network based adaptive filter. It was found that the derivation of recursions for the mean weight update that can be used to predict the weights and mean squared error over time. The effect of the step size and the initial weight values upon the algorithm behaviour has also shown in this work.

Yonggang Yan et al. [9] had proposed a novel variable step size LMS algorithm based on the hyperbolic tangent function to increase the convergence rate and eliminate the disturbances of existing independent noise. This algorithm mitigates the influence of independent noise by using the auto correlation of the current error signal e (n) and the previous error signal e(n-1).

Ting-Ting Li et al. [10] had proposed a new variable step-size LMS algorithm based on the analysis of variable step-size algorithms. The proposed algorithm establishing a new nonlinear relationship between the step size and the error, the algorithm eliminates the inapplicable noise and improves the convergence rate to obtain a better stability.

B.Burton et al. [11] had proposed a new method of the random weight change (RWC) algorithm, which is based on the method of random search for the error surface gradient. The performance of the new form of RWC is very much same as conventional back propagation with on-line training. This type of fast ANN can identify and control the motor currents within a few milliseconds.

S.K. Fathy and M.M. Syiam [12] had identified the problems of neural network backpropagation implementations on parallel processing and proposed a parallel backpropagation algorithm to decrease the communication time overhead between different related processor elements. Transputer network had used to implement the backpropagation algorithm. The developed algorithm had been experimenting with the problem of printed Arabic character recognition. After simulation, it was found that speed increases up to 15.2 when 16 transputer had utilized.

M.Stella [13] had implemented a simple neural network called adaline for noise cancellation. MATLAB software was used for simulation purpose. The experiment had been performed for engine noise cancellation in the car. Simulation results show that the SNR improves after passing through a noise cancellation system.

B.widrow et al. [14] had discussed the applications of the neural nets in adaptive pattern recognition and adaptive filtering. They also discussed the adaptive filtering algorithm LMS, MRII and MRI first rule for adaptive signal processing.

Ngia et al. [15] had proposed a new training algorithm for nonlinear adaptive filters which use multi layer feed forward neural nets as filter structures. The algorithm is based on recursive Levenberg-Marquardt (LM) search direction.The proposed algorithm was used with the echo cancellation application. It was found that LM convergence faster than the steepest descent and Gauss-Newton methods.

Kollias and Anastassiou [16] had developed a least square algorithm for the training of the ANN. The training algorithm was based on a modification of the Marquardt-

Levenberg least-squares optimization method. LM algorithm has better convergence properties than the conventional backpropagation learning technique. The performance of this algorithm had been analyzed on logic operations such as XOR and XNOR etc.

Stan and Kaman [17] had proposed a new localized algorithm called local linearized least squares for training multilayer feedforward neural network. This algorithm was developed to remove the inconsistencies found in the other localized algorithm. The objective function of this algorithm is the sum of the squares of the linearized backpropagated error signals. The simulation results show that the performance of the proposed algorithm comparable to the global extended Kalman filter.

Charalambous [18] had proposed a novel approach for the training of multilayer feed forward neural networks. The proposed algorithm use conjugate gradient algorithm. The algorithm updates the input weights to each neuron in a parallel way and has a better performance compared to conventional artificial neural network.

S.Guarnieri [19] had proposed a novel adaptive Spline activation function based neural network. Spline activation function based neural network has high representation capabilities, small interconnections in a network. These networks are used for pattern recognition and data processing, real time problems. Gradient based learning algorithm is used in this neural network model. This model has a low complexity and very effective learning capabilities.

Jiang and Kong [20] had proposed block based neural networks for ECG heartbeat pattern classification. Block based neural networks are a two dimensional array of Modular component neural networks. These components were implemented in a reconfigurable digital hardware such as field-programmable gate arrays. Connection weights and network structure was optimized using evolutionary and gradient based search operators. The Gradient based operator was used to increase optimization speed. It was observed that proposed technique is a better classification technique compare to other ECG classification technique.

A.Dinu et al. [21] has proposed an algorithm for compact neural network hardware implementation and implemented it on Xilinx FPGA. This algorithm first performs digitization of the mathematical model of ANN after that digitized model converted to logic-gate structure. A set of C++ program is used for generating very fast hardware

description language code. This algorithm is directly available only to neurons that have step activation functions. It was observed that this method bridge the gap between ANN design software and hardware design package and useful for low number of inputs and inputs with low number of bits.

A.Gomperts et al. [22] has developed hardware implementations of generalized backpropagation Multilayer architecture and implemented in Xilinx Virtex FPGA. This work was aimed to minimize the hardware cost and maximize performance, accuracy and parameterization. They described a method that offers a high degree of parameterization and performance compared to other multilayer perceptron implementations. VHDL language was used to implement the design on FPGA. The Linear interpolation technique was used for approximation of sigmoidal function that uses one adder and one multiplier.

Zhiying Guo et al. [23] had proposed a variable step size LMS algorithm based on the neural network (Backpropagation LMS) for suppressing the problem of convergence performance of the standard LMS algorithm. This algorithm was applied to the adaptive digital pre distortion system. The implemented algorithm results show that improved pre distortion amplifier better than previous algorithms.

B. Widrow et al. [24] had described the statistical efficiency of LMS algorithms. In this paper relation between LMS algorithm and backpropagation algorithm used for training of the neural network was described. They described the similarity between two gradient descent algorithms LMS and LMS/NEWTON and issues related to implementations.

S.Himavathi et al. [25] has proposed a hardware efficient multilayer feed forward neural network and implemented in Xilinx FPGA XCV400hq240. Instead of using many layers, single largest layer was used with the help of layer multiplexing to reduce resource requirement. It was found that this method utilize a less number of hardware resource compared to other methods. The percentage saving in hardware resources increases when number of layers increases in the network architecture with moderate overhead in speed.

J.Hertz et al. [26] had proposed a new backpropagation algorithm that avoids the calculation of the derivative of the activation function. This algorithm replaced the standard backpropagation algorithm by non linear gradient descent approach. The

designed algorithm was used with net talk problem. Further, it was observed that the performance of the proposed algorithm very similar to the standard backpropagation algorithm and easier to implement in the electronic hardware.

A.Cichocki and R.unbehauen [27] had proposed an on-chip learning algorithm for proper calculation of principal component and minor component. For this they developed a large class of the loss functions and minimized by gradient descent approach. Non-linear squashing function was used to implement this algorithm. The advantage of this algorithm is that it reduces the interconnection between the processing units and ensures proper on chip learning.

Xiangyu Kong et al. [28] has proposed unified learning algorithm for principal component analysis and minor component analysis. In the unified learning algorithm is working as a minor component extractor simply by altering the sign. Implementations of this algorithm are very easy. The algorithm can be extended for tracking the principal subspace and minor subspace. The algorithm was analyzed by the fixed-point analysis method. The result shows that the proposed algorithm outperforms compare to many existing unified algorithms.

David Hunter et al. [29] have proposed a partial solution to use the least number of the neurons with a high number of training patterns. The proposed solution was used in the error backpropagation algorithm (EBP), Levenberg Marquardt (LM) algorithm and Neuron-by-Neuron (NBN) algorithm. The efficiency of the different network topologies was also discussed. The training tool for NBN algorithm has developed that is capable of handle arbitrarily connected neural network. FCC topology was also optimized in this work.

Antony W. Savich et al. [30] has designed the multilayer perceptron trained using the backpropagation algorithm for analyzing the effect of arithmetic representation. The MLP-BP model was coded in the VHDL and then implemented in the FPGA. In this study, they were also shown the effect of the Fixed point format and Floating point format for the hardware resource requirement and speed of operation. It was found that MLP-BP requires a less clock cycle and less hardware resource when compiled in fixed point format compared to floating point format. Further, it was observed that the resource requirement two times less when the network implemented in the FXP format for similar precision and range of the representation.

1.7 Motivation and Objective

The LMS algorithm described above is widely used in the design of adaptive filter which are being used in real world scenario such as noise cancellation, signal enhancement, linear prediction and plethora of signal processing task.

Apart being implemented in software, some authors have reported FPGA and ASIC implementation of LMS adaptive filters. However, it is felt by the author that implementation of artificial neural network trained with backpropagation algorithm could be a better way of achieving the desired signal processing objective since weight/coefficient update mechanisms in backpropagation algorithm is primarily based on the method of least mean square. In addition to implementation of adaptive filter a neural chip can also serve a real time classifier, repressor and a multipurpose DSP processor such an implementation is likely to result general purpose deliverable in the form of novel hardware design. In subsequent sections the similarity between pure LMS filter and backpropagation trained neural network will be presented. However, in this case we would like to make crucial training parameters like learning rate an adaptive one so as to cater to the needs of different data sets. This has served as the primary motivation for the author to investigate the learning performance of an artificial neural network trained with backpropagation algorithm and an adaptive learning rate. FPGA implementation of such a network has also been envisaged in this dissertation. In a nutshell, following objectives can be outlined for this dissertation.

1. To devise a mechanism for making learning rate adaptive so that classification success rate improves with minimum overhead in complexity.

2. To implement the devised technique on FPGA.

3. To implement a general purpose ANN on FPGA with the adaptive learning rate.

In subsequent sections the similarity between pure LMS filter and backpropagation trained neural network will be presented

.

1.8 Novel Aspects of this Dissertation

The work presented in this dissertation brings forward some novel analysis, methodologies and implementation results which are being enlisted as follows:

1. To the best of the author's knowledge this has been the first attempt to implement a PCA based learning rate variation technique on to the MATLAB.
2. Unlike the previous workers, the author has applied PCA on the output feature space rather than on the input itself.
3. The author has proposed a multiplexer based technique to reduce the number of multipliers used which has resulted in saving scarce computational resources.

1.9 Organization of Dissertation

In order to achieve the previously discussed aims this dissertation is organized into the following seven sections:
Chapter 2 gives an overview of backpropagation algorithm and discussed the weight update mechanism to adjust the synaptic weights of the neural network.

Chapter 3 describes the Principal component analysis algorithm and the learning rate parameter adaption using the Principal component analysis.

Chapter 4 describes the classification of the neural network, issue related to the hardware implementation of the neural network and requirement for the FPGA implementation of artificial neural network.

Chapter 5 describes the FPGA implementation of the artificial neural network using multiplexer based weight updating for effective resource utilization.

Chapter 6 describes the results and discussion. The results are divided into three parts: Simulation results of PCA based learning rate variation, FPGA synthesis results, and ASIC synthesis results. These results illustrate the functionality, device utilization summaries, timing analysis and the power consumption analysis of the neural network.

Finally, Chapter 7 sums up the conclusions and future scope of this work

CHAPTER 2

WEIGHT UPDATE MECHANISM

2.1 INTRODUCTION

Backpropagation algorithm is the generalized form of the ubiquitous least mean square algorithm. Backpropagation algorithm is used to train multilayer feed forward network. The Multilayer feed forward network consists of an input layer, one or more hidden layers and output layer computational nodes. The input signal propagates through the layer by layer in forward direction. These neural networks are also called multilayer perceptron. The backpropagation algorithm is the most popular algorithm for the supervised training of multilayer perceptrons. Backpropagation is a gradient technique not an optimization technique. Backpropagation algorithm is also known as error back-propagation algorithm. Error backpropagation algorithm learning has two passes through the different layers of the network.

1. **Forward pass:**

   In the forward pass an input vector is applied to the input node of the network and its responses propagates through the network, layer by layer**,** finally a set of output is produced. In the forward pass the synaptic weights of the network are all fixed.

2. **Backward Pass:**

   In the backward pass synaptic weights are modified in accordance with the error correction rule. The error signal is propagated through the network. The direction of propagation is opposite to the synaptic connection. The synaptic weights are adjusted to make the output of the network move closer to the desired response.

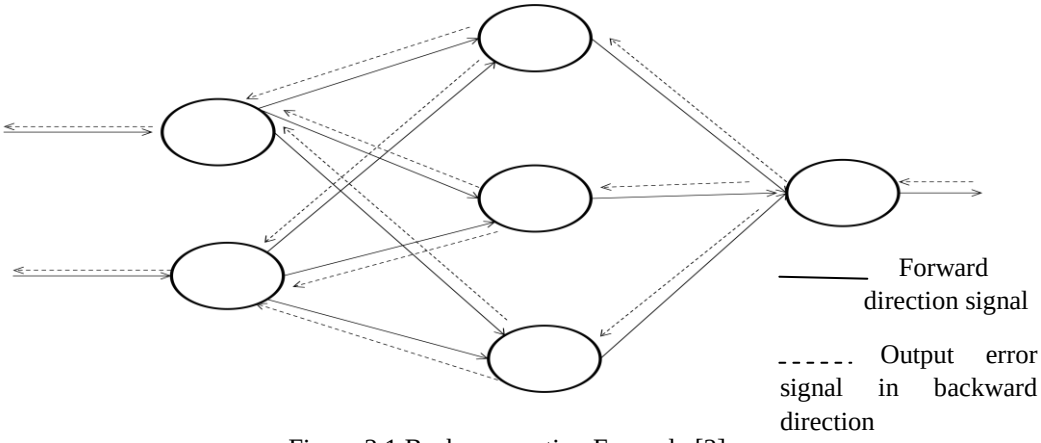


Figure 2.1 Backpropagation Example [2]

Back-propagation has two properties:

1. It is simple to compute locally.
2. It performs pattern by pattern updating of synaptic weights.

These two properties are responsible for the advantage and disadvantage of backpropagation learning.

2.2 Salient Features of Backpropagation Algorithm

Backpropagation algorithm learning performance is affected by many factors. These factors are initial parameters, learning rate, network size and learning database. The Optimum value of these parameters may be speeding up the learning process. Influence of the some of these is described below:

1. **Random selection of initial weights:**

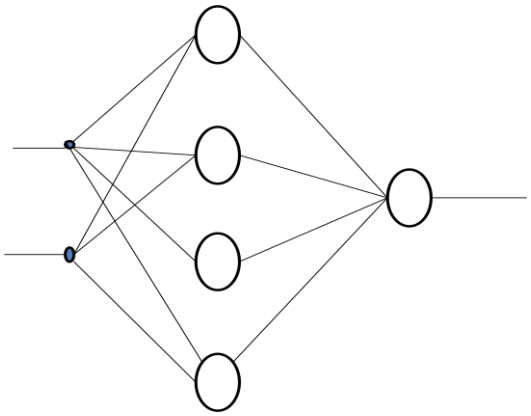
   The backpropagation algorithm learning procedure is largely affected by the initial value of the weights. Theoretically learning rate has no relationship with the initial value of the weights. The Initial value of the weights should be close to the one of the global minima in the weight space. In real life problem global minimum information is not present so initial weights are chosen by choosing of few sets of weights and calculates the error function for these different sets. Weights corresponding to the minimum error function are used as initial weights.

2. **Learning rate adaptation:**

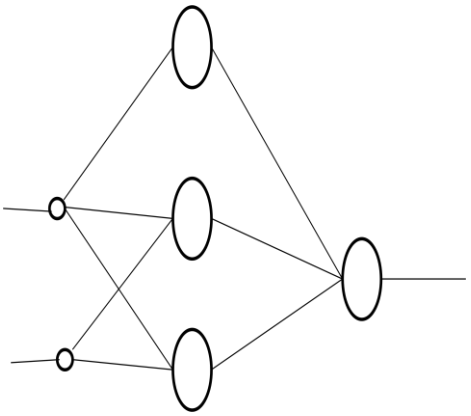
   The learning rate parameter η is determined the increment of the weight at every updating step. A Small learning rate η may slow down the learning procedure and usually kept small to prevent oscillations. A large learning rate is required to move the weights rapidly toward the minimum point in weight space. The learning rate parameter should be chosen so that it reduces the learning time and to obtain faster convergence. To make learning rate adaptive several mechanisms has been proposed. In one mechanism learning rate parameter to increase if the total error function is decreased and to decrease if new error is increased by some pre specified ratio. In the other method if the gradient vector has the same sign for several iterations the corresponding learning rate is increased. When the gradient component changes the sign for several consecutive steps, the corresponding learning rate decayed exponentially.

3. Number of hidden layers and neurons

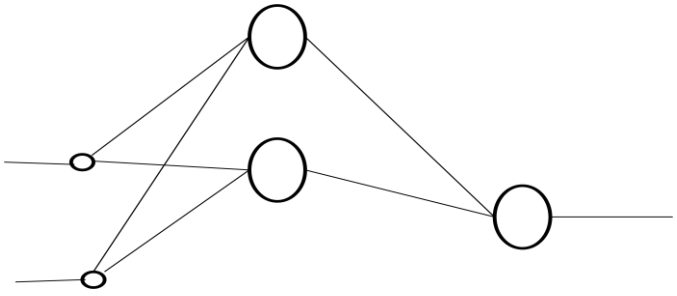
The number of hidden layers and hidden neuron are not known in advanced. Optimum number of hidden layer and neurons are obtained by trial and error. Number of hidden layer and neurons should be reduced to decrease computation complexity. The Network should have a small number of neurons, layer and input so that network should be capable of performing a given task. This type of structure is called optimum structure. Number of output neurons can be easily determined for a specified task. There is no fundamental rule to determine hidden units and input. In many works, it has been seen that two hidden layers solves the problem easier than a network with a single hidden layer. Two approaches are used to obtain the optimum number of hidden layer and neurons. The first approach is used; a small number of neurons and hidden units initially when learning procedure trapped in local minima then new input and hidden units are added. In the second approach a large network is created and gradually removes redundant units. This iterative procedure is used for monitoring and examining the occurrence of local minima during the learning.



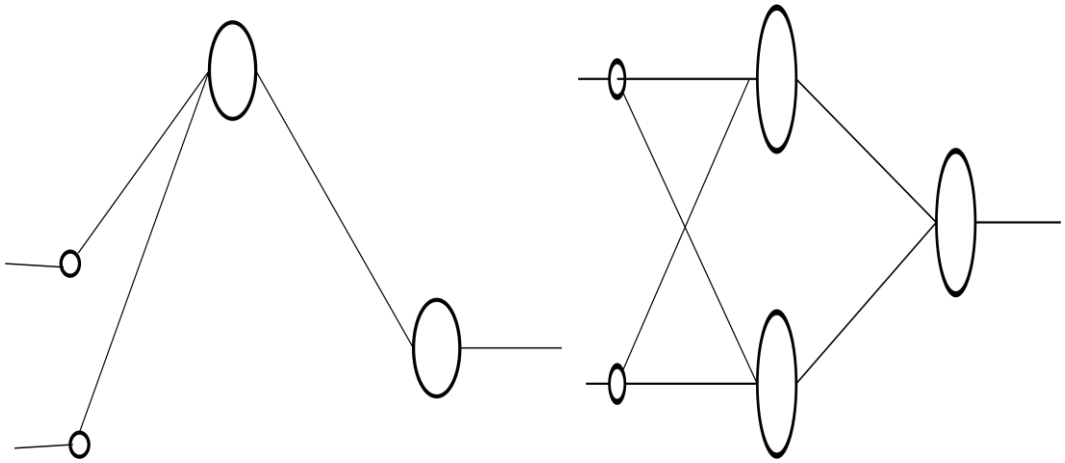
Initial network (a)



deleting a unit (b)



Deleting another unit (c)



The network is uncapable of learning (d) optimal structure for learning (e)

Figure 2.2 Optimizing network size by dynamically deleting layer unit

4. **Local minimum Problem:** The ideal learning technique should be search optimum value of the weight to obtain global minimum point of the error function. The gradient descent method based backpropagation algorithm may be reached a local minima or saddle points. This local minimum is related to the high level of error surface. Local minima occur when $\frac{\partial e}{\partial w}$ is zero or small but the error is still at a very high level. Gradient descent searching is not able to reach from toward the global minima. To avoid this situation learning procedure is started with new initial weights, a new learning rate parameter and a network size.

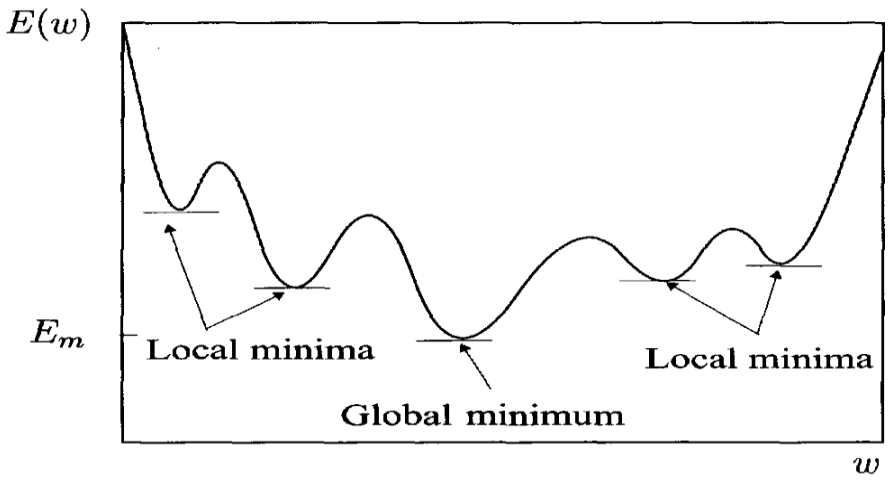


Figure 2.3 Global minima and local minima of the error function [2]

In Gradient descent based learning the speed of the learning is very slow near the global minimum point. A Small learning rate is required during this period of learning.

5. **Connectionism:**

The back-propagation algorithm is depending on local computations. Local computations decide the information-processing capabilities of the neural network. Local computations allow the use of parallel architectures for the implementation of artificial neural network. Back-propagation algorithm has been implemented on parallel computers. VLSI architectures have been developed for the hardware realization of multilayer perceptron. Local computation of the artificial neural network is similar to the biological neural networks, but some points are against this similarity these points are given below:

(a) Hormonal and other types of global communication are ignored in a multilayer perceptron. In a real nervous system these global communications are necessary for state setting functions.

(b) In the backpropagation learning information are propagated backward along on the axon. This type of operation doesn't take place in the human neural system.

(c) Backpropagation learning is based on supervised learning means requirement of teacher. Supervised learning will be possible in the human brain when a set of neuron with novel properties are present. The existence of such neurons is biologically impossible.

6. **Computational Efficiency:**

The computational complexity of the artificial neural network is estimated in terms of the number of additions, multiplications and storage requirement. Backpropagation algorithm is computationally efficient because computational complexity is polynomial in the number of adjustable parameters. In backpropagation algorithm computational complexity is linear in W (synaptic weight) and given by O(W).

7. **Sensitivity Analysis:**

Backpropagation algorithm is the efficient way to Sensitivity analysis of the input-output mapping. The sensitivity of the input, output mapping function F with respect to the parameter w of the function is given by $= \frac{\frac{\partial F}{F}}{\frac{\partial w}{w}}$ **.** The Complexity involved in computing each of these partial derivatives is linear in W.

8. **Robustness:**

LMS algorithm is robust because noise with small energy only gives rise to small estimation error. The model is linear and the LMS algorithm is an H^{∞} -optimal

filter. H^{∞} -optimal filter means LMS algorithm minimizes the maximum energy gain from the noise to the estimation error. Backpropagation algorithm is a locally H^{∞} -optimal filter. LMS and backpropagation algorithm belong to the same class of H^{∞} -optimal filter.

2.3 The Backpropagation Algorithm [32]

In back-propagation learning, synaptic weights are modified by pre synaptic activity and error signal. These signals are independent from the post synaptic activity. Back propagation algorithm is used following steps to update the weight of the hidden and output neurons when a finite length input patterns $x_1(k), x_2(k), x_3(k) \ldots\ldots x_n(k)$ ($1 \leq k \leq K$) and desired $d_1(k), d_2(k) d_3(k) \ldots\ldots\ldots\ldots d_m(k)$ patterns are given:

Step 1: In step 1 total number of layers M and number of neurons in the hidden layer are determined.

Step 2: Randomly selection of the initial value of the weight vector.

Step 3: In the forward pass functional signals of the network are computed on

a neuron by neuron basis. The functional signal appearing at the output of neuron j is given by

$$y_j(n) = \varphi(v_j(n)) \tag{2.1}$$

Where $v_j(n)$ is the induced local field of neuron j, defined by

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{2.2}$$

Where m is the total number of inputs applied to neuron j, and $w_{ji}(n)$ is the synaptic weight connecting neuron i to neuron j and $y_i(n)$ is the input signal of neuron j or equivalently. The forward pass of computation start at the first hidden layer by introducing with the input vector and terminates at the output layer by computing the error signal for each neuron of this layer.

Step 4: Calculate the output error

$$e_j(n) = d_j(n) - y_j(n) \tag{2.3}$$

Neuron j is an output node.

Step 5: Calculate the output delta

$$\delta_j(n) = e_j(n)\, \varphi'_j(v_j(n)) \tag{2.4}$$

Step6: Recursively calculate the hidden neuronal delta values

$$\delta_j(n) = \varphi'_j\left(v_j(n)\right)\sum_k \delta_k(n)\, w_{kj}(n) \tag{2.5}$$

Step7: update weight vector for hidden and output layer is given by the following equation

$$\Delta w_{ij}(n) = \Delta w_{ij}(n-1) + \eta\delta_j(n)y_i(n) \tag{2.6}$$

Where, η is the learning rate parameter and $\Delta w_{ij}(n)$ is the change in weight incident on neuron j from neuron *i*. here $\Delta w_{ij}(n) = \eta\delta_j(n)y_i(n)$.

2.4 Sequential and Batch Modes of Training

One complete presentation of the entire training set during the learning process is called an epoch. The learning process is continuing on an epoch-by-epoch basis until the synaptic weights and bias level approaches to the optimum value so that the average squared error converges to some minimum value**.** Back-propagation learning can proceed in one of the two basic ways.

1. **Sequential Mode**: In the sequential mode of operation weight updating is performed after the presentation of each training example. For example, consider an epoch consisting of N training examples arranged in the order $(X(1), d(1),), \ldots\ldots\ldots (X(N), d(N))$.The first example pair (X (1), d (1)) in the epoch is applied to the network and the sequence of forward and backward computations is performed. Synaptic weights and bias levels of the network are adjusted by these operations.

2. Batch Mode: In the batch mode of back-propagation learning, weight updating is performed when all the training examples in an epoch are presented. For a particular epoch the cost function as the average squared error is given by

$$\varepsilon_{av} = \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in C} e_j^2(n) \tag{2.7}$$

The inner summation is performed over all the neurons in the output layer of the network k. Outer summation with respect to n is performed over the entire training set in the epoch. Sequential mode of training is preferred over the batch mode because this mode requires less storage for each synaptic connection.

CHAPTER 3

LEARNING RATE ADAPTATION USING PRINCIPAL COMPONENT ANALYSIS

3.1 Karhunen–Loeve Transform

Karhunen–Loeve transform is also called principal component analysis. Principal component analysis is decreased the dimensionality of the data set. Principal component analysis extracts the data, remove unwanted or redundant information, shows hidden features and describes the main relationship that exist between observations. PCA is a very powerful technique, it has been applied in signal processing for speech processing, image processing and Pattern classification[3]. Principal component analysis is based on the significance of the information. PCA estimates the direction of signal with maximum energy and variance. The Principal component of vector set X is identified by the following procedure:

First X is represented by a $X_{M\ X\ N}$ matrix where M is represent the number of vector and N represents the dimension of the vectors x_i. $X=\begin{bmatrix} x_{11} & x_{12} \dots \dots & x_{1n} \\ x_{21} & x_{22} \dots \dots & x_{2n} \\ x_{m1} & x_{m2} \dots \dots & x_{mn} \end{bmatrix}$

PCA Algorithm:

STEP 1. Obtain the mean vector and mean vector is given by the following equation:

$$U_x = \frac{1}{M}\sum_{i=0}^{m-1} x_i \tag{3.1}$$

STEP 2. Obtain the covariance matrix and covariance matrix is given by the following equation:

$$C_x = \frac{1}{M}\sum_{i=0}^{m-1}(x_i - u_i)(x_i - u_i)^T \tag{3.2}$$

STEP 3. Obtain the eigenvectors and eigenvalues are given by the following equation:

$$C_x E = \boldsymbol{\lambda}\, E \tag{3.3}$$

Where E is the Eigen vector and $\boldsymbol{\lambda}$ is the Eigen value.

a. eigenvalue:

$$\det\{C_x - \lambda_N I\} = 0 \tag{3.4}$$

b. eigenvectors:

$$(C_x - \lambda_k I)E = 0 \tag{3.5}$$

STEP 4. Analyze:

a. Projection Definition

$$a_m = x_i^T E_m \tag{3.6}$$

b. The compression ratio shows that the ratio between the total number of dimensions and the number of subspaces acquired by PCA.

c. compression

The above algorithm can be expressed in simple block diagrams that are given below in fig. 3.1. After applying the PCA, a subset of eigenvectors that are related to highest eigenvector are identified and projected on the orthogonal axes. They are aligned with the directions of the highest variances. The resulting data look like a cloud. Each cloud is the subspace of the vector set X. The significance of each subspace is represented by the respective eigenvalue. Lower significant Eigen value is discarded. This approach reduces the data dimensionality without significant loss of information.

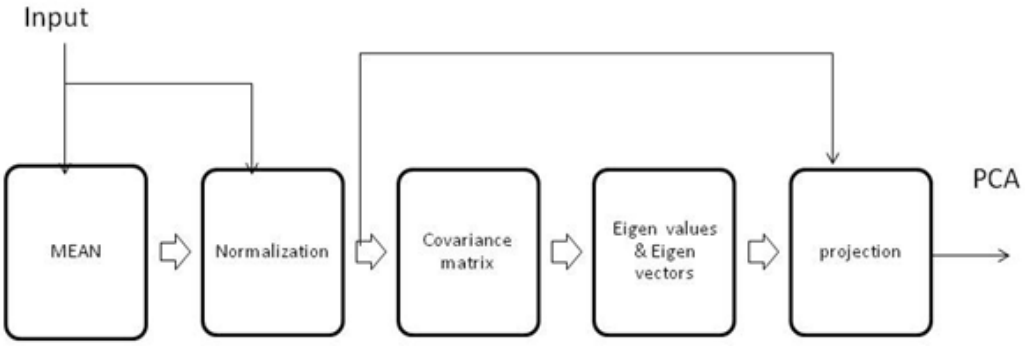


Figure 3.1 operation of the PCA

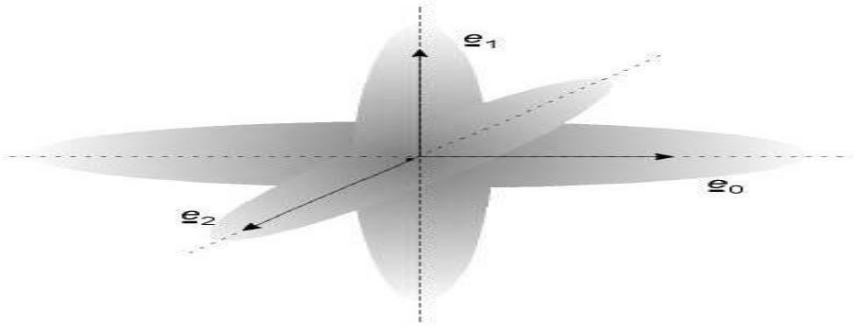


Figure 3.2 Orthogonal axes of the dataset [3]

3.2 Learning Rate Adaptation using Principal Component Analysis

For linear neuron structure learning rate parameter can be changed according to the principal component of the input distribution. This approach is similar to the hebbain-type adaption rule for its synaptic weight. The Hebb's rule is said synaptic weight W_i varies with time and growing stronger when presynaptic signal and post synaptic signal coincide with each other so weights are updated by the following equation:

$$w_i(n+1) = w_i + \eta y(n)x(n) \tag{3.7}$$

Where y, x is the presynaptic and postsynaptic signal respectively. Hebbian algorithm is computationally inexpensive to find the first l Eigenvectors of the correlation matrix R. R is the correlation matrix of input data vectors. The reason behind this above statement is that if input data vectors have a large dimension, the correlation matrix R_{xx} is very large. For example m by 1 input vector have m by m order correlation matrix R so computation requirement is very large. In Hebbian learning algorithm need not to compute the correlation matrix R so computational saving is enormous. The Generalized Hebbian algorithm can be used with adaptive learning rate $\eta(k)$ in order to improve convergence and exactness. Generalized Hebbian algorithm is based on feed forward connections for principal component analysis. In adaptive principal component extraction algorithm (APEX) uses both feed forward and feedback connections. Optimum learning rate in APEX algorithm is given by the following equation:

$$\eta_{j,opt} = \frac{1}{\sigma_j^2(n)} \tag{3.8}$$

Here $\sigma_j^2(n)$ is the average output power of neuron j or variance of the output. The above equation shows that the variance of the output can be used for learning rate parameter variation. Variance of the output is related to the eigenvalues of the correlation matrix of the random vector X (n). The above discussion can be used to implement the Principal component analysis based adaptive learning algorithm. In this algorithm each neuron in the output layer of the network should be linear and network has fewer outputs than inputs

3.3 Learning Rate Variation in Backpropagation Algorithm using PCA

The concept of the APEX algorithm can be used for computing the optimal step size in gradient descent algorithms. The on-line version is computationally efficient and it

is applicable to large backpropagation networks trained for the large data sets. Hessian matrix is the second order derivative of the objective function with respect to the weight vector w. $\varepsilon_{av}(w)$ is the cost function. Hessian matrix of the cost function is given by $H = \frac{\partial^2 \varepsilon_{av}(w)}{\partial w^2}$. The Hessian matrix has an important influence in the neural network some of them are following:

1. The Eigenvalues of the Hessian matrix have a profound influence on the dynamics of the back-propagation learning.
2. The inverse of the Hessian matrix provides a basis for pruning or deleting insignificant synaptic weights from a multilayer perceptron.
3. Hessian matrix is a second order optimization method as an alternative to backpropagation learning.

The Eigen structure of the Hessian Matrix has a profound influence on the convergence properties of the LMS algorithm. Backpropagation algorithm is the generalized form of LMS algorithm so hessian matrix also has an influence on backpropagation algorithm in a more complicated way. The eigenvalues of the hessian matrix of the error surface for a multilayer perceptron trained with the back-propagation algorithm has the following composition

- A small number of small eigenvalues
- A large number of medium-sized eigenvalues
- A small number of large eigenvalues

The factors affecting the composition of the eigenvalues as follows:

1. Nonzero-mean input signals or induced neuronal output signals
2. Correlations between the elements of the input signal vector and correlation between neuronal output signals.

The learning rate of LMS algorithm is sensitive to variations of the condition number $\frac{\lambda_{max}}{\lambda_{min}}$ where $\lambda_{max}$the largest Eigen value of the hessian and λ_{min} is smallest nonzero Eigen value. The same result is also hold for the back propagation algorithm. For an input with nonzero mean, the ratio $\frac{\lambda_{max}}{\lambda_{min}}$ is larger than compare to Zero mean inputs. Hessian matrix is the computationally expensive. The below mentioned algorithm is used for estimating the principal Eigen values and eigenvector of the objective functions of the second derivative matrix (Hessian). The algorithm is not required to

calculate the hessian. Figure 3.3 and figure 3.4 show the optimal learning rate in the gradient descent algorithm. The direction of largest second derivative is the principal eigenvector of H, and the largest second derivative is the highest Eigen value. Optimum learning rate is the inverse of the largest Eigen value of H and it is given by the following formula $\eta_{opt} = \frac{1}{\lambda_{max}}$.

The Eigen structure of the hessian is intricately related to the learning rate parameter and Hessian can only be defined as the output layer since hidden layer doesn't produce error term so any attempt to vary the learning rate should take into account the variance of the data, converging in each output node. This is served as the motivation of the author to vary the learning rate, according to the Eigen structure of the output feature space.

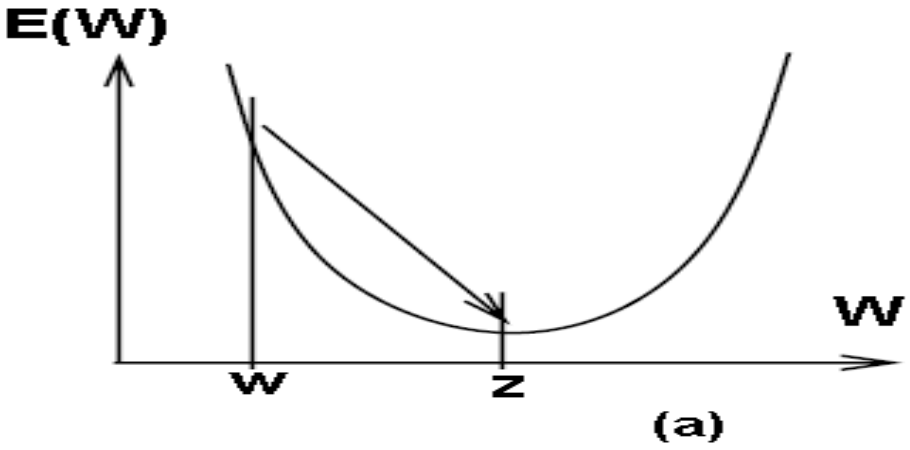


Figure 3.3 gradient descent with optimal learning rate in one dimension [4]

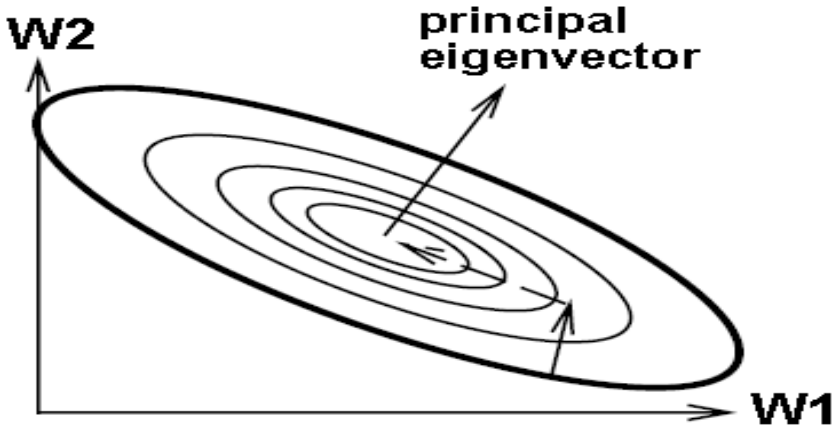


Figure 3.4 Gradient descent with optimal learning rate in two dimensions [4]

The PCA based variable learning rate algorithm is described below:

START:

FOR signal samples x (k): FROM K=1 to N

(1) Set error goal e

FOR every neuron from j=1 to m

(2) Set W_j (0) randomly

(3) Initialize learning rate parameter η to a small value

FOR epoch index M from M=1 to Max _M

(4) Obtain output Y (k)

(5) Weight updation term W_j (k)

If Wj (k-1) - W_j (k) < e then

(6) $W_j = W_j$ (k), GO to EXIT

(7) Apply PCA on the output

(8) Extract the highest Eigen value, Eigen value show variance capture.

(9) $\eta(n+1) = \eta(n) +$ MAX (Variance capture by using PCA * variance capture by PC1, PC2)

Where Pc1, Pc2 are the highest Eigen values.

EXIT

Set the output.

CHAPTER 4

HARDWARE IMPLEMENTATION OF THE NEURAL NETWORK

4.1 INTODUCTION

Artificial neural networks (ANN) became a solution for a wide variety of problems in many fields, some of the fields reached a hardware implementation phase either commercial or with prototype. A vast majority of neural networks is still implemented in software on sequential machines. The implementation might be too expensive and too slow when implemented in software so dedicate hardware should be an interesting solution. Classification of the implementation types of the hardware solution is a difficult task. The neural network hardware can be classified into the following categories according to the Fig 4.1. The Fig 4.1 shown that the global hardware solution is called neurocomputers, which can be divided into standard chips or neuro chips. The standard chips can further be classified either by sequential and accelerator boards or multi-processor solutions. The neurochips, which are constituted by application specific integrated circuits (ASICs), can be classified either as analog, digital or hybrid.

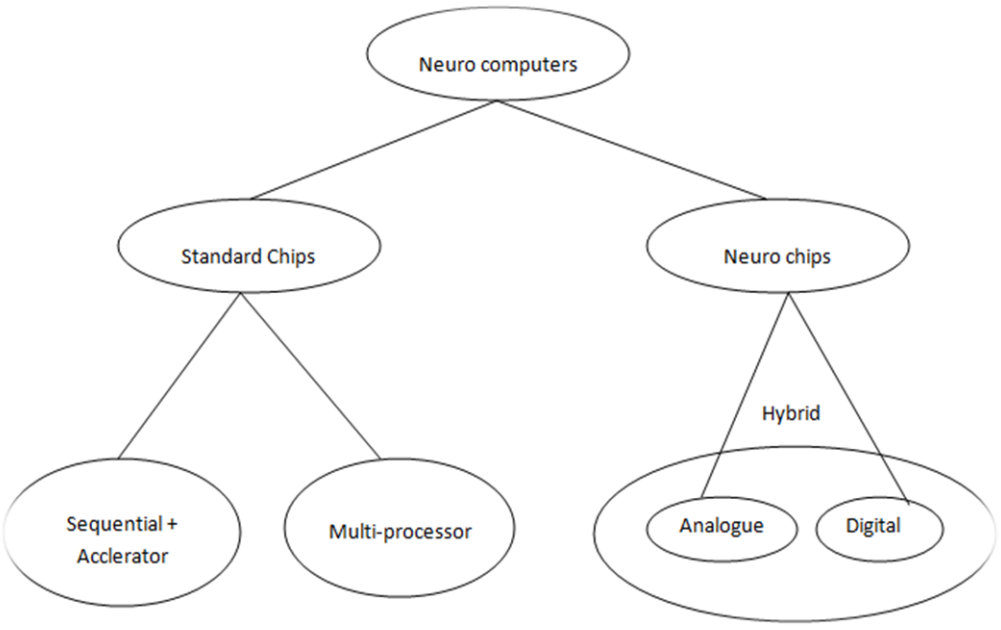


Figure 4.1 Neural networks hardware categories [31]

Neural networks have several types of parallelism and a careful inspection of these is required in order to both determine the most suitable hardware structures as well as the best mappings from the neural-network structures onto given hardware structures. For example, parallelism can be of the SIMD type, MIMD type, bit-parallel, word-parallel and so forth. In general fully parallel implementation in hardware is not feasible. Virtual parallelism is necessary and this in turn implies some sequential processing. The Learning algorithm is required to train the neural network. Many supervised neural network training algorithms are present like Levenberg-Marquardt algorithm, gradient descent algorithm and complex conjugate algorithm are the type of supervised training algorithm. The basic concept of these algorithms is "minimum disturbance principal". The principal is said to inject new information into the network in this way that modified the new information to the smallest extent. These algorithms can also be implemented serially and parallely. In reality, neural network have five types of parallelism. These parallelisms described below:

1. **Training parallelism:**

   In the neural network different training session can be run in parallel. The example of this type of processor is single instruction multiple data and multiple instruction multiple data processors. The level of parallelism in this level is medium (close to 100) so fully mapped onto current FPGAs.

2. **Layer parallelism:**

   In the multilayer perceptron trained with backpropagation algorithm has a different numbers of layers. These different layers can be processed in parallel. Density of parallelism at this level is low approximately ten. This type of parallelism can be exploited through pipelining**.**

3. **Node parallelism:**

   Node parallelism is related to the individual neurons. Node parallelism is the most important level of parallelism. If the node parallelism is fully exploited then all above mentioned parallelism should be fully exploited. Node parallelism is mapped easily in the FPGA because of the FPGAs large numbers of the cells operate in the parallel.

4. **Weight parallelism**:

   The output of the neural network is calculated by the following equation $Y = \Phi(\sum_{i=1}^{n} w_i x_i)$ where w_i is a weight and x_i is an input. The product of these can be

computed parallel. The sum of these products can be calculated by using adder-tree of logarithmic depth (high parallelism).

5. **Bit –level parallelism:** Bit-level parallelism is depending on the different number of the functional unit. The wide variety of the bit-level parallelism is available, for example: bit serial, serial-parallel, word-parallel etc.

These parallelisms can be implemented in the FPGA easily. FPGA structure can be considered alternative to software.FPGAs may be able to deliver better cost performance ratios on given applications. Moreover, the reconfiguration means that may be extended to a range of applications, e.g. several different types of neural networks. Thus the main advantage of the FPGA is that it may offer a better cost performance ratio than either custom ASIC Neurocomputers or general-purpose processors with more flexibility than the ASIC. The advantage of the FPGAs is changes in density and speed.As stated above, FPGAs offer a cheap, easy and flexible choice for hardware implementations. They also have several specific advantages for neural implementations. Some of them are listed below:

1. Reprogrammable FPGAs permit prototyping:

In most applications, several neural architectures must be tested so as to find the most efficient one. Moreover a good architecture that has been designed and implemented may be replaced later by a better one without having to design a new chip.

2. On-chip learning:

On-chip learning is often considered as difficult and useless. It is used very rarely, on-chip learning usually results in a loss of efficiency in a hardware implementation. It requires some specific operators, a higher precision, etc. In a reconfigurable FPGA, on-chip learning may be performed prior to a specific optimized implementation of the learned neural network on the same chip.

3. Embedded application:

FPGAs may be used for embedded applications, when the robustness and the simplicity of neural computations are most needed, even for low scale productions. FPGA-based implementations may be mapped onto the new improved FPGAs, which is a major advantage. In recent period FPGA speeds and areas approximately double each year. Even large neural networks may soon be implemented on single FPGAs, provided that the implementation method is scalable enough. Here the global

hardware solution is called Neuro computers which can be made of standard chips or neurochips. The standard chips can be classified either by sequential and accelerator boards.

4.2 Performance Evaluation of the Neural Network

Proper evaluation of the neural network requires two things. The first thing is the matrices and benchmarks used to obtain the measurements. Most commonly used matrices are connections-per-second (CPS) and connection update per second (CUPS).The connection-per-second is defined as the rate at which neuron multiply-add operations carried out. Connection updates per second is defined as the rate at which updates of weights are carried out. There are different problem are associated with these matrices: 1) CPS and CUPS cannot be used for all types of networks Ex: Radial basis function networks.2) Large values of the CPS and CUPS doesn't mean better performance in contrast of the different algorithm.

4.3 Basic Requirements for ANN Design

The artificial neuron is the basic element of the neural network. The design of the neuron is the foundation of the backpropagation (BP) neural network. The BP neural network is used in many applications. The proper neuron models will enhance the final neural network architecture designs and implementations.

4.3.1 Artificial Neuron Implementations

The artificial neuron model basically consists of two stages. The first stage is used for multiplication and addition of parallel inputs and weights and the second is the nonlinear and linear squashing function to limit the output signal at particular value. The structure of the processing stage is massively dependent on the degree of the parallel computation in feed forward stage, backward stage and weight update stage. Basically three types of processing technique are used for the design of the processing stage. These techniques are given below:

1**. Serial Processing (SP):** In the serial processing multiplication is followed by accumulation process. This is a very common operation in many digital systems. Multiply and accumulator structure is illustrated in figure 4.2. Multiply and accumulator structure has two inputs. The multiplications of the two inputs are added

to the previously accumulated value. This process will be continued until all the input pairs are completed. The final sum is passed through the squashing function to produce the neuron output.

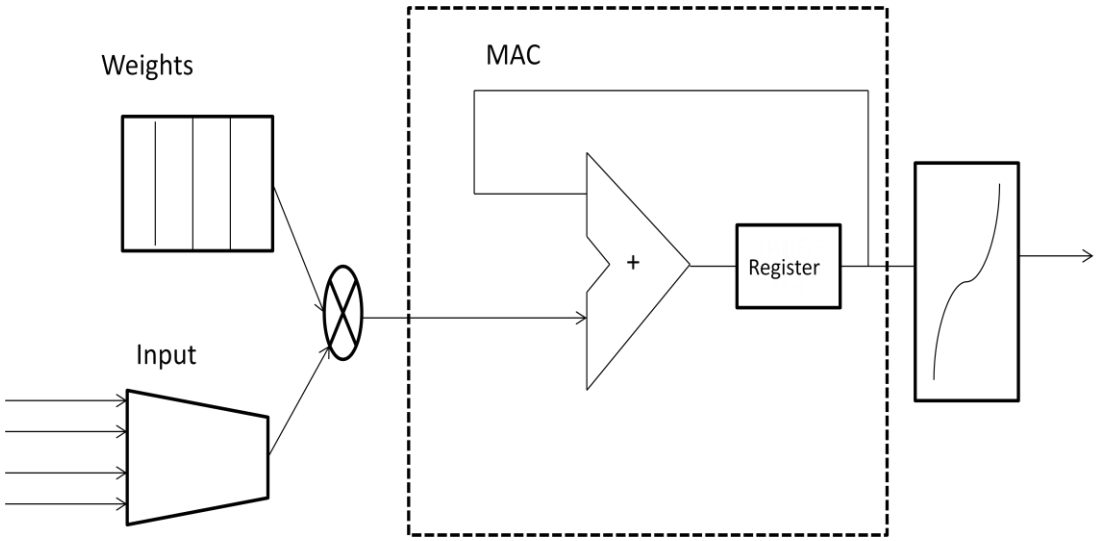


Figure 4.2 Neuron Serial Processing Computing model

2. Partial parallel processing (PPP): Partial parallel processing has a two stage. The first stage usage a number of multipliers and second stage is the adder tree. There is a one multiplier for each input/weight pairs. The result of first two multipliers is added together and added to the result of other multiplier. This process continues until all multiplication results are accumulated. The resulted sum is passed through the squashing function.

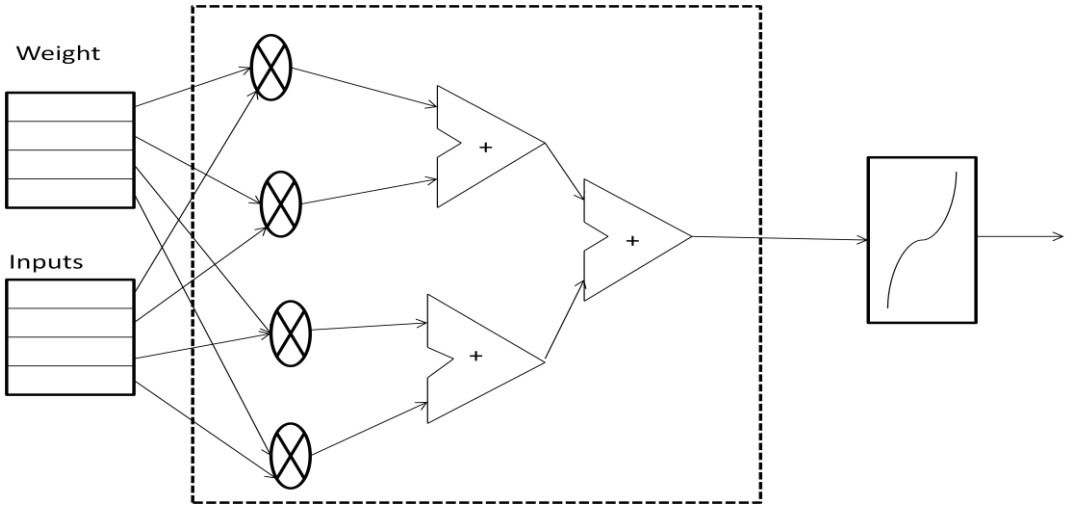


Figure 4.3 Neuron Partial Parallel Processing computing model

3. Full parallel processing (FPP): Partial parallel processing and full parallel processing are very much similar to each other. There is a one difference between these techniques. The former technique has a parallel array of multipliers. In addition, of the parallel array of multipliers FPP also have a one multi input parallel adder module that performs the calculation in one step. The final sum is passed through the squashing function to produce's the neuron's output.

Every processing technique has an own advantage and disadvantage in terms of the speed and hardware area requirement. Comparisons between these three techniques are given in table 2. The above table shows that serial processing is required less number of hardware compare to partial and full parallel processing, but time required for the computation is very high.

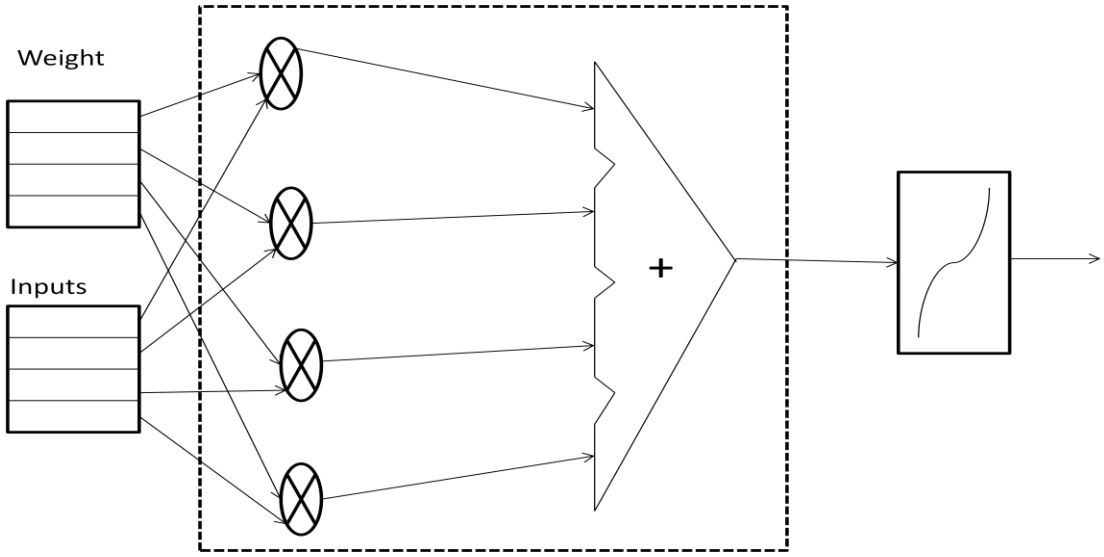


Figure 4.4: Neuron Full Parallel processing computing Model

Table 1 Comparison of Neuron computing methods

PROCESSING TECHNIQUE	AREA	TIME
Serial processing	Minimum	Highest
Partial parallel processing	Medium	Medium
Full parallel processing	Highest	Minimum

4.4 Data Representation

Fixed point (FXP) arithmetic and floating point (FLP) is used for encoding the neural network parameters and performing the computations. HDL language supports binary, integers for synthesis while real number can be used for simulation purpose only, they are not synthesizable. These previous mentioned two number system are used for representing real numbers so for digital signal processing.

1. Fixed- point (FXP) Format: Fixed-point format is used for representing real numbers. It is used for a number that has a fixed number of digits after (and sometimes also before) the radix point. FXP format is illustrated in fig. 4.5.

There are two parts in an FXP number. The first is the integer part; the second is the fractional part. FXP can be signed or unsigned. In the signed fixed-point format, the first bit of the integer part represents the sign bit. Fixed point representation is signed 2's complement binary representation, when the base of the FXP number is 2. The dual FXP format is encoded similarly to an FXP number with one additional "exponent" bit representing the position of the radix point. The dual fixed point format is illustrated in the figure 4.6. The FXP architecture is always smaller in area, as compared to FLP architecture with similar precision. The FXP is also faster than its FLP counterpart. FLP has an advantage that it can support a much wider range of values for same number of bits when compared to the FXP format.

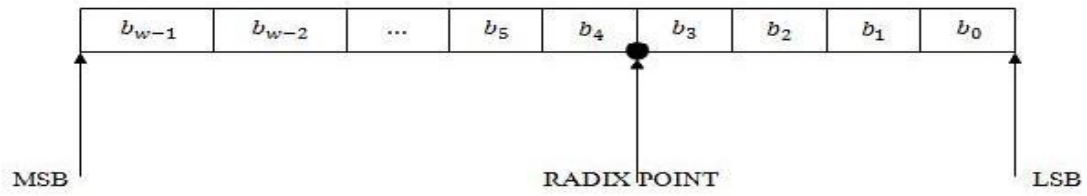


Figure 4.5 Fixed point format

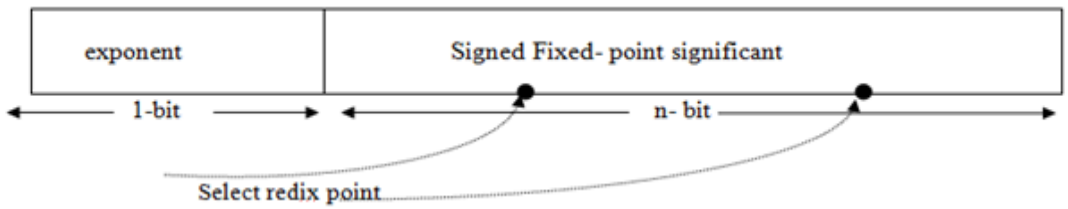


Figure 4.6 Format of a dual FXP number [30]

2. Floating-point Format: In general, while using FLP represented approximately to a fixed number of significant digits and scaled using an exponent. The base for the scaling is normally 2, 10 or 16. The typical number that can be represented exactly is of the form:

$$\pm d.dd\ldots d \times \beta^e \tag{4.1}$$

More precisely $\pm d_0 d_1 \ldots d_{p-2} d_{p-1} \times \beta^e$ represents the number

$$\pm(d_0 + d_1 \times \beta^{-1} + \cdots + d_{p-1} \times \beta^{-(p-1)}) \times \beta^e \tag{4.2}$$

where β represents the base (which is always assumed to be even) represents the exponent, and is the precision expressed as number of significant digits or bits for β = 2. One of the most common FLP is the single precision IEEE 754-1985 format shown in fig. 4.7**.**

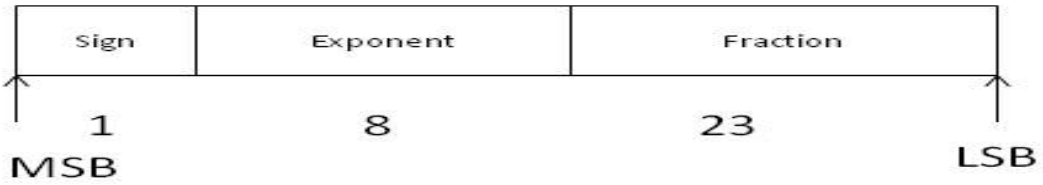


Figure 4.7 IEEE standard 754-1985 format for single precision [30]

4.5 Squashing Function

The activation function is also called squashing function, denoted by φ (v). The output of the neuron is defined in terms of the induced local field v. Three types of the activation function is used to implement a neuron.

a). Threshold function: The threshold function is also called Heaviside function such a threshold function is also referred as McCulloch-Pitts model. In this model if the induced local field of the neuron is nonnegative then the output is 1 otherwise 0. Threshold function can be described in the following form:

$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v \leq 0 \end{cases} \tag{4.3}$$

if the activation function range is -1 to 1 then the threshold function is given by the following equation

$$\varphi(v) = \begin{cases} +1 & \text{if } v > 0 \\ 0 & \text{if } v = 0 \\ -1 & \text{if } v < 0 \end{cases} \tag{4.4}$$

The above equation is commonly referred to as the Signum function.

b). Piecewise-linear function: The following equation is used to describe the piecewise- linear function

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq +\frac{1}{2} \\ v & \text{if } -\frac{1}{2} > v > +\frac{1}{2} \\ 0 & \text{if } v \leq -\frac{1}{2} \end{cases} \quad (4.5)$$

The function amplification factor in the linear region of operation is assumed to be unity**.** This form of an activation function may be viewed as an approximation to a non-linear amplifier. The following two situations are the special form of the piecewise-linear function.

1. The piecewise-linear function reduces to a threshold function if the amplification factor of the linear region is made infinitely large.
2. A linear combiner arises if the linear region of operation is maintained without running into the saturation.

c) Sigmoid activation function: The sigmoid function is the most common type of the activation function. It is used in the construction of the artificial neural network. The activation function looks like S-shaped function. It is strictly increasing function and also has linear and nonlinear behavior. An example of the sigmoid function is logistics function that is defined by:

$$\varphi(\mathcal{V}) = \frac{1}{1+e^{(-av)}} \quad (4.6)$$

$$\varphi'(\mathcal{V}) = \varphi(\mathcal{V})\left(1 - \varphi(\mathcal{V})\right) \quad (4.7)$$

Where a is the slope parameter of the sigmoid function. Sigmoid functions of different slopes can be obtained by changing the parameter a. sigmoid function behaves like a threshold function if slope parameter approaches infinity. This type of sigmoidal function is differentiable function because it has continuing value from 0 to 1. Differentiality is an important feature of neural network theory. The Hyperbolic tangent function is the form of sigmodal function. The value of the sigmoidal function has the range of -1 to +1. Hyperbolic function is defined by $\varphi(v) = \tanh(v)$.

4.6 Implementation of the Sigmoidal Activation Function

The implementation of the sigmoidal function in Fpga is a challenging task. Due to the present of the exponential function direct implementation is not suitable for sigmoid function. Three computationally simplified approximation approaches is used to implement sigmoid function.

1) Direct Approximation (DA): In direct approximation second order nonlinear function uses an approximation of the sigmoidal function. This function can be implemented using digital technique. The following equation is used for the direct approximation of the sigmoidal function:

$$\Theta(v) = \begin{cases} 1 & for\ \ w \le v \\ v(L - mv) & for\ 0 \le v \le w \\ v(L + mv) & for - w \le v \le 0 \\ -1 & for\ w \le -v \end{cases} \qquad (4.8)$$

Where L and m represent the slope and gain of the function $\Theta(v)$ between the saturation region –w to +w. Fig. 4.8 shows the block diagram of the approximated sigmoidal function.

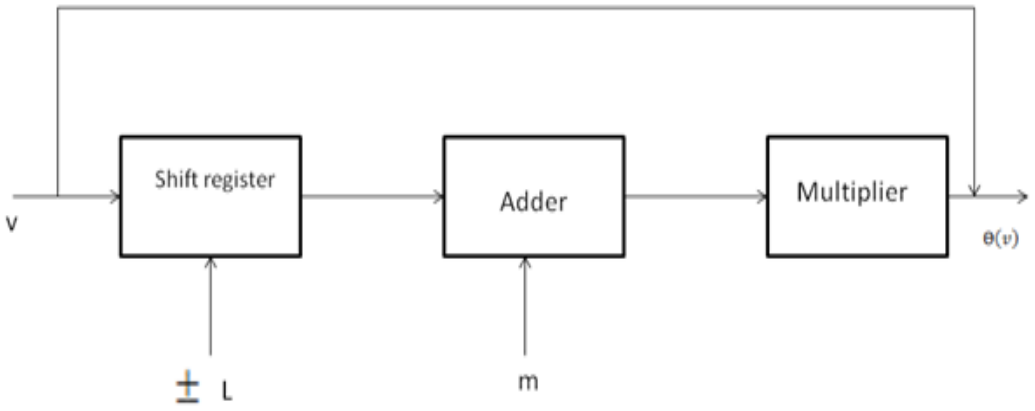


Figure 4.8 Block diagram of the sigmoid DA implementation [33]

2) **Look Up table (LUT) approximation:** In the lookup table approximation technique pre-calculated output of the sigmoid function is stored in the lookup table. Both lookup table and direct method is suffered from quantization error. The large lookup table is slow and costly. The large lookup table is requires lots of hardware, so it is not suitable for hardware implementation. The time required for the access of the content is very high in the lookup table so Power consumption is also very high.

3) **Piece-Wise Linear Approximation (PWL):** The implementation of a high-precision squashing function requires large area, but FPGAs have limited area. Thus, for implementing the squashing function in FPGA, trade-off between both the parameters should be known so the squashing function must be implemented either using PWL (Piece-Wise Linear) or LUT (Look up table). PWL should be chosen because there LUT have a high precision loss. Thus, it is undesirable implementing a LUT based squashing function in FPGA, since FPGAs have limited internal memory which has other purposes also then serving only as storage for the squashing function. Also sharing an LUT approximation for squashing function among all the neurons reduces speed. Piecewise linear (PWL) approximation is achieved by approximating the sigmoid function using a piecewise linear approximation (PWL). The mathematical representation of sigmoidal function is expressed by the following equation:

$$\Theta(v) = \begin{cases} 1 & for \;\; v \geq -4 \\ 0.0625 + 0.75 & for\; 0 < v < 4 \\ 0.5 & for \;\; v = 0 \\ 0.0625 + 0.25 & for \;\; 0 < v < -4 \\ 0 & for\; v \leq -4 \end{cases} \tag{4.9}$$

4.7 General Structure of the Artificial Neural Network

The figure 4.9 shows the complete block diagram and interconnection between these blocks. Interconnection of these blocks is used for controlling the data processing operation. The layout consists of four major blocks: the forward stage, Backpropagation stage, weight update stage and the controller.

1**) Forward stage**: The forward stage block consists of mainly two layers hidden and output layers. This processing unit determines the computational ability of the neural network. The neurons are arranged into a well structure topology in these two layers. The neurons in the each layer communicate with the neurons of successive layer. All processing elements in one layer can send signals parallel to or from the processing unit of next layer. The input layer neurons pass the input signal to the hidden layer. The neurons in the hidden and output layer perform the operation. Each neuron in the hidden and output layers calculates its output by the product of input and weight of each connection. The final output of each neuron is based on the activation function.

2) Backpropagation stage: The backpropagation stage calculates the local gradient term for the hidden and the output layers. The local gradient term is calculated after calculating error term. The error is calculated by subtracting the final output and the desired output. After calculation of the error, gradient term is calculated for each output neuron. This local gradient term is back propagated to the hidden layer, based on the output local gradient term and the associated weights in the output layer determine the delta for hidden layer. Further local gradient term is calculated for the input layers also based on the hidden deltas and associated weights with the hidden layers.

3) Update stage: The update stage adjusts the network's weights according to the local gradient term, the learning rate parameter and the input to the corresponding layer. The weight update term is added to the existing weight to produce a new weight for the next cycle of the forward stage.

4) Controller unit: The controller unit is used for data routing and controlling the timing during the operation of the previous three stages. The controller has a signal for each stage. The rising edge of each signal corresponds with the start of processing for the corresponding stage. The Finite state machine is used for the controlling the operation. The timing of the dedicated signals is determined by the processing time of each stage. Every sub module of the three stages has the information on the number of cycles needed for data to propagate through that sub module.

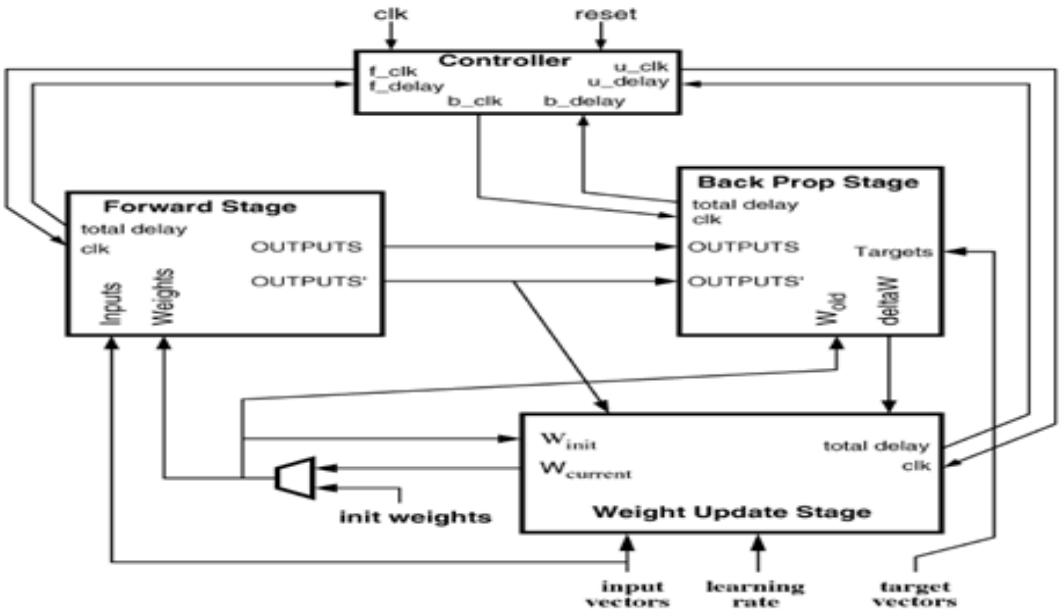


Figure 4.9 Block diagram of the error correction learning [30]

CHAPTER 5

IMPLEMENTATION OF FULL ARTIFICIAL NEURAL NETWORK IN FPGA

In the present work 4:3:3 neural network is used for the implantation of the neural network. The network has four neurons in the input layer, three neurons in the hidden layer and three neurons in the output layer. This architecture is optimized for the classification of the iris data using backpropagation algorithm implemented in the Matlab program. In this algorithm weights of the layer are updated by back propagating the error. The fixed point number system is used for encoding the neural network parameters and performing the computations. In this work weight, input and error signal uses 8-bit signed fixed numeric representation are used to full fill the requirement of the real number (which is not synthesizable for hardware implementation). MSB acts as sign-bit, three bits for integer and remaining four bits work as fractional bits. Symmetric saturating linear function has been used for the implementation of the activation function. Three satlins activation function is used for each hidden and output layer. The symmetric saturating linear activation function is given by:

$$Y = \begin{cases} 1 & if\ v > 1 \\ v & if\ -1 \leq v \leq 1 \\ -1 & if\ v < -1 \end{cases} \tag{6.1}$$

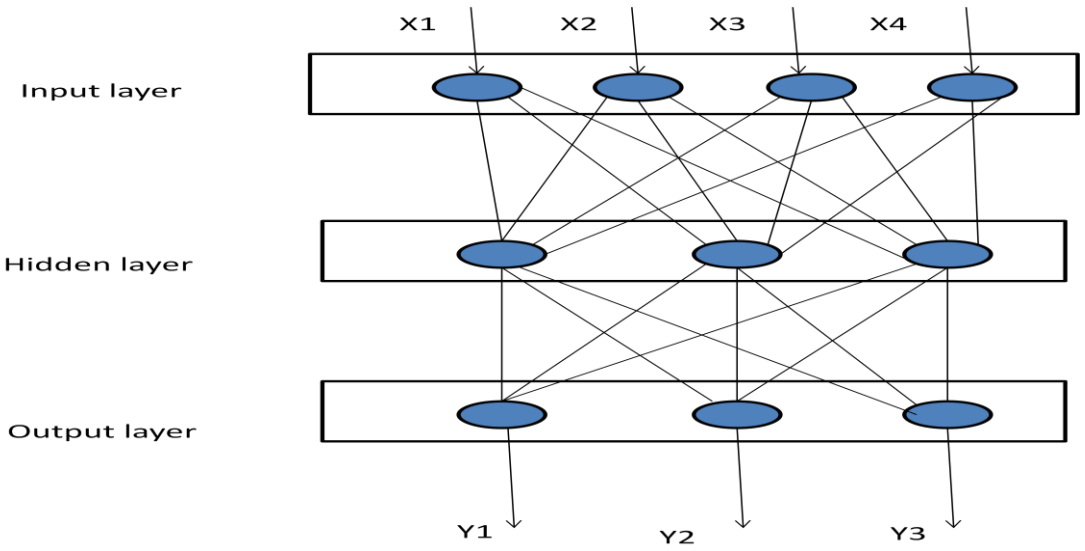


Figure 5.1 Structure of the 4:3:3 network

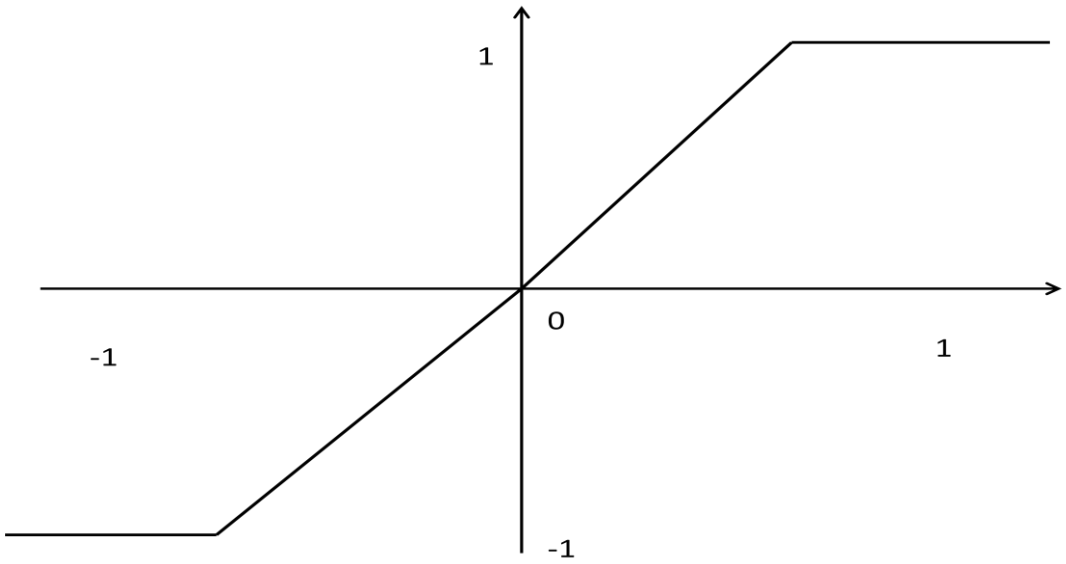


Figure 5.2 symmetric saturating linear activation function

Iris data has a four attribute and three classes. For classification of the iris data is required four inputs and three outputs. Numbers of optimum neurons in the hidden layer are determined by hit and trial method. In this work one hidden layer and three neurons in hidden layer has been used. Here, a, b, c, d is the inputs and the final outputs are o1, o2 and o3 outputs are evaluated in the following steps:

$$o_{h1} = (a \times w_{1a} + b \times w_{1b} + c \times w_{1c} + d \times w_{1d}) \quad (6.2)$$

$$o_{h2} = (a \times w_{2a} + b \times w_{2b} + c \times w_{2c} + d \times w_{2d}) \quad (6.3)$$

$$o_{h3} = (a \times w_{3a} + b \times w_{3b} + c \times w_{3c} + d \quad (6.4)$$

$$o_1 = (o_{h1} \times w_{01} + o_{h2} \times w_{02} + o_{h3} \times w_{03} \quad (6.5)$$

$$o_2 = (o_{h1} \times w_{11} + o_{h2} \times w_{12} + o_{h3} \times w_{13}) \quad (6.6)$$

$$o_3 = (o_{h1} \times w_{21} + o_{h2} \times w_{22} + o_{h3} \times w_{23}) \quad (6.7)$$

Where o_{h1}, o_{h2}, o_{h3} are the outputs of the hidden neurons respectively. The error in the final output is thus calculated as:

$$e_{01} = d_1 - o_1 \quad (6.8)$$

$$e_{02} = d_2 - o_2 \quad (6.9)$$

$$e_{03} = d_3 - o_3 \quad (6.10)$$

There d_1, d_2, d_3 are the desired responses of the network. The local gradient term at the output node is given by

$$\delta_{01} = e_{01} \times o_1' \tag{6.11}$$

$$\delta_{02} = e_{02} \times o_2' \tag{6.12}$$

$$\delta_{03} = e_{03} \times o_3' \tag{6.13}$$

There o_1', o_2', o_3' are the differential of the output. The local gradient term at the hidden node can be calculated as

$$\delta_{h1} = \left(\sum(\delta_{01} \times w_{11}) + (\delta_{02} \times w_{21}) \times (\delta_{03} \times w_{31})\right) \times o_{h1}' \tag{6.14}$$

$$\delta_{h2} = \left(\sum(\delta_{01} \times w_{12}) + (\delta_{02} \times w_{22}) \times (\delta_{03} \times w_{32})\right) \times o_{h2}' \tag{6.15}$$

$$\delta_{h3} = \left(\sum(\delta_{01} \times w_{13}) + (\delta_{03} \times w_{23}) \times (\delta_{03} \times w_{33})\right) \times o_{h3}' \tag{6.16}$$

Adjustment of the weight for the output layer is given by:

$$\Delta w_{01} = \eta \times o_1 \times \delta_{01} \tag{6.17}$$

$$\Delta w_{02} = \eta \times o_1 \times \delta_{02} \tag{6.18}$$

$$\Delta w_{03} = \eta \times o_1 \times \delta_{03} \tag{6.19}$$

$$\Delta w_{11} = \eta \times o_2 \times \delta_{01} \tag{6.20}$$

$$\Delta w_{12} = \eta \times o_2 \times \delta_{02} \tag{6.21}$$

$$\Delta w_{13} = \eta \times o_2 \times \delta_{03} \tag{6.22}$$

$$\Delta w_{21} = \eta \times o_3 \times \delta_{01} \tag{6.23}$$

$$\Delta w_{22} = \eta \times o_3 \times \delta_{02} \tag{6.24}$$

$$\Delta w_{23} = \eta \times o_3 \times \delta_{03} \tag{6.25}$$

Adjustment of the weight for hidden layer is given by the following equation:

$$\Delta w_{1a} = \eta \times x_1 \times \delta_{h1} \tag{6.26}$$

$$\Delta w_{1b} = \eta \times x_2 \times \delta_{h1} \tag{6.27}$$

$$\Delta w_{1c} = \eta \times x_3 \times \delta_{h1} \tag{6.28}$$

$$\Delta w_{1d} = \eta \times x_4 \times \delta_{h1} \quad (6.29)$$

$$\Delta w_{2a} = \eta \times x_1 \times \delta_{h2} \quad (6.30)$$

$$\Delta w_{2b} = \eta \times x_2 \times \delta_{h2} \quad (6.31)$$

$$\Delta w_{2c} = \eta \times x_3 \times \delta_{h2} \quad (6.32)$$

$$\Delta w_{2d} = \eta \times x_4 \times \delta_{h2} \quad (6.33)$$

$$\Delta w_{3a} = \eta \times x_1 \times \delta_{h3} \quad (6.34)$$

$$\Delta w_{3b} = \eta \times x_2 \times \delta_{h3} \quad (6.35)$$

$$\Delta w_{3c} = \eta \times x_3 \times \delta_{h3} \quad (6.36)$$

$$\Delta w_{3d} = \eta \times x_4 \times \delta_{h3} \quad (6.37)$$

The adjustments evaluated above are added to the original weights in the next stage (weight updating stage) the updated weights are given by the following equation:

$$w_{1a} = w_{1a} + \Delta w_{1a} \quad (6.38)$$

$$w_{2a} = w_{2a} + \Delta w_{2a} \quad (6.39)$$

$$w_{3a} = w_{3a} + \Delta w_{3a} \quad (6.40)$$

$$w_{4a} = w_{4a} + \Delta w_{4a} \quad (6.41)$$

$$w_{1b} = w_{1b} + \Delta w_{1b} \quad (6.42)$$

$$w_{2b} = w_{2b} + \Delta w_{2b} \quad (6.43)$$

$$w_{3b} = w_{3b} + \Delta w_{3b} \quad (6.44)$$

$$w_{4b} = w_{4b} + \Delta w_{4b} \quad (6.45)$$

$$w_{1c} = w_{1c} + \Delta w_{1c} \quad (6.46)$$

$$w_{2c} = w_{2c} + \Delta w_{2c} \quad (6.47)$$

$$w_{3c} = w_{3c} + \Delta w_{3c} \quad (6.48)$$

$$w_{4c} = w_{4c} + \Delta w_{4c} \quad (6.49)$$

$$w_{1d} = w_{1d} + \Delta w_{1d} \quad (6.50)$$

$$w_{2d} = w_{2d} + \Delta w_{2d} \quad (6.51)$$

$$w_{3d} = w_{3d} + \Delta w_{3d} \quad (6.52)$$

$$w_{4d} = w_{4d} + \Delta w_{4d} \quad (6.53)$$

$$w_{1c} = w_{1c} + \Delta w_{1c} \quad (6.54)$$

$$w_{2c} = w_{2c} + \Delta w_{2c} \quad (6.55)$$

$$w_{3c} = w_{3c} + \Delta w_{3c} \quad (6.56)$$

$$w_{01} = w_{01} + \Delta w_{01} \quad (6.57)$$

$$w_{02} = w_{02} + \Delta w_{02} \quad (6.58)$$

$$w_{03} = w_{03} + \Delta w_{03} \quad (6.59)$$

$$w_{11} = w_{11} + \Delta w_{11} \quad (6.60)$$

$$w_{12} = w_{12} + \Delta w_{12} \quad (6.61)$$

$$w_{13} = w_{13} + \Delta w_{13} \quad (6.62)$$

$$w_{21} = w_{21} + \Delta w_{21} \quad (6.63)$$

$$w_{22} = w_{22} + \Delta w_{22} \quad (6.64)$$

$$w_{23} = w_{23} + \Delta w_{23} \quad (6.65)$$

It is evident from equation (6.2) to (6.65) that computational complexity of backpropagation algorithm is highly dependent on the number of synaptic weights and number of neurons in the hidden layer. A significant contributor to this complexity is the multiplication operation which is required for the calculation of local gradient as well as for the weight update operation. To reduce the number of multipliers resulting from the implementation of the above equations a MUX based architecture is being proposed to process both the input samples in the feed forward and error terms in the back propagation loops respectively. Multiplication of the learning rate parameter and output of the each layer is calculated during the training process and the calculated terms are stored in registers. The outputs from the previous layer neurons are entered in next layer neuron serially so multiplication of all these two terms can be achieved by using one multiplier. As shown in Figs. 5.3 (a) and 5.3

(b), local gradient of each layer (denoted by subscript *h* and *o* for hidden and output layers respectively) is calculated by combining of serial processing and multiplexer based approach. Complete weight updates term is calculated by multiplication of local gradient term with and previous layer neuron output. In the proposed structure, twelve weight updates terms are calculated for hidden layer of 4-3-3 network and nine weight updates terms are calculated for output layer of 4-3-3 neural network. Only one multiplier is required for each layer. The calculation of the weight updates terms for hidden and output layer neurons are shown in Fig. 5.3(a) and 5.3 (b). The figure 5.5 shown below uses sequential mode of learning to train the neural network. Sequential mode of backpropagation learning is also referred as online or stochastic mode. In this work multiply and accumulate structure is chosen for neurons. In this structure one multiplier and one accumulator per neuron is used. The neuron architecture has been shown in figure 4.1**.** Weights are stored in RAM and each neuron has its own RAM to store weight.

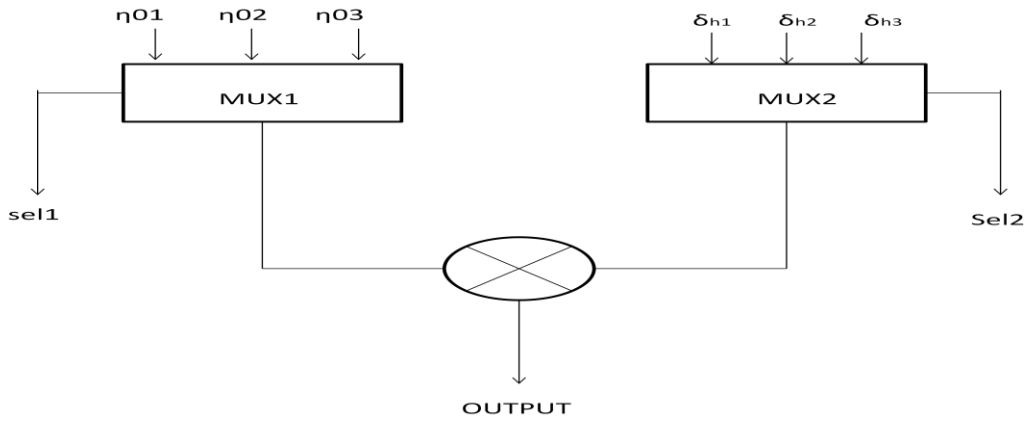


Figure 5.3(a) calculation of the weight updates term in the hidden layer

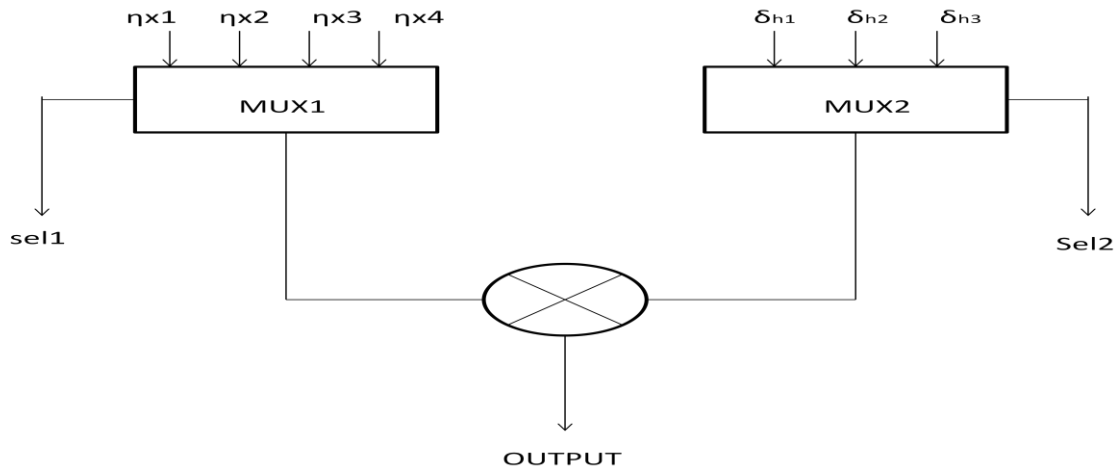


Figure 5.3(b) calculation of the output weight updates term

Multiplied values are summed in an accumulator. The processes are synchronized to clock signal. In the design all neurons in each layer are connected to the one input, but a previous layer may have several outputs. These outputs are first applied on MUX and using proper controlling operation each input from the previous layer is applied successively at each clock cycle. All of the neurons used in each layer operate parallel. They take an input from their common input line, multiply it with the corresponding weight that are stored in corresponding RAM and product are accumulated on the accumulator. In the proposed work layer has three neurons, next layer takes and processes these inputs in three clock cycles. At each clock cycle each input is applied at every neuron in the layer after processing all the inputs the layer starts to transfer these values to its output simultaneously and same previous scheme is used for the next layer to take them successively by enabling corresponding neuron's three-state output. The layer architecture is shown in fig 5.4.In this work different learning rate parameter can be chosen for different layer. The equation (6.17) – (6.36) shows that the local gradient term for each layer is different. A local gradient term for the hidden layer is less compare to the output layer. The reason behind this is that small portion of the error signal is propagated to the hidden layers. The convergence of the network is highly dependent on the weights of the hidden layer. The larger learning rate will be helped to obtain optimum weight of the hidden layer. The error signal is high for output layer so learning rate parameter should be less compare to a hidden layer. Where η_2 is grater then the η_1.

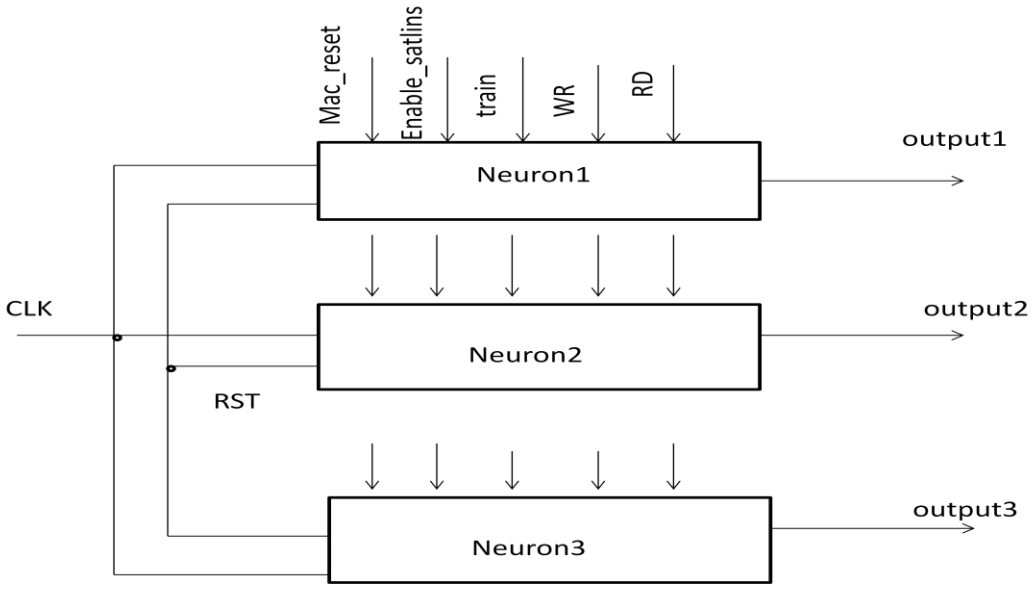


Figure 5.4 Hidden and output layer architecture

The Modified weight update equation will be help to improve the convergence rate and reduced structure complexity

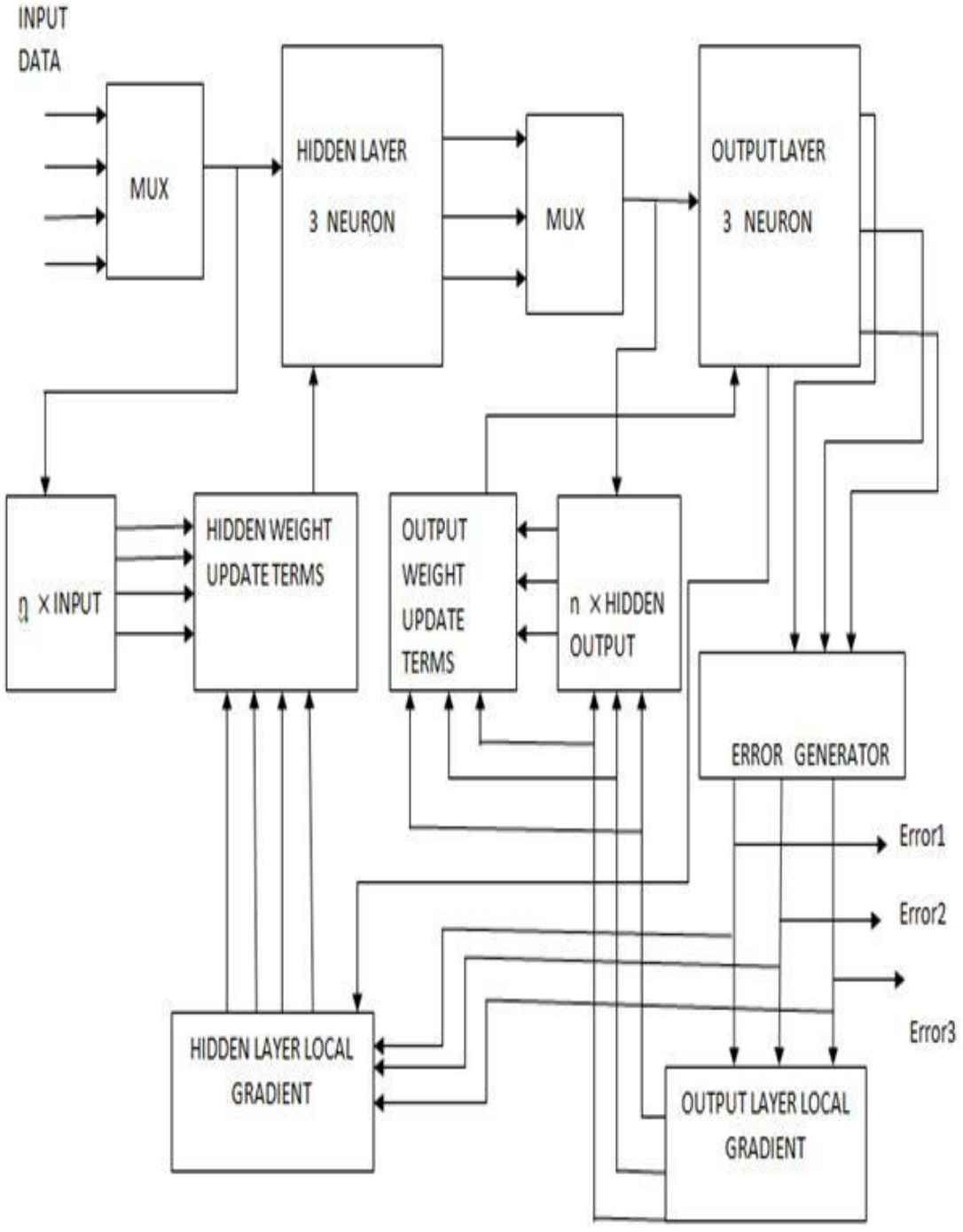


Figure 5.5 The proposed ANN architecture with multiplexer based weight updating

CHAPTER 6

RESULT & DISCUSSION

The results of the present work are divided into two sections. A first section describes MATLAB simulation results while the second section described the functional simulation results, FPGA synthesis results and ASIC synthesis results.

6.1 Matlab Simulation Results

In this dissertation work two 4:5:3 architectures have been used for the classification purpose. These two architectures are used different mechanism for learning rate parameter variation. The first architecture uses fixed learning rate parameter and second architecture is used PCA based learning rate variation. The performance comparison of these two architectures with randomly initialized weights reported in this section. All the simulation has been carried out on a computer having Intel core 2 Duo CPU with clock speed 2.00 GHz and 2 GB of RAM. The MATLAB version R2012a has been used for the simulation.

Implementation Details:

- The Iris dataset download from UCI machine learning repository. The Iris data set has 150 instances, 4 attributes and 3 classes. Every class has 50 instances. Out of these 150 instances randomly chosen instances were used for training data and testing data by the learning algorithm.
- Simulations were carried out for a maximum of 5000 epochs for fixed step-size ANN and variable learning rate ANN. TRAINGDM algorithm was used for train the network.

As a first step, the box-whisker diagram shown in Fig.6.1 for the architecture that use variable learning rate mechanism. The Box-whisker diagram shows the variation in the error performance at different value of the learning rate parameter. This parameter was initialized with initial value 0.01. The lowest average MSE has been obtained at 0.020695.Fig 6.2 shows the comparison of the MSE for the two mechanisms. The MSE has been obtained approximately same for these two mechanisms. The same architecture has been used to train the neural network at 0.020695 step-sizes. Confusion plots for both mechanisms have been shown in Fig 6.3 and Fig.6.4. The

percentage classification results for the variable learning rate ANN is high compared to fixed learning rate ANN. Furthermore, for the hardware implementation of the neural network, comparison graph was plotted between MSE and the number of neurons with five values of the learning rate parameter (0.1, 0.15, 0.20, 0.25 and 0.30). The optimum number of hidden neurons and learning rate parameter is obtained from the simulation. The graph is shown in figure 6.5 shows that the mean square error was obtained minimum when the learning rate parameter and the number of neurons in the hidden layer is 0.15 and 3 respectively. The minimum mean square error in these parameters is 0.022.

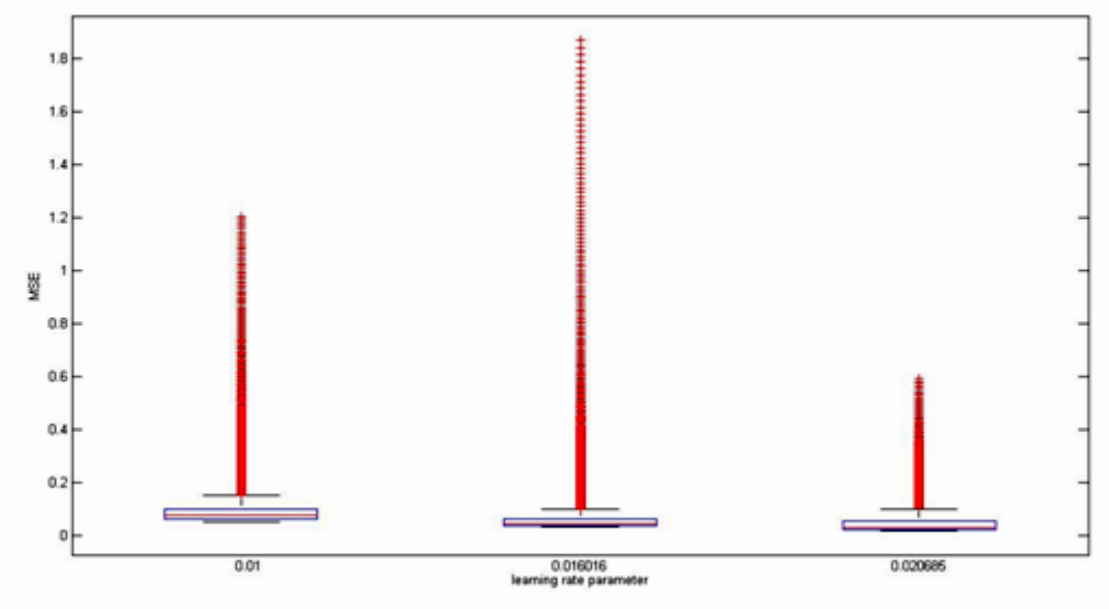


Figure 6.1 MSE vs. learning rate parameter

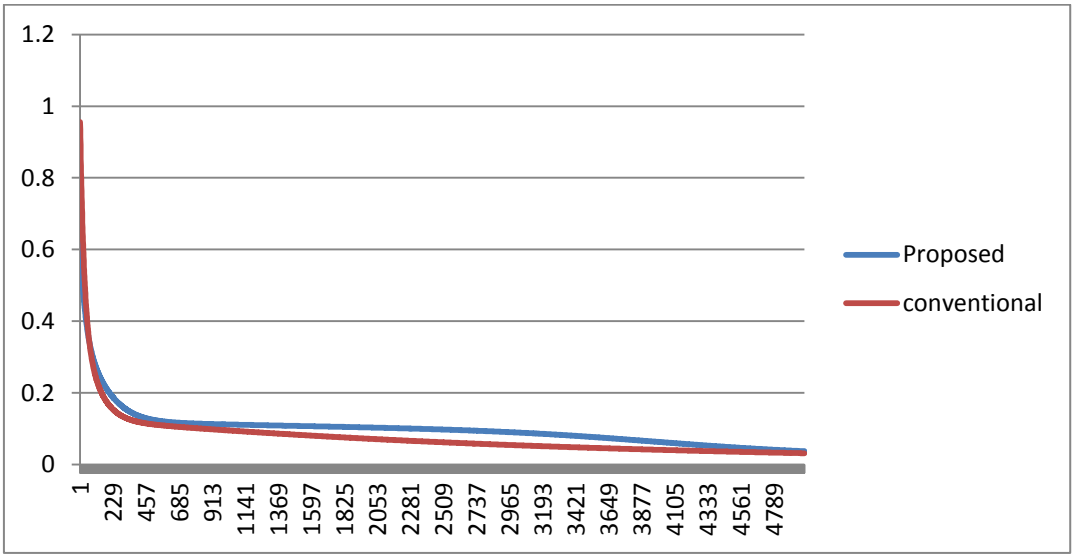


Figure 6.2 Comparison of the MSE

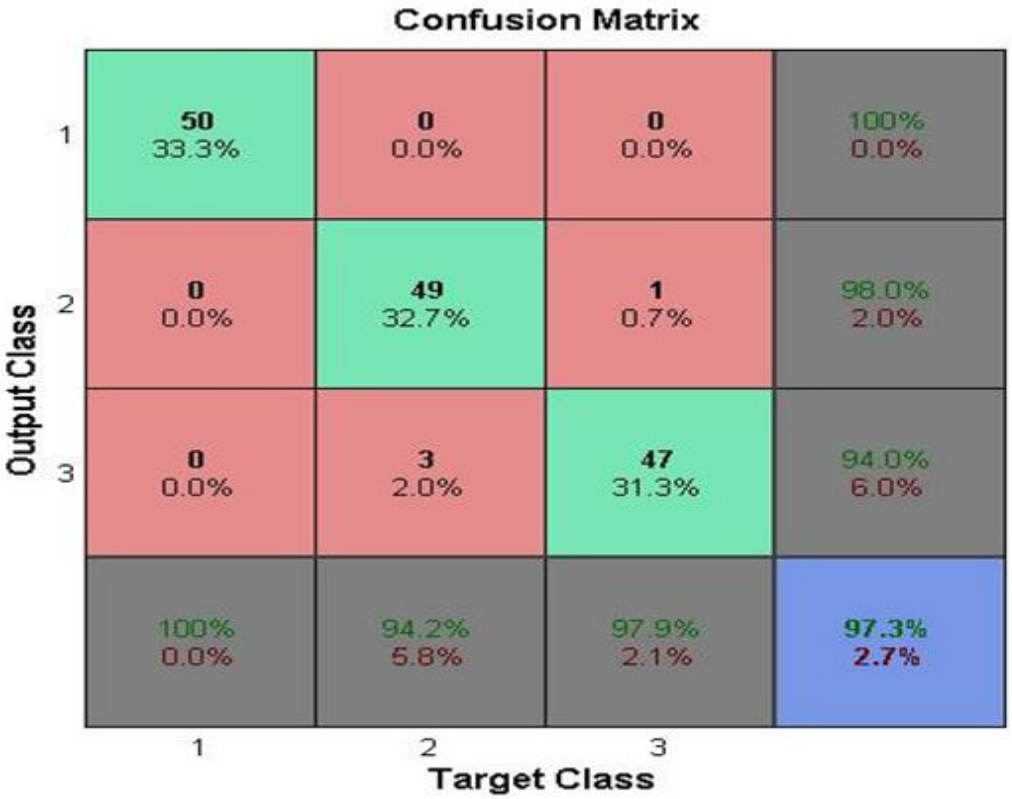


Figure 6.3 Confusion Plot for Variable learning rate ANN

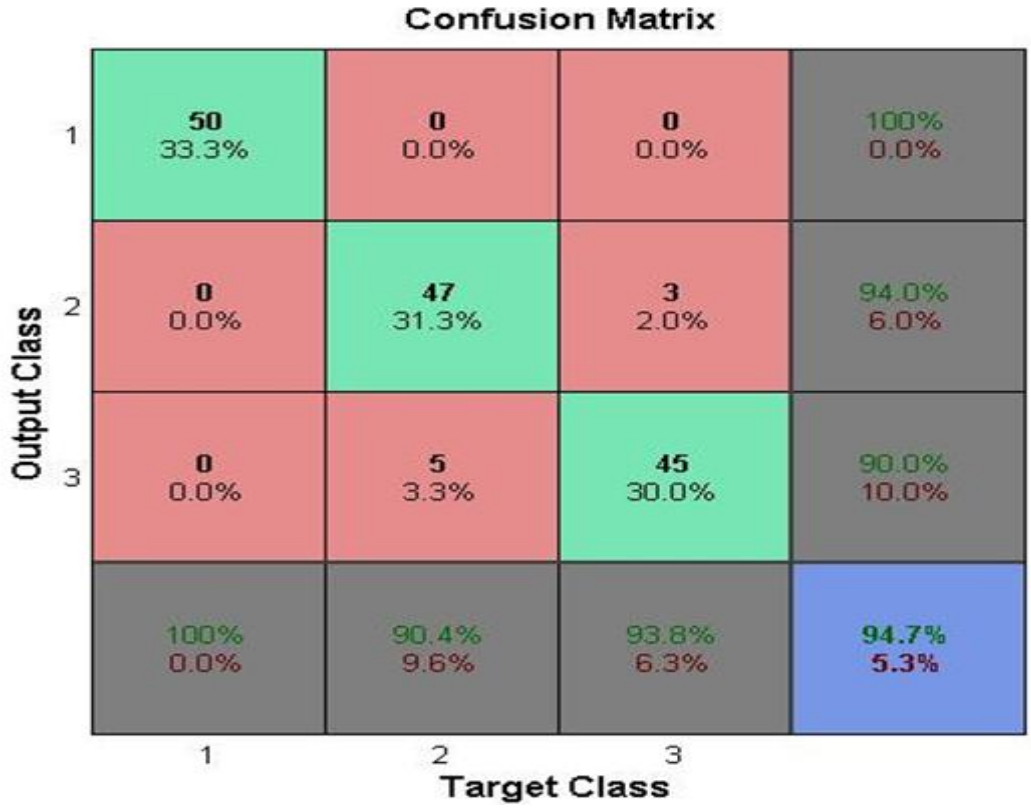


Figure 6.4 Confusion plot for fixed learning rate ANN

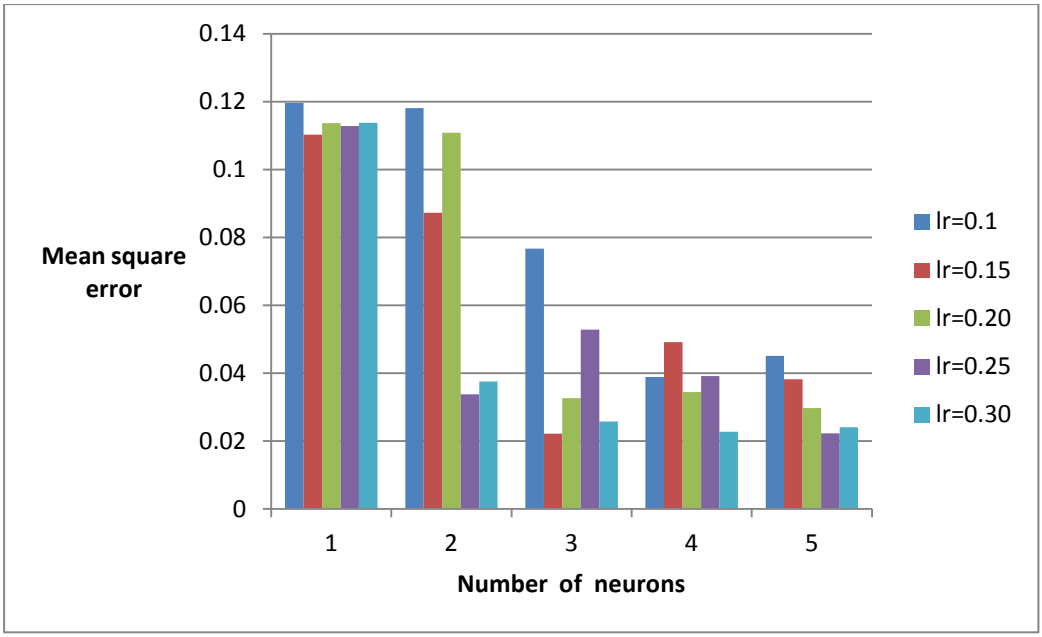


Figure 6.5 Number of neurons vs. mean square error

6.2 FPGA Synthesis Result

After the successful simulation of the designed neural network on the neural network toolbox the VHDL code of the designed neural network system was written. After that the designed vhdl code was simulated and verified using ModelSim PE Student Edition 10.3. The results of which are shown in Fig.6.6. Next, it was synthesized using Xilinx ISE 14.5 so that it can be implemented on the FPGA. This design was implemented on the Spartan 3E Fpga. Total number of design, building blocks required after HDL synthesis is shown in Fig.6.7.

The table 6.1 shows the device utilization summary after post-MAP. The MAP program maps a logic design to a Xilinx FPGA.MAP maps the design logic to the components of the target Xilinx FPGA. The output from the map file is the Native circuit Description file (NCD). The mapped NCD file can then be placed and routed using the place and route program.

Timing report after synthesizing and after place and route are shown in table 6.2, 6.3. Actual operating frequency is obtained after place and route. The results show that operating frequency was reduced after place and route.PAR program accepts the mapped NCD files as input and place and route the design and outputs a updated NCD to be used by the bit stream generator.

The power of the proposed design was calculated using Xpower estimator spreadsheet, the result of which is shown in table 6.4. XPE considers the design's resource usage, toggle rates, I/O loading and other factors to calculate the estimated power distribution. xpower analyzer can be used for more accurate estimates and power analysis. Fig. 6.7 shows the how much power was utilized from the different resources of the design.

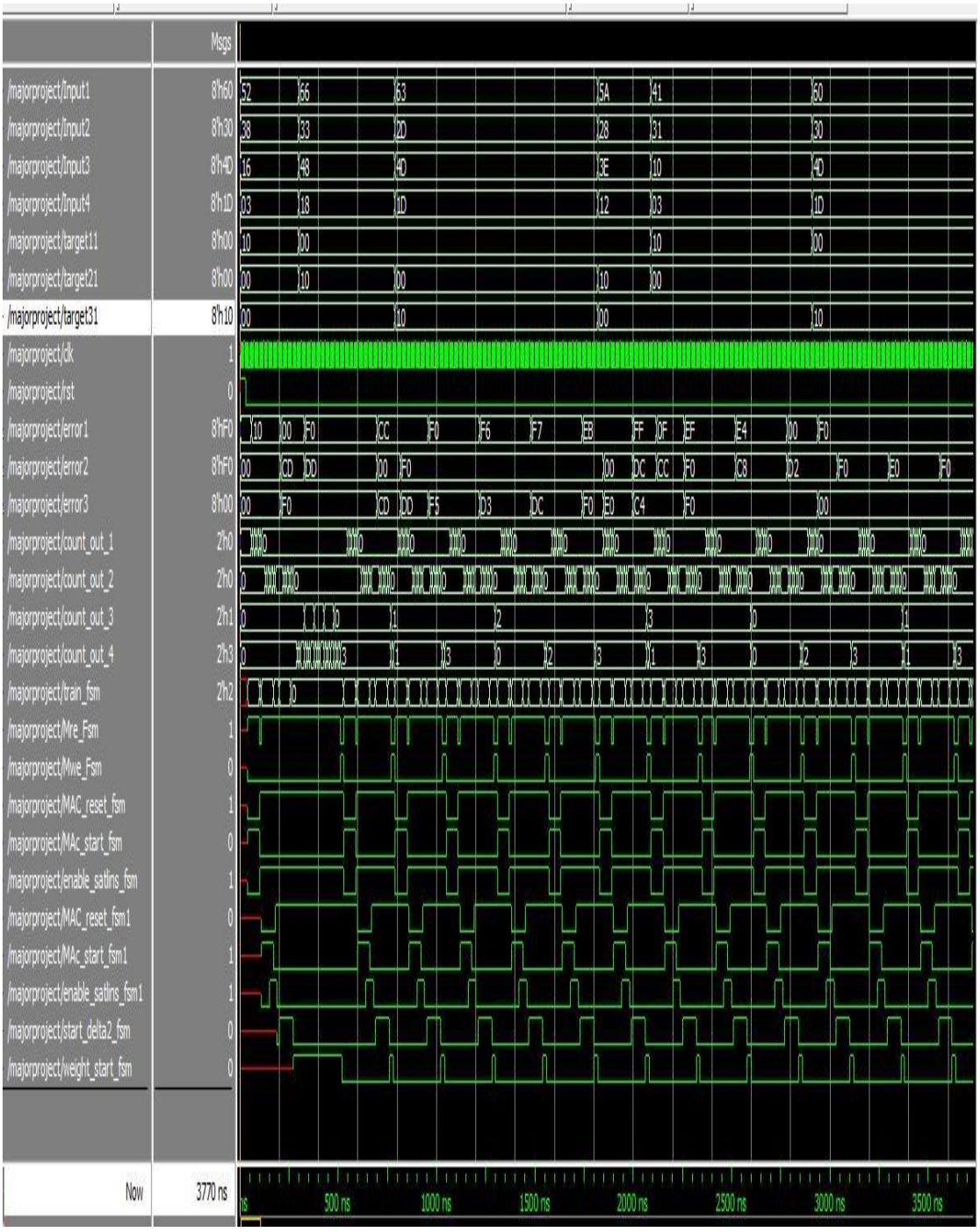


Figure 6.6 Simulation result of the 4:3:3 neural network

```
# Multipliers                                   : 13
 8x8-bit multiplier                            : 13
# Adders/Subtractors                            : 20
 16-bit adder                                  : 12
 2-bit adder                                   : 1
 3-bit adder                                   : 1
 7-bit adder                                   : 6
# Counters                                      : 4
 2-bit up counter                              : 2
 3-bit up counter                              : 1
 5-bit up counter                              : 1
# Accumulators                                  : 24
 8-bit up accumulator                          : 24
# Registers                                     : 122
 1-bit register                                : 33
 16-bit register                               : 36
 2-bit register                                : 6
 8-bit register                                : 47
# Comparators                                   : 14
 3-bit comparator greater                      : 1
 3-bit comparator lessequal                    : 1
 8-bit comparator greater                      : 6
 8-bit comparator less                         : 6
# Multiplexers                                  : 17
 2-bit 4-to-1 multiplexer                      : 2
 8-bit 4-to-1 multiplexer                      : 15
# Xors                                          : 24
 1-bit xor2                                    : 3
 1-bit xor3                                    : 21
```

Figure 6.7 HDL Synthesis Report

Table 2 Device utilization

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	939	9,312	10%
Number of 4 input LUTs	920	9,312	9%
Number of occupied Slices	709	4,656	15%
Number of Slices containing only related logic	709	709	100%
Number of Slices containing unrelated logic	0	709	0%
Total Number of 4 Input LUTs	920	9,312	9%
Number of bonded IOBs	82	232	35%
Number of BUFGMUXs	1	24	4%
Number of MULT 18X 18SIOs	13	20	65%
Average Fan-out of Non-Clock Nets	3.07		

Table 3 Timing Report after synthesize

Clk Freq. (MHZ)	Minimum Period (ns)	Minimum input arrival time before clock (ns)	Maximum output required time after clock (ns)
98.290	10.174	9.296	4.846

Table 4 Timing Report after place and route

Clk Freq. (MHZ)	Minimum Period (ns)	Minimum input arrival time before clock (ns)	Maximum output required time after clock (ns)
81.281	12.303	9.715	8.994

Table 5 Power Report

Static power (W)	Dynamic power(W)	Total Power (W)
0.079	0.083	0.162

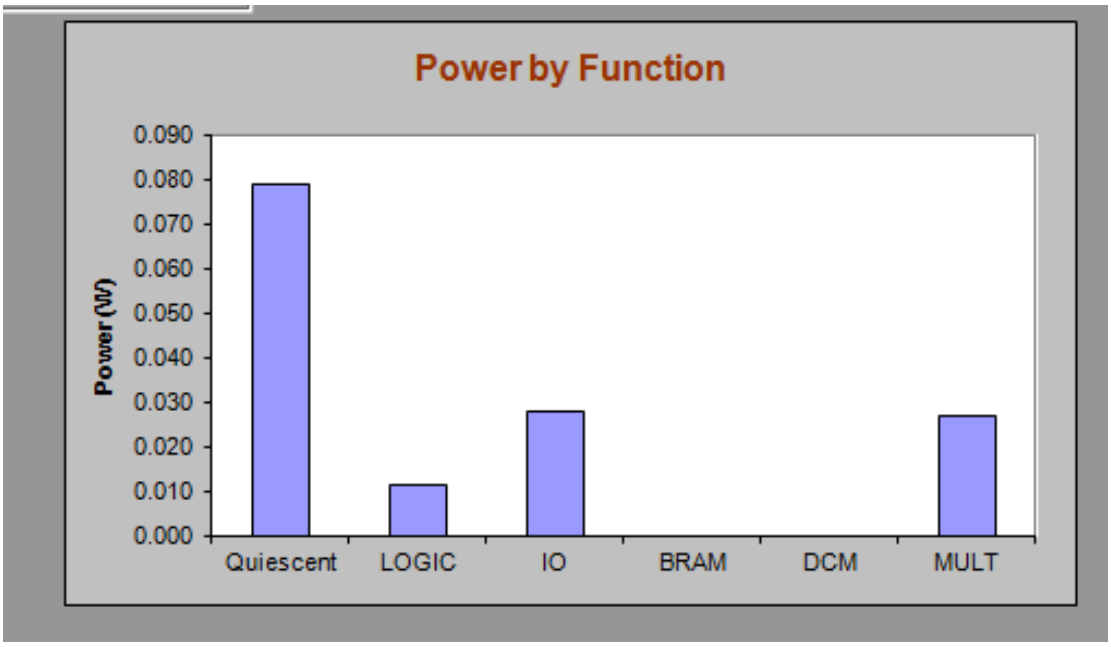


Figure 6.8 On-Chip Power by Function for Neural network in Spartan 3E

6.3 ASIC Synthesis Results

In this dissertation work Mentor tools have been used for ASIC implementation. Leonardo spectrum was used for synthesizing the HDL code to the netlist. Leonardo spectrum Level 3, 2013a.3 has been used to convert vhdl code to netlist (.v) file. This design has been synthesized at 250nm. After an optimization process number of cells used in the design was obtained. Table 6.6 shows the Leonardo spectrum synthesis report. Synthesize report shows the number of ports, nets and gates used in this design. After that Mentor design architect has been used to create the schematic of the design. This tool uses optimize netlist (.v) file to create the schematic of the design and functionality of the design can be checked by this tool. The Mentor IC design was used to create the layout of the design. This tool use schematic that were generated from mentor design architect to generate the layout. The Mentor calibre was used to perform DRC and LVS of the design. The complete ASIC design flow shown in Fig. 6.8.Fig.6.9 shows the layout of the weight hidden term calculation part of the design.

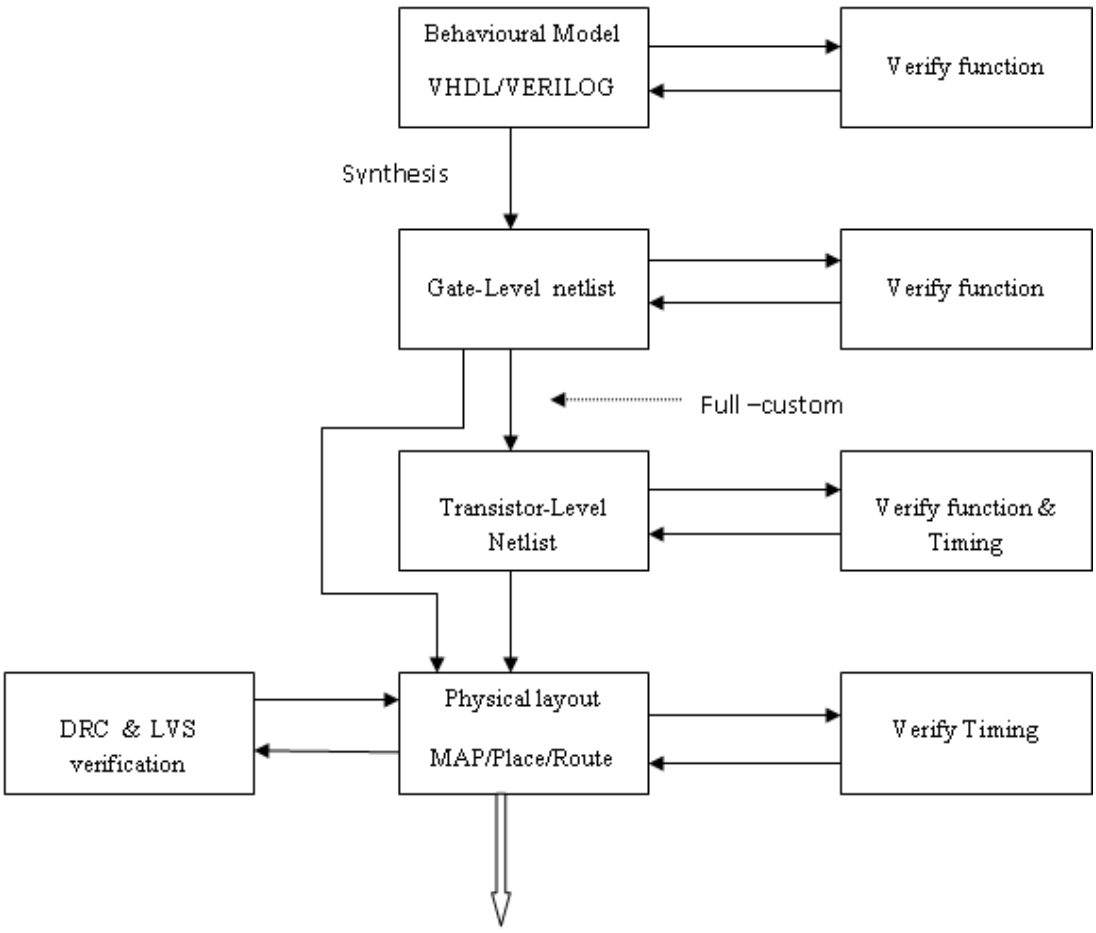


Figure 6.9 Block diagram of the ASIC design Flow

Table 6 Leonardo spectrum synthesize report

Number of ports	82
Number of nets	407
Number of instances	84
Number of accumulated instances	7992
Number of gates	15015

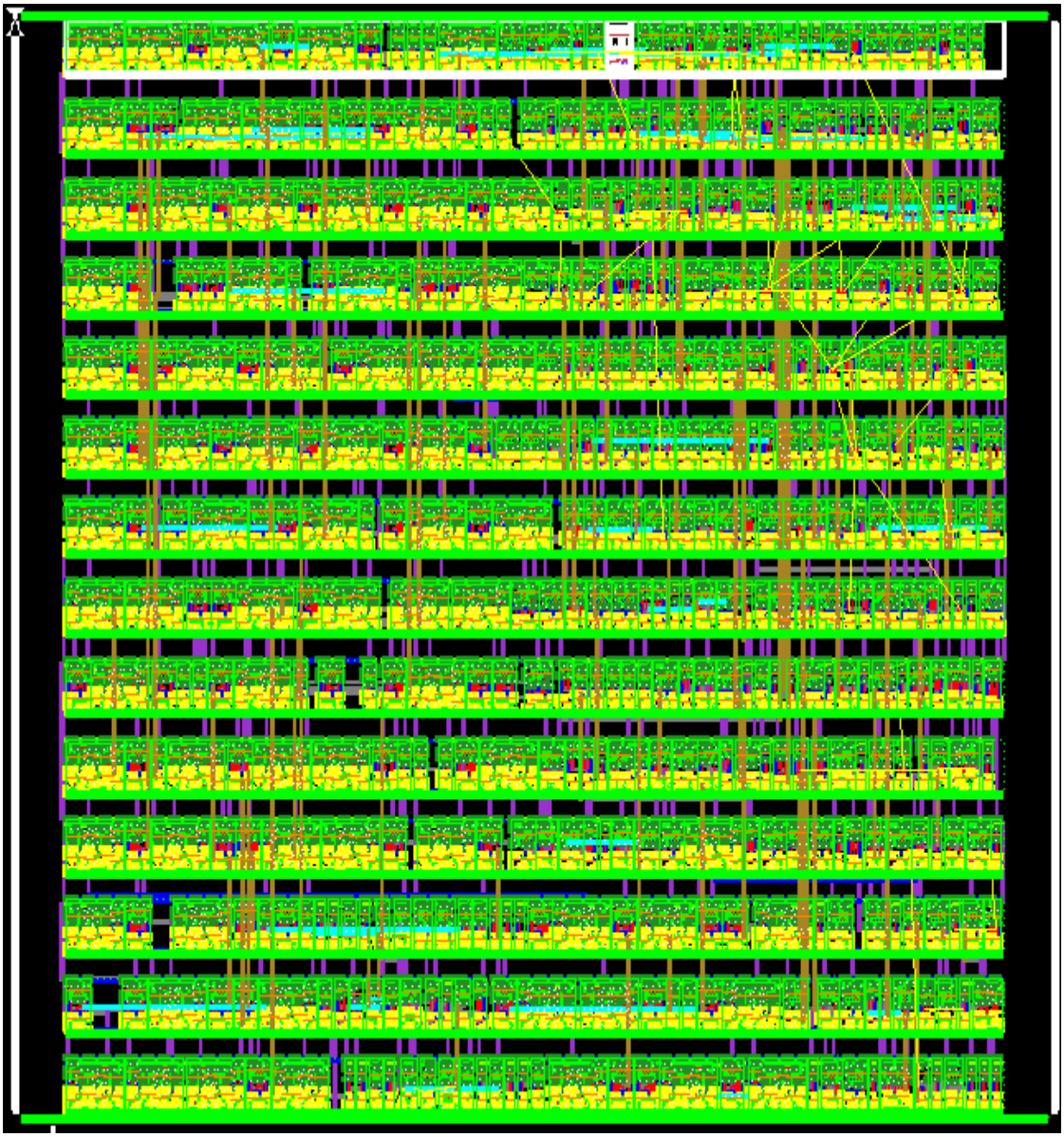


Figure 6.10 Layout of the weight hidden update term

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

Training of a neural network based LMS filter using gradient descent learning algorithm such as backpropagation algorithm requires lot of time on large, complex problems. The learning rate parameter dramatically affects training speed and generalization accuracy of such algorithm. For larger learning rate both accuracy and the training speed will poor and too small learning rate is the wasting of the computational resources. Variable learning rate should be used to avoid these problems. The previous work was focused on the improving the speed of the convergence and fail to focus on generalization accuracy of LMS filter. In this thesis work a PCA based algorithm is used to change the learning rate parameter to increase the training speed and generalization accuracy of the LMS filter. Based upon the results presented in the previous chapter it can be concluded that incorporating the PCA based variable learning rate results in better classification performance of the ANN. Hence, a better LMS filter can be implemented using such an ANN. It was also observed that by using the proposed technique, generalization performance was improved in spite of the training performance being not so good. PCA based algorithm uses output variance to change the learning parameter so this algorithm can be used with the output of the hidden layer so future work will be based on the different learning rate parameter for each layer for fast filtering operation.

One of the major problems associated with the hardware implementation of the LMS filter using artificial neural network is the present of the huge number of multipliers. Multipliers affect the circuit size and performance of the neural network. In this thesis work learning circuit was modified to reduce the multiplier to make the hardware efficient LMS filter, introduction of the multiplexer saves a lot of hardware resources since the number of multipliers were reduced significantly. This saving would increase manifold if the architecture is made more complex. This paves the way for efficient implementation of higher order LMS filters using the ANN. However, there is ample scope on the speed enhancement front and future work may be based on reducing the delay overhead introduced due to the use of the multiplexer.

PUBLICATIONS

1. Deepak Gupta and Ravi Kumar, "Implementation of variable step size back propagation algorithm", *International Conference on Innovations in Electrical, Electronics and Computer science Engineering (IEECSE)*, 22-23 June 2014, Chandigarh.

2. Deepak Gupta and Ravi Kumar, " Artificial Neural Network implementation in FPGA using multiplexer based weight updating for effective resource utilization", *Electronics Letters, IET*.(communicated)

REFERENCES

[1] Abraham, *Artificial Neural Networks*. John Wileyand Sons Ltd, 2005: 901-908.

[2] M. Gupta, L. Jin, N. Homma, Static and Dynamic Neural Network form Fundamentals to Advanced Theory. Wiley-IEEE Press, 2003, pp. 103-170.

[3] F.Vargas, D.Lettnin, M.C.FelippettodeCastro,M.Macarthy,"Electrocardiogram pattern recognition by means of MLP network and PCA: a case study on equal amount of input signal types", VII Brazilian Symposium on Neural Networks (SBRN), 2002, pp. 200- 205.

[4] Yann LeCun, Patrice Y. Simard, and Barak Pearlmutter, "Automatic Learning Rate Maximization by On-Line Estimation of the Hessian's Eigenvectors", International conference on Advances in Neural Information processing system", 1993, vol.5, pp.156-163.

[5] Tetsuy Houya, Hiroyuki Kamata and Yoshihisa Ishida, "Design and implementation of the adaptive filter using Neural networks", in *Proc. Int. Joint Conf. Neural Netw.–IJCNN'93,* 25-29 Oct. 1993, vol.1,pp. 979-982.

[6] S.Dixit and D.Nagaria, "Neural Network Implementation of Least-Mean-Square Adaptive Noise Cancellation", International conference on Issue and Challenges in Intelligent Computing Techniques (ICICT), 7-8 Feb. 2014, pp. 134-139.

[7] Qian Xiao, GE Gang and WANG Jianhui, "The Neural Network Adaptive Filter Model Based on Wavelet Transform", Ninth International Conference on Hybrid Intelligent Systems (HIS), 12-14 Aug. 2009,vol.1, pp. 529-534.

[8] Mohamed Ibnkahla," on the influence of the number of layers on the performance and convergence behaviour of the back propagation algorithm", IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP),21-24 Apr,1997 vol.4, pp. 3209 – 3212.

[9] Yonggang Yan, Junwei Zhao and Zhankui Wang, "An Novel Variable Step Size LMS Adaptive Filtering Algorithm Based on Hyperbolic Tangent Function", International Conference on Computer Application and System Modeling (lCCASM 2010), Jiaozuo, China, 22-24 Oct. 2010, vol.14, pp. V14-233 - V14-236

[10] Ting-Ting Li, Min Shi, Qing-Ming Yi, "An improved variable step-size LMS algorithm",7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), 23-25 Sept. 2011,pp. 1-4.

[11] Bruce Burton, Farrukh Kamran, Ronald G. Harley and Thomas G. Habetler,"Identification and control of Induction Motor Stator Currents Using Fast On-Line Random Training of a neural network", *IEEE Trans. Ind. Electron*, vol.33, no.3, pp. 697-704, May/jun. 1997.

[12] Sherif Kassem Fathy and Mostafa Mahmoud Syiam, "A Parallel Design and Implementation for Backpropagation Neural Network Using MIMD Architecture", 8th Mediterranean Electrotechnical Conference (MELECON)", 13-16 May 1996, vol.3, pp.1472-1475.

[13] Maja Stella, Dinko Begušić, Mladen Russo, "Adaptive Noise Cancellation Based on Neural Network", International Conference on Software in Telecommunications and Computer Networks (SOFTCOM) Sept. 29 2006-Oct. 1 2006 ,pp.306 – 309.

[14] Bernard Widrow, Rodney Winter, "Neural net for Adaptive filtering and Adaptive Pattern Recognition", computer, vol.21, no .3, pp.25-39, 1988.

[15] Lester S.H Ngia, Jones Sjoberg and Mats Viberg, " Adaptive Neural Nets Filter Using a Recursive Levenberg-Marquardt Search direction", Conference Record of the Thirty-Second Asilomar Conference on Signals, Systems & Computers 1-4, Nov. 1998, vol .1,pp.697-701.

[16] Stefananos Kollias and Dimitris Anastassopu, "An Adaptive least square Algorithm for the efficient Training of Artificial neural Networks*", IEEE Trans. Neural Netw.*, vol. 36, no .8, pp.1092-1101 ,1989

[17] O.Stan and E. Kamen, "A local linearized least square algorithm for training feedforward neural networks", *IEEE Trans. Neural Netw.*, vol.11, pp.487-495, 2002.

[18] C.Charalambous, "Conjugate training algorithm for efficient training of the artificial neural network", *IEE Proceedings G on Circuits, Devices and Systems*, vol.139, no.3, pp.301-310, 1992.

[19] Stefano Guarnieri, Francesco Piazza, and Aurelio Uncini, "Multilayer Feeforward Networks With adaptive spline activation function", *IEEE Trans. Neural Netw.*, vol. 10, no. 3, pp. 672-681, 1999.

[20] Wei Jiang and Seong G. Kong, "Block-based neural networks for personalized ecg signal classification", *IEEE Trans. Neural Netw.*, vol.18, no. 6, pp.1750-1761, 2007.

[21] Andrei Dinu, Marcian N. Cirstea, and Silvia E. Cirstea, “Direct neural network hardware implementation algorithm”, *IEEE Trans. Neural Netw.*, vol. 57, no. 5, pp. 1845-1848, 2010.

[22] Alexander Gomperts, Abhisek Ukil, and Franz Zurfluh, “Development and Implementation of parameterized FPGA-based general purpose neural network for online applications”, *IEEE Trans. Neural Netw.*, vol. 7, no. 1, pp. 78-89, 2011.

[23] Zhiying Guo, Jingchang Nan, Jiuchao Li,“Research for adaptive digital predistortion based on BP-LMS”, International conference on computational problem–solving (ICCP), 3-5 Dec. 2010, pp. 131-135.

[24] B.widrow and M. kamenetsky, “statistical efficiency of the adaptive algorithms”, Neural Networks, vol.16, no.5-6, pp. 735-744, june-july 2003.

[25] S.Himavathi, D.Anitha, and A. Muthuramalingam, “Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization”, *IEEE Trans. Neural Netw.*, vol. 18, no. 3, pp. 880-888, May 2007.

[26] John Hertz, Anders Krogh, Benny Lautrup, and Torsten Lehmann, “Nonlinear backpropagation: Doing backpropagation without derivatives of the activation function",*IEEE Trans. Neural Netw.*, vol. 8, no. 6,pp. 1321-1328, Nov. 1997.

[27] A.Cichocki and R.Unbehauen, “Robust estimation of principal components by using neural network learning algorithms”, *Electronics Letters*, *vol*. 29, no. 21, pp. 1-2, oct. 1993.

[28] Xiangyu Kong, Changhua Hu, Hongguang Ma, and Chongzhao Han, “A unified self-stablizing neural network algorithm for principal and minor components extraction”, *IEEE Trans. Neural netw.*, vol. 23, no. 2, pp. 185-197, Feb. 2012.

[29] David Hunter, HaoYu, Michael S. Pukish, Janusz Kolbusz, and Bogdan M. Wilamowski, “Selection of proper neural network sizes and architectures”, *IEEE Trans. Ind. Electron.*, vol. 8, no. 2, pp. 228–240, May 2012.

[30] Antony W. Savich, Medhat Moussa and Shawki Areibi, “The Impact of Arithmetic Representation on Implementing MLP-BP on FPGAs: A Study”, *IEEE Trans. Neural Netw.*, vol. 18, no. 1,pp. 240-252, Jan. 2007.

[31] Fernardo Morgado Dias, Anaantunes and Alexander Manual Mota, “Artificial neural networks: a review of commercial hardware”, *Engineering Applications of Artificial Intelligence* , vol. 17, no. 8, pp. 945-952, 2004.

[32] S.Haykin, Neural Networks: A Comprehensive Foundation. Ed. Martia Horton. Prentic Hall, 2005.

[33] Amos R. Omondi and Jagath C. Rajapakse, “FPGA Implementations of Neural Networks. Springer, 2006.

[34] UCIMachine Learning Repository, http://archive.ics.uci.edu/ml/datasets/Iris