ARTIFICIAL NEURAL NETWORK MODELS FOR DIGITAL IMPLEMENTATION

by

Chuan Zhang TANG

A Dissertation
Submitted to the Faculty of Graduate Studies and Research
Through the Department of Electrical Engineering
in Partial Fulfilment of the Requirements for
the Degree of Doctor of Philosophy at the
University of Windsor

Windsor, Ontario, Canada

1996

ARTIFICIAL NEURAL NETWORK MODELS FOR DIGITAL IMPLEMENTATION

by

Chuan Zhang Tang

Doctor of Philosophy in Electrical Engineering, 1996

University of Windsor, Windsor, Ontario, Canada N9B 3P4

Supervisor: Dr. Hon Keung Kwan

The last decade has witnessed the revival and a new surge in the field of artificial neural network research. This is a thoroughly interdisciplinary area, covering neurosciences, physics, mathematics, economics, and electronics. Although artificial neural networks have found diverse applications in pattern recognition, signal processing, communications, control systems, optimization, among others, this is still a research field with many open problems in the areas of theory, applications, and implementations. Compared with the development in neural network theories, hardware implementation has lagged behind. In order to take full advantages of neural networks, dedicated hardware implementations are definitely needed. Today, harnessing VLSI technology to produce efficient implementations of neural networks may be the key to the future growth and ultimate success of neural network techniques.

This dissertation deals with the development of neural network models

implementation technologies are basically a digital implementation medium, which offers many advantages over its analog counterpart, artificial neural networks must be adapted to an all-digital model in order to benefit from those advanced technologies. In this dissertation, new models of multilayer feedforward neural networks with single term powers-of-two weights, quantized neurons, and simplified activation functions are proposed to facilitate the hardware implementation in digital approach. Dedicated training algorithms and design procedures for these models are also developed. To demonstrate the feasibility of the presented models, performance analysis and simulation results are provided, and VHDL and FPGA designs are implemented. It has been shown that these proposed models can achieve almost the same performance as the original multilayer feedforward networks while obtaining significant improvement in digital hardware implementation in terms of silicon area and operation speed. By using the models developed in this dissertation, a digital implementation approach of multilayer feedforward neural networks becomes very attractive.

To my wife, Barbara Zhou, my son, Joshua Tang,

and

my parents, Shi-xian Tang and Kun-shu Ma

First of all, I would like to acknowledge my dissertation supervisor, Dr. H. K. Kwan, for helpful advice and useful suggestions throughout the progress of my dissertation. Dr. Kwan has introduced me into this exciting field of artificial neural networks (ANNs) and recommended the multiplierless digital implementation of ANNs as my research direction. I would also like to thank Dr. Kwan for providing me research assistantship through his NSERC research grant, and the Department of Electrical Engineering for providing me teaching assistantship. I would like to acknowledge the University of Windsor for granting me the University of Windsor Postgraduate Scholarship (1991-1993) and the Ontario Graduate Scholarship program for awarding me the Ontario Graduate Scholarship (1993-1995).

I am grateful to my external examiner, Dr. M. I. Elmasry of the University of Waterloo, who examined and provided useful comments on the dissertation. Special thanks go to my outside department reader, Dr. R. Du, for his constructive comments on this research. I want to express my sincere thanks to my department reader, Dr. G. A. Jullien, for comments that improved the presentation of this research. I also like to thank Dr. W. C. Miller for serving as my department reader and for his helpful comments. My thanks also go to Ms. S. Ouellette, Mr. J. Novosad, and Mr. A. Johns for their assistance during the progress of this research.

Finally, I wish to express my sincere appreciation to my wife for her constant encouragement and understanding, and to my parents for their care and affection.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

The last decade has witnessed the revival and a new surge in the field of artificial neural network research. The term neural network originally referred to a network of interconnected neurons which are basic building blocks of the nervous system. Today, this term, or more properly artificial neural networks, has come to mean any computing architecture that consists of a massively parallel interconnection of simple neuron-like processors. These architectures have been inspired by our current understanding of the brain, but do not necessarily conform strictly to that understanding.

The fact that an one-year-old baby is much better and faster at recognizing objects, faces, and so on than even the most advanced artificial intelligence system running on the fastest supercomputer may imply that there are numerous problems in the real world that are difficult with today's computing technology but are easily solved by human beings or even animals. In view of this fact, the research on artificial neural networks has been

thoroughly interdisciplinary area, covering neurosciences, physics, mathematics, economics, computer sciences, and electronics. There are thousands of new comers entering this exciting research field every year.

1.1 History of Artificial Neural Networks

The initial effort in Artificial Neural Network (ANN) research may be traced back to early 1940s when McCulloch and Pitts [McCulloch and Pitts, 1943] modeled a neuron as a simple threshold binary device to perform logic functions. In this model, each neuron can be in only one of two possible states and has a fixed threshold. It can receive inputs from excitatory synapses, all of which have identical weights. It can also receive inputs from inhibitory synapses, whose action is absolute; that is, if the inhibitory synapse is active, the neuron cannot turn on.

Later in 1949, Hebb [Hebb, 1949] published his book *The Organization of Behaviour* and for the first time proposed a neural learning rule for synaptic modification that has been known as the Hebb rule. Hebb stated that if one neuron repeatedly fires another, some change will take place in the connecting synapse to increase the efficiency of such firing. This correlational synapse modification rule has become the basis for many neural network models

1987 and 1988] and Hopfield Network [Hopfield, 1982].

The most significant work at the early stage of neural network research was the Perceptron model which was developed by Rosenblatt [Rosenblatt, 1959 and 1962] in late 1950s and early 1960s. It was the first precisely specified, computationally oriented neural network. The basic classification element in the Perceptron is the R-unit, which forms a weighted sum of the active elements times the connection strengths. The unit has a threshold. If the sum is greater than the threshold, the R-unit takes the value 1; if less than the threshold, the unit takes the value -1. This simple network generated much interest when initially developed because of its ability to learn to recognize simple patterns.

However, the Perceptron model has its own limitations. It is capable of realizing only those linearly separable functions. This weakness was seized by Minsky and Papert [Minsky and Papert, 1969] in 1969 when they proved mathematically that the Perceptron cannot be used for complex logic functions. The publication of their famous book, *Perceptron*, caused a sharp decline in research on neural networks.

The present impetus in neural network research is due in part to the

these papers, he presented a recurrent model of neural computation that is based on the interaction of neurons. He also pointed out that there are emergent computational capabilities at the network level that are nonexistent at the single neuron level. Such neural networks are now known as Hopfield networks.

During the 1970s when no one else was working on neural networks, Steven Grossberg and Teuvo Kohonen were making significant contributions. In 1980s, Grossberg and Carpenter [Carpenter and Grossberg, 1983, 1987, and 1990] developed their Adaptive Resonance Theory (ART) neural network architectures, based on the idea that the brain spontaneously organized itself into recognition codes. These are self-organizing neural implementations of pattern clustering algorithms, that is, they form clusters and are trained without supervision.

At the same time, Kohonen [Kohonen, 1982 and 1984] proposed his idea of a self-organizing map, based on the fact that the brain is organized, in many places, so that aspects of the sensory environment are represented in the form of two-dimensional maps; the placement of neurons is orderly and often reflects some physical characteristic of the external stimulus being sensed. It is a sheet-like artificial neural network, the cells of which become specifically tuned to

learning process.

In the mid 1980s, David Rumelhart and his colleagues rediscovered the backpropagation algorithm [Rumelhart et al., 1986], which was originally discovered by Paul Werbos [Werbos, 1974] when he applied the LMS algorithm to multiple layers of Perceptrons in the study of social sciences. The publication of their landmark book on parallel distributed processing [Rumelhart and McClelland, 1986] established the backpropagation algorithm and multilayer feedforward neural networks (MFNNs) as the major paradigm of the field of neural network research. This work and earlier works have finally galvanized a large number of scientists into thinking in terms of collective neural computation rather than single neurons.

From the late 1980s through the 1990s, with some neural network paradigms having reached a considerable degree of maturity, more and more efforts have been directed towards the area of neural network implementation as well as applications. The pioneering work by Mead [1989] marked the beginning of a new era in hardware implementation of neural networks. Since then, with the technological advances of VLSI circuits and systems, the field of VLSI artificial neural networks experienced an exponential growth and a new engineering discipline was born. Various work on analog, digital, pulse-

and Elmasry, 1992, Oh and Salam, 1993 and 1994, Kim and Shanblatt, 1992, Zaghloul et al., 1994, Sheu and Choi, 1995].

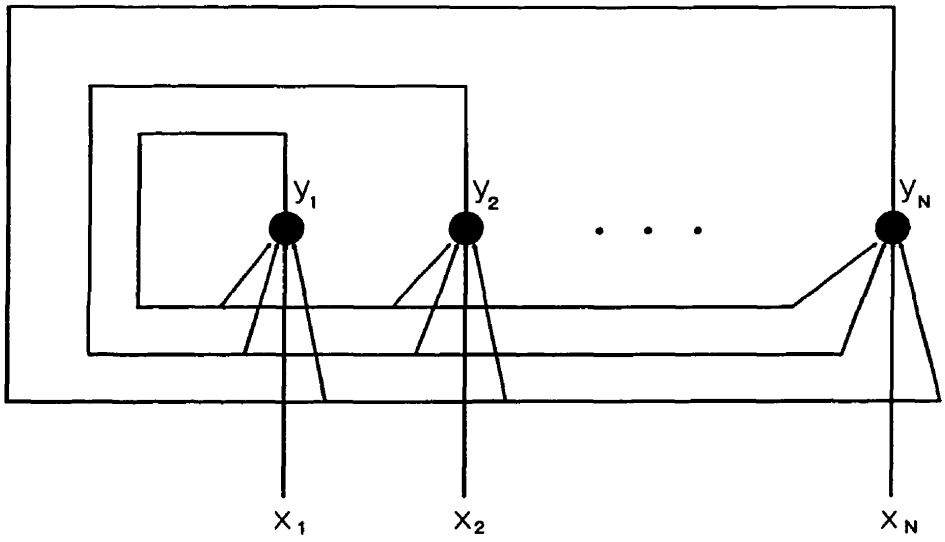
By far, Hopfield networks, ART networks, self-organizing maps, and multilayer feedforward networks are the most popular artificial neural network models that have ever been proposed. Other important ANN models may include Neocognitron[Fukushima, 1975 and 1980], Boltzman machines[Hinton and Sejnowski, 1986][Ackley et al., 1988], bidirectional associative memories (BAM's)[Kosko, 1987 and 1988], and fuzzy ARTMAP[Carpentaer et al, 1992 and 1993].

1.2 ANN Features

Generally speaking, an artificial neural network model is specified by three factors:

- a set of basic processing elements, called neurons (or nodes)
- a specific topology of weighted interconnections between neurons
- a training or learning rule which specifies an initial set of weights and indicates how weights should be adapted during use to improve performance

sums a number of weighted inputs and passes the result through a nonlinear activation function. More complex neurons may include temporal integration or other types of time dependencies and more complex mathematical operations than summation. The topologies of ANNs fit broadly into two classes: recursive and feedforward. A recursive ANN is a network with feedback. In such a network, each neuron receives as input a weighted output from every other neuron in the network, possibly including itself. A typical example of recursive neural networks is the Hopfield network shown in Figure 1.1. A feedforward network does not contain any closed synaptic loops or feedback. The most famous feedforward network is the Multilayer Feedforward Neural Network which will be discussed thoroughly in Chapter 2. Training algorithms for ANNs can be described either as supervised training or unsupervised training. The distinction between supervised and unsupervised algorithms depends on information they use. Supervised training, also called learning with a teacher, assumes that the desired output of the network is known. This is then used to form an error signal which is used to update the weights. On the other hand, in unsupervised training the desired output is not known, but instead training is based simply on input/output values. Such training algorithms usually act to extract features from sets of input data.



$$y_j = f\left(\sum_{i=1}^{N} w_{ij} y_i + x_i\right)$$

Figure 1.1 A Hopfield Network

The potential benefits of neural networks extend beyond the high computation rates provided by massive parallelism. Some of these benefits are outlined below.

- Neural networks typically provide a greater degree of robustness or fault tolerance than von Neumann sequential computers because there are many more processing elements, each with primarily local connections.

- Neural networks have the ability to adapt to changes in the data and

areas such as speech and image recognition. Adaptation also provides a degree of robustness by compensating for minor variabilities in characteristics of processing elements.

- Neural networks can perform functional approximation and signal filtering operations which are beyond optimal linear techniques because of their nonlinear nature.

- Neural network classifiers are non-parametric and make weaker assumptions concerning the shapes of underlying distributions than traditional statistical classifiers.

- Neural networks are model-free classifiers because they approximate functions with raw sample data.

Because the motivation of ANN research comes mainly from the fact that humans are much better at pattern recognition than digital computers, there is no surprise that ANNs have found many applications in vision processing and speech processing[Sejnowski et al., 1987][Lang et al., 1990][Taylor, 1990][Levin, 1993][Kung and Taur, 1995][Zhang and Fulcher, 1996]. Besides, ANNs have also been applied to the areas of optimization[Tank and Hopfield,

and Winarske, 1988][Choi et al, 1993][Kechriotis, et al., 1994][Ansari, et al., 1995], control systems[Nguyen and Widrow, 1990][Narendra and Parthasarathy, 1990][Sebald and Schlenzig, 1994][Sanger, 1994][Lewis, et al., 1996], and medical applications[Nikoonahad and Liu, 1990][Nekovei and Sun, 1995][Choong, et al., 1996], to mention a few.

1.3 Motivations and impact of this research

Although artificial neural networks have found diverse applications in control, signal processing, and pattern recognition, among others, this is still a research field with many open problems in the areas of theory, applications, and implementations. Compared with the development in neural network theories, hardware implementation has lagged behind. In order to take the full advantages of neural networks, there has to be dedicated hardware implementations. Research in hardware implementations belongs to the main areas of activity in the field of neural networks and plays a unique role in the progress of the entire field. The surge of interest in neural networks, which started in mid eighties, was to a large extent caused by advances in VLSI technology. Today, harnessing VLSI technology to produce efficient implementations of neural networks may be the key to the future growth and ultimate success of neural network techniques.

success in their own application domains. Each technique has its own pros and cons. The selection between digital and analog circuits depends on many factors, for example, speed, precision, adaptiveness, programmability, and transfer/storage of signals. This dissertation deals with the topic in digital VLSI implementations of artificial neural networks. An all-digital artificial neural network VLSI implementation offers several advantages over its analog counterpart[White and Elmasry, 1992][Kung, 1993].

1) Digital design has an overall advantage in terms of system-level performance. Dynamic range and precision are critical for many complex neural network models. Digital implementation offers much greater flexibility of precision than its analog counterpart.

2) In most real-world applications, neural networks are embedded in existing digital systems. An all-digital ANN implementation provides compatibility.

3) Real-world applications usually require large scale neural networks, in some cases, of tens of thousands neurons and synapses. Digital VLSI is more appropriate at this level of complexity, whereas analog VLSI suffers from noise and difficulties in fabricating high-precision resistors and

4) Larger ANN's may require multichip implementations, and an analog implementation makes it more difficult to transfer signals from chip to chip, and also to match board-level capacitive loads and time constants. An all digital technique makes it easier to transfer signals form chip to chip.

5) At any given time, digital VLSI technology is always more mature than its analog counterpart in terms of fabrication technology and simulation and design automation tools. It also offers a wide range of fabrication technologies, including such technologies as ASIC for application oriented design and FPGA for rapid prototyping.

6) Real-world neural network applications may suffer from I/O bottlenecks, which are best addressed by digital techniques such as input buffers, shift registers, and pipelining. Moreover, power dissipation reduction techniques, such as dynamic logic and complementary operation, can be used.

7) Digital implementation offers a homogeneous implementation environment between the processing elements and the on-chip or

Because the state-of-the-art VLSI implementation technologies are basically a digital implementation medium, artificial neural networks must be adapted to an all-digital model in order to benefit from these technologies.

Meanwhile, there are also certain shortcomings of digital VLSI implementation that must be resolved in order to implement ANN's efficiently. Most ANN neuron calculations involve a weighted sum of the neuron inputs, and the multiplier required for this multiply-accumulate operation is slow and consumes large silicon area in a digital VLSI implementation.

The solution of this problem may be approached from 1) advances in VLSI technologies; and 2) adapting existed models to today's available technologies. This dissertation deals with the latter issue and will develop new models of MFNN's which are suitable for digital hardware implementations.

In silicon design, the cost of a chip is primarily determined by its two-dimensional area. Smaller chips are cheaper chips. Within a chip, the cost of an operation is roughly determined by the silicon area needed to implement it. As pointed out previously, in digital neural network systems, multiplications are area-consuming and slow operations and there are massive such operations

inputs to neurons and their corresponding weights can be reduced, a reduced silicon area and higher speed will be resulted. Consequently, a lower cost will be achieved. The basic ideas behind the models proposed in this thesis are powers-of-two coefficients and functions, which will result in the replacement of multiplications by shift operations, which are much faster and have much smaller area, as well as the simplification of the realization of nonlinear activation functions. By using these proposed models, certain computational burdens in digital implementations will be alleviated without jeopardizing the performance of the ANN system, and a digital implementation scheme becomes very attractive.

1.4 Literature Survey

The idea of powers-of-two factors was first proposed for digital filter implementations and has been successfully applied to many designs[Kwan and Chan, 1989 and 1990][Lim and Parker, 1983a and 1983b][Lim et al. 1982][Lim and Constantinides, 1979][Xue and Liu, 1986][Zhao and Tadokoro, 1988], in which multiplications were either replaced by shift only operations or reduced to shift operations plus very few additions, depending on how many terms of powers-of-two were used. Single term powers-of-two factors are most desired because they require the least operations in hardware implementation.

artificial neural networks require very high density of computations including large number of multiplications. In such cases, powers-of-two factors or at least quantized weights are needed to reduce the amount of computation and hardware requirements.

A digi-neocognitron model for VLSI implementation was proposed by White and Elmasry[White and Elmasry, 1992]. The original neocognitron (NC) model[Fukushima 1980] was adapted to an efficient all-digital implementation for VLSI. The new model, the digi-neocognitron (DNC), has the same pattern recognition performance as the NC. The DNC model was derived from the NC model by a combination of preprocessing approximations and the definition of new model function, e.g., multiplication and division are eliminated by conversion of factors to powers of 2, requiring only shift operations. The DNC model has substantial advantages over the NC model for VLSI implementation with a two to three orders of magnitude improvement in the area-delay product.

A one-dimensional Kohonen network with quantized weights and inputs was studied by Thiran and Hasler [Thiran and Hasler, 1994]. The implementation of a Kohonen network on a digital circuit realization yields the quantization of all the input signals and weight values. It is crucial to see whether this modification perturbs the self-organizing feature. In [Thiran and

organization property of the original Kohonen network for the one-dimensional case is conserved when the weights are quantized provided that its parameters are well chosen.

The application of discrete weights and the powers-of-two technique in multilayer feedforward neural networks has been studied by several authors [Marchesi et al. 1990][Nakayama et al. 1990][Piazza et al. 1993] and most recently by Marchesi et al.[Marchesi et al. 1993]. In [Marchesi et al. 1993], a fast neural network model was proposed for digital VLSI implementation along with a dedicated learning procedure. In their model, weight values were restricted to powers-of-two or sum of powers-of-two and adaptive biases and automatic learning rate control were employed to compensate the quantization error.

It was pointed out that one of the major problems of digital architectures implementing neural networks, affecting both performance and chip area, is the presence of multipliers. The multiplications between inputs and weights, which are slow compared to other operations and require a lot of chip area if a direct VLSI implementation is planned, can be the bottle-neck of the system. By introducing the idea of powers-of-two weights, it is possible to substitute multiplications with simple shifts or much fewer shift-and-add operations,

However, the learning algorithm developed in [Marchesi et al. 1993] was not very effective. Its convergence performance was not satisfactory due to lack of sufficient adjustable parameters. For a given problem, the starting point of their proposed learning algorithm is the solution of the same mapping problem with a conventional MFNN having continuous weights, applying the BP learning algorithm. The obtained weights are then quantized to powers-of-two values and the BP will be applied again to adjust the discretized weights in hope to converge to the final solution. It is obvious that there is little room to improve the network by adjusting the quantized weights. These quantized weights are distributed in some discrete points and the gaps among these points are usually much larger than the weight updating amount required by the BP algorithm, so they are not suitable for fine tuning the neural network, especially when single term powers-of-two format is used. Moreover, the quantization scheme they adopted appears to be fairly complicated. Actually, the minimization of the sum of squared weight quantization error as adopted in [Marchesi et al. 1993] does not necessarily reduce the sum of squared output error of the network due to the non-linear nature of the neural network systems. It will be much simpler to adopt direct quantization of weights to their nearest powers-of-two values as proposed in this dissertation.

digital VLSI implementation, no investigation on real hardware issues had been presented in the published work and that was a major weakness of their paper.

In view of this situation, a new algorithm for design of MFNNs with single term powers-of-two (STPT) weights will be presented in Chapter 3 of this dissertation, which has more degrees of freedom to adapt to a given problem. In Chapter 4, an all new model of MFNNs with quantized neurons will be proposed for digital hardware implementation, in which multiplications can still be avoided and the implementation of nonlinear activation functions will also be simplified.

1.5 Organization of this Dissertation

The remaining of this dissertation is organized as follows.

Chapter 2 begins with the multilayer feedforward neural network model. The structure of the network and the backpropagation (BP) learning algorithm are discussed. Some modifications to the BP algorithm are also presented. At the end of this chapter, the issue of hardware implementation of MFNNs is addressed, which introduces the necessity of the models to be developed in Chapters 3 and 4.

with single term powers-of-two (STPT) weights. The design procedures are provided along with simulation results.

Another MFNN model - MFNNs with quantized neurons will be proposed in Chapter 4. The concept of quantized neuron is introduced and followed by the corresponding training algorithm. The design methodology of such networks is developed and the mapping capability of the new model is examined.

Chapter 5 presents more MFNN models suitable for digital implementations, starting with a simplified sigmoid activation function which is easy for direct hardware realization, and followed by some MFNN models designed to accommodate continuous and discrete input patterns.

Chapter 6 concludes the dissertation and suggests possible future research directions.

Chapter 2

MULTILAYER FEEDFORWARD NEURAL NETWORKS

As mentioned in Chapter 1, multilayer feedforward neural networks (MFNNs) are one of the most important and widely used ANN models. In this Chapter, a review of the structure, properties, and training algorithms of MFNNs will be presented as a preparation for the new models proposed in the following chapters.

2.1 MFNN Architecture

A Multilayer Feedforward Neural Network is a unidirectional network in which adjacent layers are fully connected. The general structure of such a network can be illustrated by Fig.2.1. For an L-layer MFNN, there is an input layer (denoted as layer 0) with N_0 input nodes, an output layer (denoted as layer L) with N_L output neurons, and one or more hidden layers with N_h ($h = 1, 2, ..., L-1$) neurons at layer h.

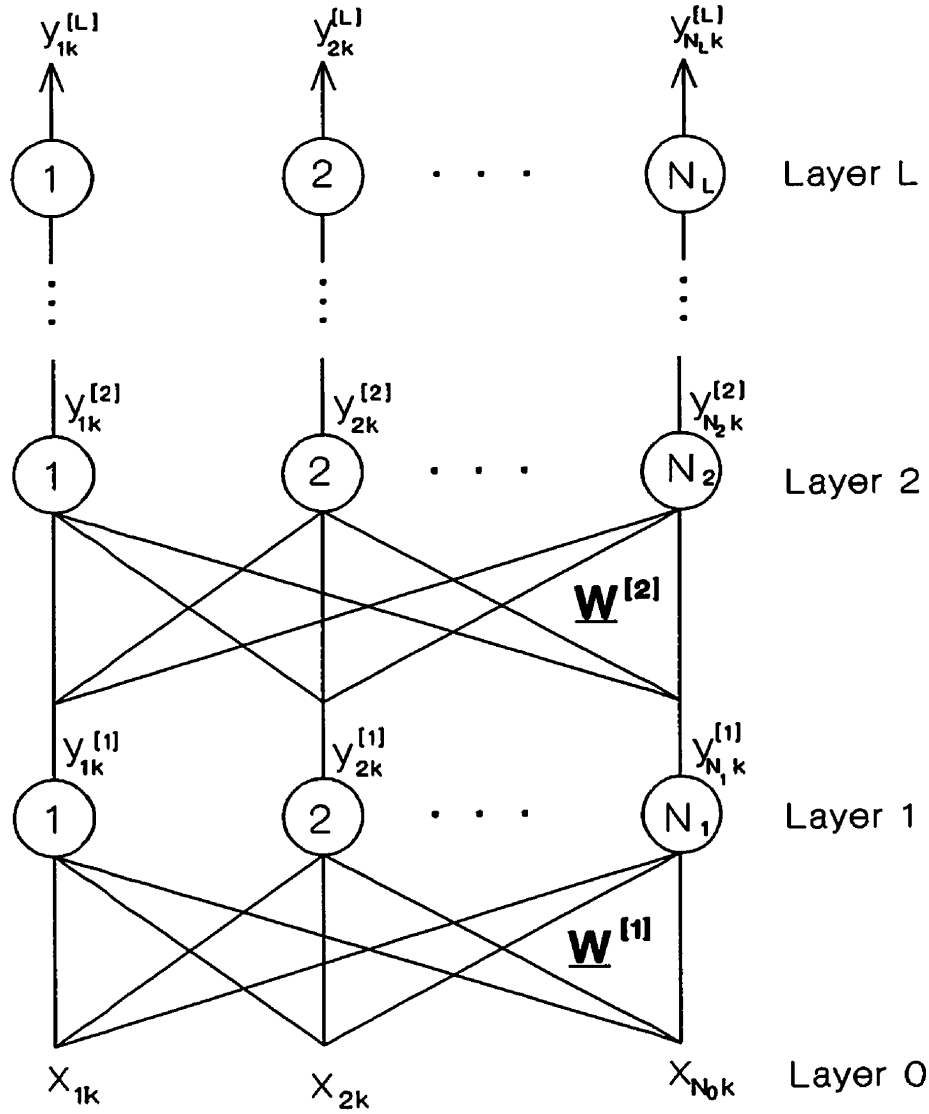


Figure 2.1 A Multilayer Feedforward Neural Network

summations, and calculation of nonlinear functions. A typical neuron of MFNNs is illustrated in Fig.2.2, where x_i's are inputs to the neuron, w_i's are corresponding connection weights, z is the net input to the neuron before activation, and y is the output of the neuron. The input-output relationship of the neuron can be described as

$$z = \sum_{I=1}^{N} w_I x_I + b \tag{2.1}$$

and

$$y = F(z) \tag{2.2}$$

where F(•) is a nonlinear activation function. Some commonly used forms of F(•) include the hardlimit function, threshold logic function, and sigmoid function, which are shown in Fig.2.3.

Usually, the collective features of neural networks are of more interest than those of single neuron. When the entire network is concerned, the input-output relationship of a multilayer feedforward neural network can be described by the following set of equations

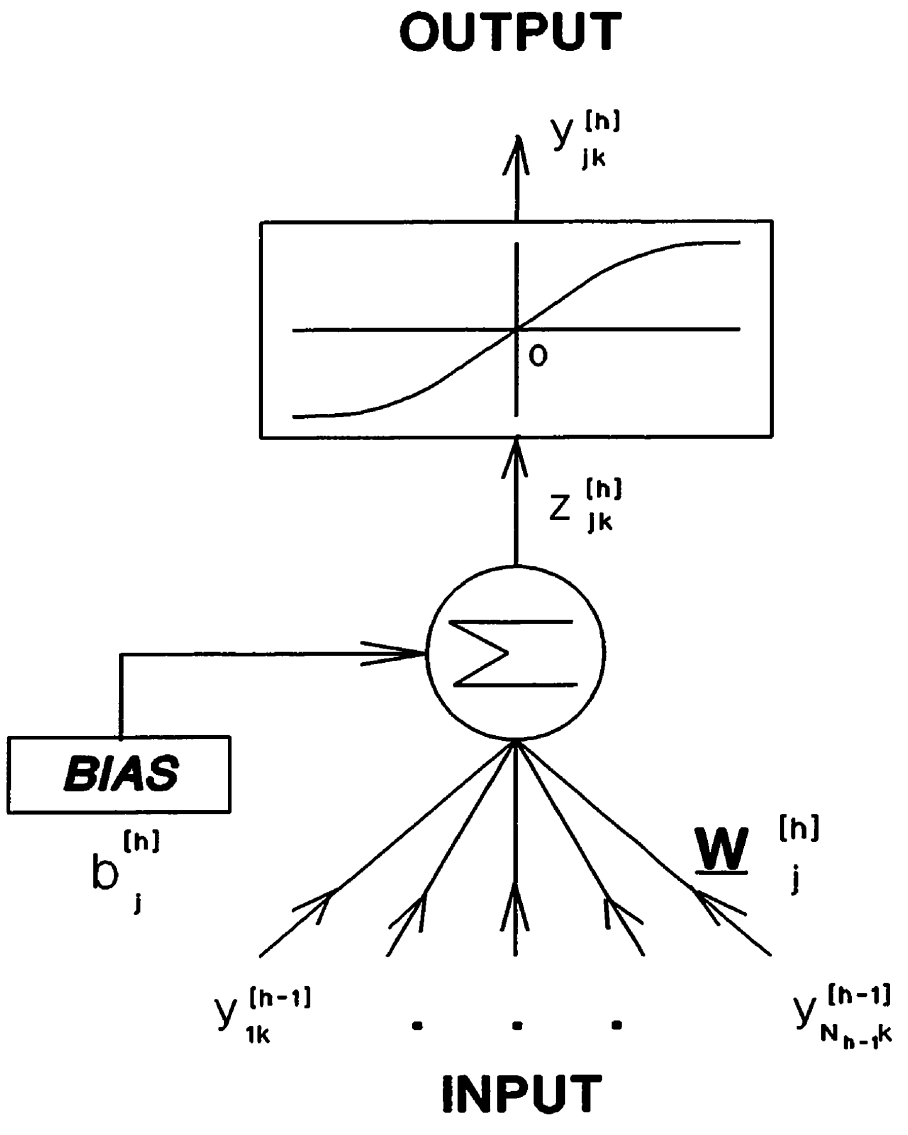
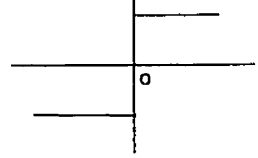
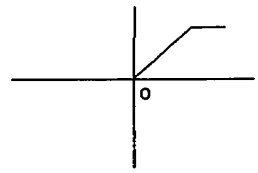


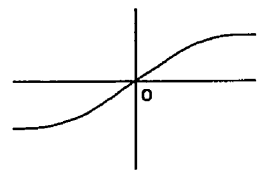
Figure 2.2 A Typical Neuron in MFNNs



Hard Limiter



Threshold Logic



Sigmoid

Figure 2.3 Commonly Used Nonlinear Activation Functions

$$y_{jk}^{[h]} = F\left(\sum_{i=1}^{N_{h-1}} w_{ij}^{[h]} y_{ik}^{[h-1]} + b_j^{[h]}\right) \tag{2.3}$$

with

$$y_i^{[0]} = x_{ik} \qquad \text{for } i = 1, 2, \ldots, N_0 \tag{2.4}$$

where $y_{ik}^{[h-1]}$ is the output of neuron *i* at layer *h-1*; $w_{ij}^{[h]}$ is the connection weight between neuron *i* at layer *h-1* and neuron *j* at layer *h*; $b_j^{[h]}$ is the bias of neuron *j* at layer *h*; x_{ik} is element *i* of the input pattern when pattern *k* is presented to

of the network.

Due to the use of nonlinearities within neurons, multilayer feedforward neural networks overcome many of the limitations of single layer perceptrons[Rosenblatt, 1958 and 1962], which can only be applied to linearly separable problems. Now, it is possible to use MFNNs to distinguish between arbitrarily complex decision regions. Actually, it has been shown that MFNNs with a single hidden layer and arbitrarily bounded and nonconstant activation functions are universal approximators provided that sufficiently many hidden neurons are available [Hornik et al., 1989]. Although one hidden layer is usually sufficient, sometimes a problem is easier to solve with more than one hidden layer. In this case, easier means that the network learns faster.

For a given problem, the parameters of a MFNN, such as the number of layers, the number of hidden neurons, the formula of activation functions, and the values of weights, need to be determined before the neural network can be applied to solve the problem. Among them, how to obtain the appropriate values of weights is the major concern, i.e., how to train the network to adapt to each particular problem. Learning algorithms for MFNNs has been a research topic since 1960s. So far, the backpropagation (BP) algorithm[Rumelhart et al. 1986] has been the most popular one in spite of the existence of some

1992], simulated annealing [Kirkpatrick et al., 1988][Szu, 1986], Choice of Internal Representations (CHIR) [Grossman et al., 1989], and layer by layer optimization [Ergezinger and Thomsen, 1995][Wang and Chen, 1996]. In the next section, the BP algorithm will be reviewed and some of possible modifications will be discussed.

2.2 The Backpropagation Algorithm

Consider the multilayer feedforward neural network as shown in Fig.2.1. Adopting the same definition as in Section 2.1, the input-output relationship of the network can be described as follows:

$$y_{jk}^{[h]} = F\left(\sum_{i=1}^{N_{h-1}} w_{ij}^{[h]} y_{ik}^{[h-1]} + b_j^{[h]}\right) \tag{2.5}$$

with

$$y_i^{[0]} = x_{ik} \qquad \text{for } i = 1,2,\ldots,N_0 \tag{2.6}$$

Usually, the output of the network is not exactly the same as the desired output during the learning process, there is an error associated with each pattern. The error can be measured as the sum of the squared difference

a pattern k is presented to the network, the error at neuron j of the output layer is calculated as

$$e_{jk} = (t_{jk} - y_{jk}^{[L]})^2 \tag{2.7}$$

Here t_{jk} represents element j of the target pattern k and L refers to the output layer of an MFNN with L layers. Then the sum of squared error (SSE) related to a particular pattern k can be defined as

$$e_k = \sum_{j=1}^{N_L} (t_{jk} - y_{jk}^{[L]})^2 \tag{2.8}$$

And, the total squared error (TSE) over all patterns in the training set is defined as

$$TSE = \sum_{k=1}^{K} e_k \tag{2.9}$$

Where K is the number of patterns contained in the training set. A learning process, or training algorithm, is attempting to reduce the output error by adjusting the weights and, in some situations, other parameters of the network.

First proposed by Werbos [Werbos, 1974] and rediscovered by Rumelhart

method which allows updating of the weights of a feedforward neural network. The idea of the gradient descent algorithm is to make the change in a weight proportional to the negative derivative of a cost function, such as TSE, with respect to that weight. Hence, by following this rule, the change in weight $w_{ij}^{[h]}$ (due to pattern k) can be calculated as

$$\Delta_k w_{ij}^{[h]} = -\varepsilon \frac{\partial e_k}{\partial w_{ij}^{[h]}} \qquad \text{for } 1 \le h \le L \tag{2.10}$$

where ϵ is a learning rate parameter of weights, which controls the pace of each weight adjustment.

By applying the chain rule (see Appendix A), the following formulas can be obtained for weight updates in the BP learning algorithm

$$\Delta_k w_{ij}^{[h]} = \varepsilon \delta_{jk}^{[h]} y_{ik}^{[h-1]} \tag{2.11}$$

where

$$\delta_{jk}^{[h]} = F'(z_{jk}^{[h]}) \sum_{i=1}^{N_{h+1}} \delta_{ik}^{[h+1]} w_{ji}^{[h+1]} \qquad \text{for } h<L \tag{2.12}$$

and

Here F'( ) is the derivative of the activation function F( ). It can be seen that the update of weights starts from the output layer down to the input layer. In this process, the derivative of the activation function plays a very important role. If the derivative is zero, no learning will occur even though there is a large amount of error. A very flat activation function, i.e., an activation function with very small values of derivative, may result in a very long learning process.

In summary, the BP algorithm may be carried out as follows:

Step 1: Initialization of weights with small random numbers

Step 2: Presentation of input patterns and desired output patterns

Step 3: Calculation of actual output and squared output error

Step 4: Check TSE$<E_0$? If yes, then stop; otherwise proceed to Step 5

Step 5: Update of weights

Step 6: Go back to Step 2

Although the BP algorithm remains as the most popular and effective way to train MFNNs, there are some drawbacks accompanying it. The convergence speed of the BP algorithm is usually slow, and, in some situations,

In view of these problems, modifications to the original BP algorithm have been proposed by some researchers [Vogl et al., 1988][Jacobs, 1988][Minai and Williams, 1990][Schreibman and Norris, 1990][Kruschke and Movellan, 1991][Lee et al., 1991].

2.3 Improvements to the BP Algorithm

Since backpropagation suffers from low convergence speed, modifications to the original algorithm have been proposed to improve the learning speed. Some of these modifications are discussed in this section.

2.3.1 Adjustable Learning Rate

Choosing an appropriate learning rate parameter ϵ is a key factor in controling the learning speed of the backpropagation. At different stages of a learning process, the best value of ϵ may be different. Instead of using a constant learning rate for the entire learning process, a good idea is to adjust it automatically as learning progresses [Jacobs, 1988][Vogl et al., 1988]. The usual approach is to check whether a particular weights update did actually decrease the output error. If it didn't, then the process overshot, and ϵ should be reduced. On the other hand, if several steps in a row have decreased the error, then perhaps the learning process is being too conservative, and ϵ could

$$\Delta\varepsilon = \begin{cases} +a & \text{if } \Delta E<0 \text{ consistently} \\ -b\varepsilon & \text{if } \Delta E>0 \\ 0 & \text{otherwise} \end{cases} \qquad (2.14)$$

Where ΔE is the difference between the network output errors at two consecutive times t+1 and t, and a and b are positive constants.

2.3.2 Momentum Term

As stated above, it is difficult to choose an appropriate learning rate parameter ϵ for a particular problem. The learning can be very slow if the learning rate ϵ is too small, and can oscillate widely if ϵ is too large. A momentum term can be introduced to deal with this problem [Phansalkar, 1994]. This scheme is implemented by giving a contribution from the previous weight update to each of the current weight change:

$$\Delta w(t+1) = -\varepsilon \frac{\partial E}{\partial w} + \mu \Delta w(t) \qquad (2.15)$$

where $\Delta w(t+1) = w(t+1)-w(t)$, $\Delta w(t) = w(t)-w(t-1)$, and μ is the momentum parameter which is a positive number between 0 and 1.

If the learning process is marching through a plateau region of the error surface, then $(\partial E/\partial w)$ will be about the same at each time-step and the above

$$\Delta w = -\frac{\varepsilon}{1-\mu}\frac{\partial E}{\partial w} \qquad (2.16)$$

with an effective learning rate of $\epsilon/1\text{-}\mu$. On the other hand, in an oscillatory situation, Δw responds only with coefficient ϵ to instantaneous fluctuations of $(\partial E/\partial w)$. The overall effect is to accelerate the long term trend by a factor of $1/(1\text{-}\mu)$, without magnifying the oscillations.

2.3.3 Adjustable Biases and Activation Functions

Since biases can be considered as the weights which are connected to constant input 1, it is also possible to adjust biases using the gradient descent method as in weight adaptations. To be specific, the following formula can be employed.

$$\Delta_k b_j^{[h]} = -\varepsilon_b \frac{\partial \Theta_k}{\partial b_j^{[h]}} \qquad (2.17)$$

Where ϵ_b is the step size for bias adjustment and $b_j^{[h]}$ is the bias of neuron j at layer h. Similar to weight updates, the following equations may be obtained by using the chain rule

where

$$\delta_{jk}^{[h]} = F'(z_{jk}^{[h]}) \sum_{i=1}^{N_{h+1}} \delta_{ik}^{[h+1]} w_{ji}^{[h+1]} \tag{2.19}$$

for $h < H$, and

$$\delta_{jk}^{[H]} = 2(t_{jk} - y_{jk}^{[H]}) F'(z_{jk}^{[H]}) \tag{2.20}$$

Here $F'(\ )$ is the derivative of the activation function $F(\ )$, and t_{jk}, $y_{jk}^{[h]}$, and $z_{jk}^{[h]}$ have the same definition as in Section 2.2.

It is well known that the nonlinear activation functions play a very important role in the performance of MFNNs. Also, it can be seen from (2.11)-(2.13) that the derivative of the activation function $F'(x)$ is a key factor in the weight adaptation process. This indicates that the learning process can be improved by controlling the shape of the activation functions. For the most widely used sigmoid activation function

$$F(x) = \frac{1 - e^{-\alpha x}}{1 + e^{-\alpha x}} \tag{2.21}$$

the shape of the function can be controlled by the slope, which, in turn, can be

the idea of the gradient descent method, i.e.,

$$\Delta_k \alpha_j^{[h]} = -\varepsilon_\alpha \frac{\partial e_k}{\partial \alpha_j^{[h]}} \tag{2.22}$$

Some results [Kruschke and Movellan, 1991][Tang and Kwan, 1993] have shown that this method can speed up the learning process significantly. The detailed derivation of adaptation equations and discussion will be presented in Chapter 3.

2.4 Hardware Implementations of MFNNs

As pointed out in Chapter 1, digital implementation of neural networks is very attractive, especially with the currently available ASIC and FPGA technologies. However, when applied to MFNNs, a direct implementation scheme may not be appropriate due to the large number of multiplications involved.

If we look at a typical neuron in an MFNN, a direct implementation will generate a cell as shown in Fig.2.4. Among the functional blocks involved, multipliers are not favoured by digital VLSI technologies since they consume large chip areas and have slow speed. Implementing nonlinear activation

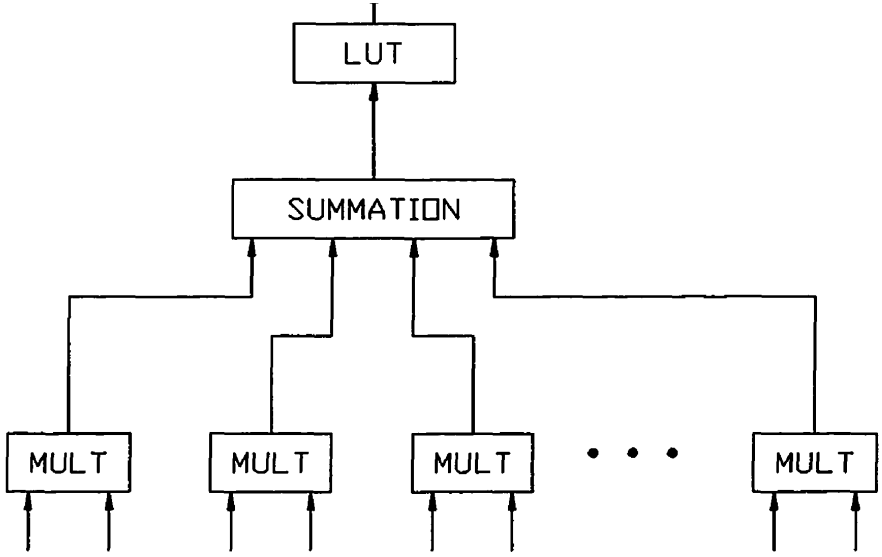


Figure 2.4 Block diagram of direct implementation of a neuron in MFNNs

functions using look-up-table method will also require large silicon area. Large silicon area means high cost. Reducing the cost always has high priority in any real applications.

In chapter 3, an MFNN model using single term powers-of-two weights will be proposed and consequently multipliers will be replaced by shifters. And, in Chapter 4, an MFNN model with quantized neurons will be developed which can eliminate multipliers as well as simplify the digital implementation of nonlinear activation functions. The proposed models will result in a significant improvement in both area and speed of digital implementation of multilayer feedforward neural networks.

Chapter 3

MULTILAYER FEEDFORWARD NEURAL NETWORKS WITH SINGLE TERM POWERS-OF-TWO WEIGHTS

As discussed in previous chapters, in order to alleviate the burden of multiplications in digital hardware implementation of MFNNs, powers-of-two valued connection weights can be used in place of the original continuously valued weights such that the multiplications can be replaced by shift operations. It is no doubt that the format of single term powers-of-two (STPT) would be of the most interest. To be specific, when the STPT format is used, all weights in an MFNN would only be able to take values from the following set W_{stpt}

$$W_{stpt}=\{ \pm 1, \pm 2^{-1}, \ldots , \pm 2^{-M}, 0 \} \qquad (3.1)$$

Where M is the maximum number of bits that may be shifted. It is noted that the above definition constrains the absolute value of weights to be less than or

It is also possible to extend admissible weight values to be sum of two or more terms of powers-of-two, which has been proposed for both digital filters and neural networks [Marchesi et al., 1993]. Although such expansion would make the learning process easier due to the increased number of available weight values, it will substantially weaken the advantages of the powers-of-two technique because of the increased complexity of weight management and the higher number of operations. Thus, in the following, the discussion will concentrate on single term powers-of-two (STPT) format.

Since they are not involved in multiplications, biases of neurons are not necessarily limited to powers-of-two format, they can still be real numbers. As pointed out in Chapter 1, after adding single term powers-of-two constraint to weights, their ability to adapt to various problems is dramatically reduced due to limited choices. Therefore, it may not be adequate to adjust only weights in an MFNN with powers-of-two weights as in [Marchesi et al., 1993], new adjustable parameters must be introduced to provide more degrees of freedom in learning. One of the key factors which have significant impact on the performance of an MFNN is, as mentioned previously, the nonlinear activation function. In this Chapter, the adaptive slope of activation functions will be introduced to enhance the learning capability of the post-quantization MFNNs.

MFNNs with STPT weights consists of three stages. First, the conventional backpropagation algorithm is applied to find the continuous solution (a set of continuous weights) for a given problem; then, quantization is adopted to convert the obtained weights into appropriate STPT values. Finally, adaptation of the slope of the activation function will, in addition to adjustment of weights and biases, be employed to fine-tune the post-quantization network to the pre-determined error level based on the method presented below.

3.1 Adaptation of activation functions in MFNNs

Consider a multilayer feedforward neural network with the following form of sigmoid activation function

$$F(x) = \frac{1 - e^{-\alpha x}}{1 + e^{-\alpha x}} \tag{3.2}$$

This nonlinear activation function is a key factor in determining the performance of a neural network. It can also be seen from Eq.(2.11)-(2.13) that the activation function and its derivative play a very important role in the process of weight updates and, as a result, any change in the activation function will affect the learning process. Therefore, a proper choice of the shape of activation function can result in a better adjustment of weights and also affect the input-output relationship of the network. Shown in Fig. 3.1 are sigmoid

activation function may be controlled by the slope of the function and the slope of the activation function is controlled by the parameter α. This property was used in [Kruschke and Movellan, 1991] to speed up the BP learning process and improve generalization capability.

The idea of gradient descent method can be extended to adaptation of parameter α, i.e., the change in α will be in the opposite direction of the partial derivative of the squared output error of the network with respect to α. Hence, the change in $\alpha_j^{[h]}$, the parameter α of neuron j at layer h, due to the presentation of pattern k can be expressed as

$$\Delta_k \alpha_j^{[h]} = -\epsilon_\alpha \frac{\partial \theta_k}{\partial \alpha_j^{[h]}} \tag{3.3}$$

where ϵ_α is a step size for α update and e_k has the same definition as in section 2.2. By applying the chain rule (see Appendix B), the following relationship are obtained

$$\Delta_k \alpha_j^{[h]} = \epsilon_\alpha F_\alpha'(z_{jk}^{[h]}, \alpha_j^{[h]}) \sum_{i=1}^{N_{h+1}} \delta_{ik}^{[h+1]} w_{ji}^{[h+1]} \qquad \text{for } h<L \tag{3.4}$$

and

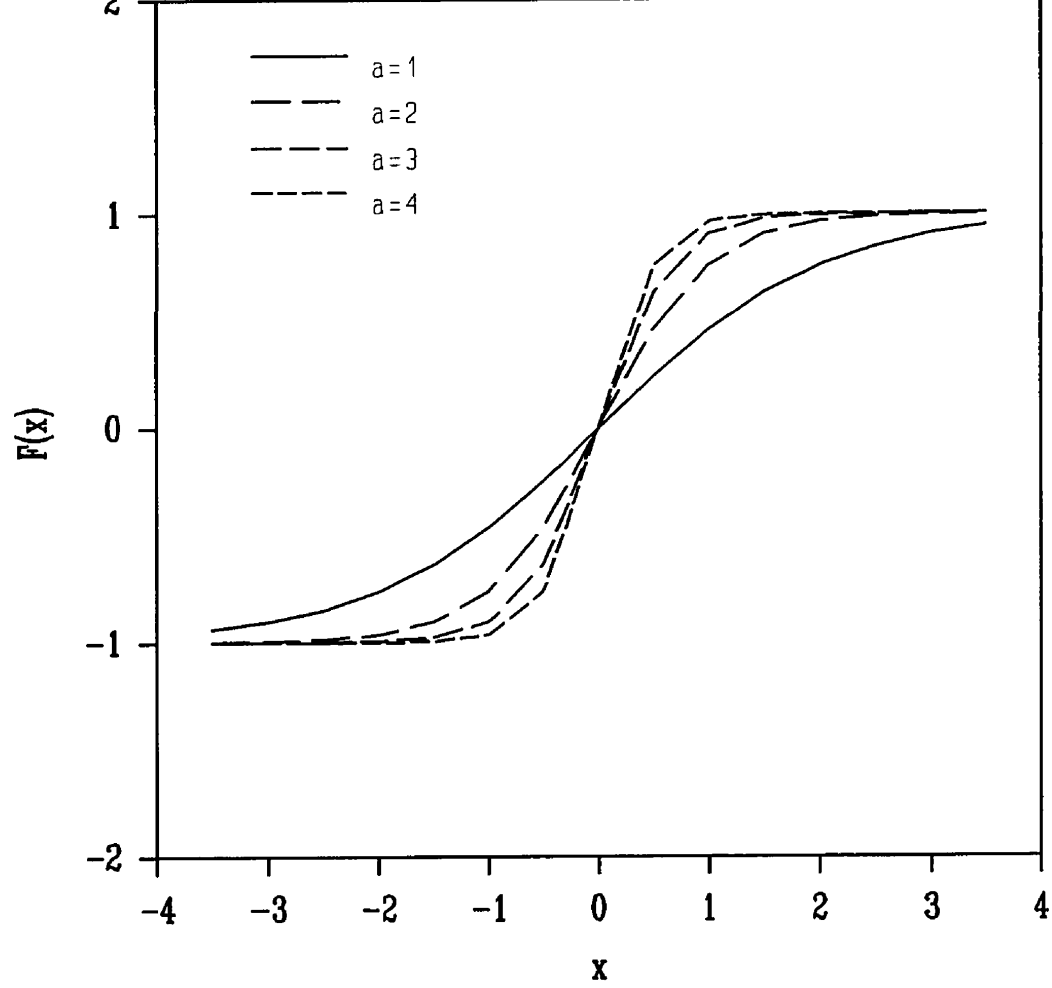


Figure 3.1 Sigmoid Functions with Different α

$$\Delta_k \alpha_j^{[l]} = 2\epsilon_\alpha (t_{jk} - y_{jk}^{[l]}) F_\alpha (z_{jk}^{[l]}, \alpha_j^{[l]}) \tag{3.5}$$

where $F'_\alpha(z,\alpha)$ is the partial derivative of the activation function F() with respect to α, i.e.,

$$F_\alpha(z,\alpha) = \frac{\partial E}{\partial \alpha}$$

and $\delta_{ik}^{[h+1]}$, L, N_{h+1}, $z_{jk}^{[h]}$, t_{jk}, and $y_{jk}^{[L]}$ are all defined in the same way as in Chapter 2.

It is also possible and usually helpful to include a momentum term to the update equation of α. When taking all training patterns into account, the $\alpha_j^{[h]}$ will be updated as

$$\alpha_j^{[h]}(t+1) = \alpha_j^{[h]}(t) + \sum_{k=1}^{K} \Delta_k \alpha_j^{[h]} + \mu_\alpha \{\alpha_j^{[h]}(t) - \alpha_j^{[h]}(t-1)\} \tag{3.7}$$

This scheme has been proved to be very effective in improving the learning speed of MFNNs. In this Chapter, the adaptive slope of the activation function will be used in the design procedure of MFNNs with STPT weights for the purpose of post-quantization network fine-tuning, i.e., adjust the network to compensate the error resulted from weight quantization.

3.2 Design Procedures for MFNNs with STPT Weights

3.2.1 Basic Ideas

Consider a multilayer feedforward neural network as illustrated in Fig.2.1 where the nonlinear activation function applied at the output of each neuron is

network with STPT weights from the set $\{\pm 1, \pm 2^{-1}, \pm 2^{-2}, \ldots, \pm 2^{-M}, 0\}$, where M determines the number of quantization levels of weights. For a given M, there are $2M+3$ distinguished values of weight to choose from.

Given a mapping problem between the input and output spaces of the MFNN and a set of training pattern pairs $\{\mathbf{X}_k, \mathbf{T}_k\}$, the starting point of the design procedure is the solution for the same mapping problem from a conventional MFNN with continuous weights, using BP as the learning algorithm. Then, at the next stage, continuous weights will be transformed into single term powers-of-two weights and activation functions are scaled accordingly to accommodate such quantization. Finally, the slopes of activation functions will be adjusted based on the algorithm described in the previous section to compensate any increase in the output error of the network caused by quantization. Since the bias of each neuron, $b_j^{[h]}$, is not involved in multiplications, it can remain continuous.

Before the quantization of weights, it is necessary to introduce a normalization process such that all weights will be in the interval of [-1, +1] because the set of quantization levels is defined by (3.1). Consider a particular neuron *j* at layer *h*, where all connections going into the neuron are denoted by weights $w_{ij}^{[h]}$, $i = 1, \ldots, N_{h-1}$. Define

$$w_{j\text{-max}} = \max_{i}(|w_{ij}| \quad i = 1,2,\ldots,n_{h-1}) \qquad (3.8)$$

Then, the normalization will be carried out by dividing weights $w_{ij}^{[h]}$ by $w_{j\text{-max}}^{[h]}$ as follows

$$w'^{[h]}_{ij} = \frac{w_{ij}^{[h]}}{w_{j\text{-max}}^{[h]}} \qquad (3.9)$$

$w'^{[h]}_{ij}$ now belongs to the interval [-1, +1]. However, it can be seen that the input-output mapping relationship of the network will be changed if normalized weights are used in the network without any other appropriate adjustment of the network. By examining (2.3), (2.4) and (3.2), it is found that it is possible to compensate the normalization of weights by scaling up the parameter of α accordingly. That means if the weights are normalized by their maximum value $w_{j\text{-max}}$, then adjusting the parameter α of the sigmoid activation function as follows will keep the network mapping relationship unchanged.

$$\alpha'^{[h]}_{j} = \alpha_j^{[h]} w_{j\text{-max}}^{[h]} \qquad (3.10)$$

Now, we can quantize the normalized weights to STPT format. The criterion used here is to round a weight to its nearest STPT value selected from the set W_{stpt}. This scheme can be described as follows

$$Q(w) = sgn(w)\begin{cases} 2^{-m} & C_m < |w| < C_{m-1} \end{cases}$$ (3.11)

where sgn(w) denotes the sign of w and

$$C_m = (\frac{3}{4})\,2^{-m} \qquad for\ m = 0,1,\ldots,M \tag{3.12}$$

A quantization curve based on the above definition when M = 4 is shown in Fig.3.2.

Usually, there will be an increase in the output error of the network due to weight quantization. Therefore, more adjustment to the network is necessary in order to bring this error down to a predetermined level. In this proposed model, the method of adapting the slope of the activation functions, as described in Section 3.2, will be used for this purpose. At this point, since weights have already been quantized to discrete values and are not able to make arbitrary changes (required by BP algorithm), there will be an update in weights only when such an update can result in a reduced output error. At the same time, the bias of each neuron can still be adjusted by using BP algorithm because they are not involved in any multiplication and may remain to be continuous.

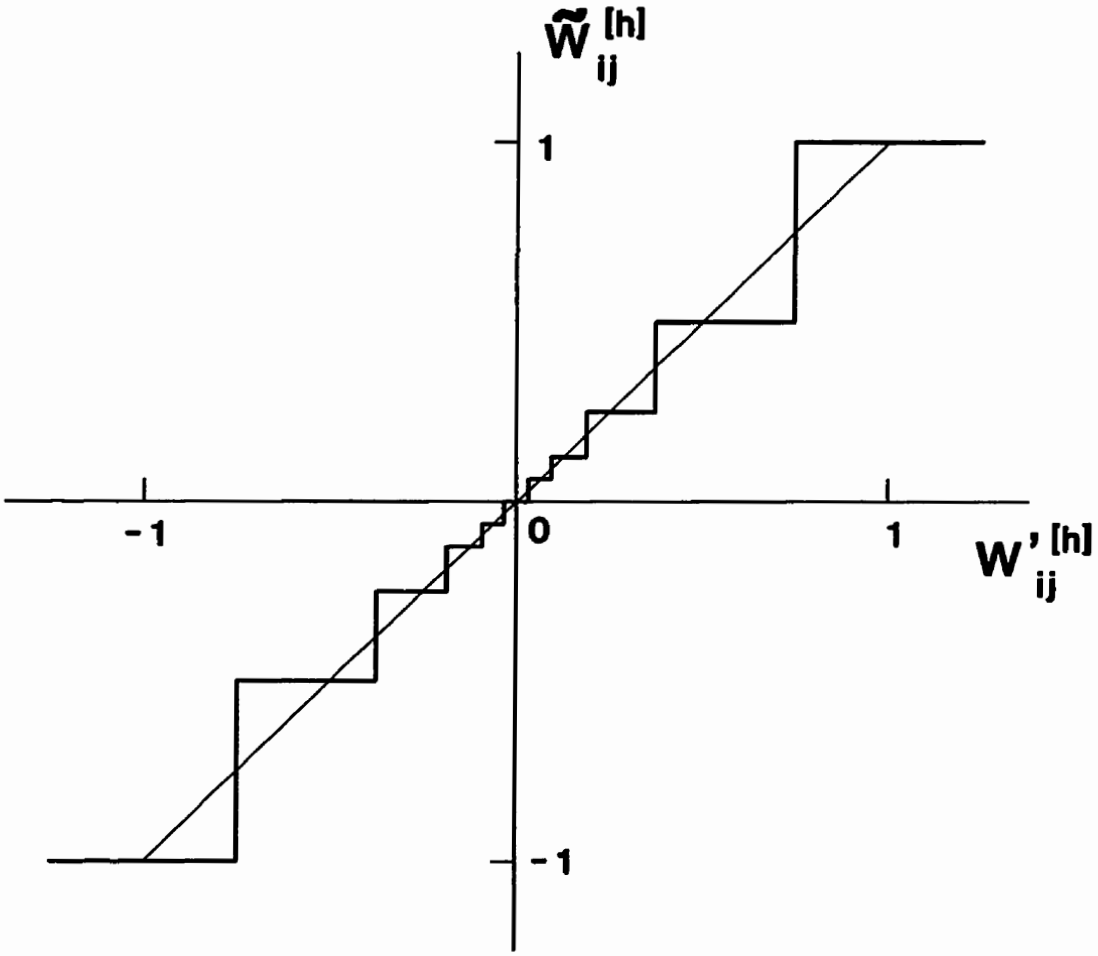


Figure 3.2 Weight Quantization Curve When M = 4

3.2.2 Design Algorithm

Based on the idea presented in Section 3.2.1, a procedure for design of

developed and is illustrated in the following.

Step 1: Set $\alpha_j^{[h]} = \alpha_0$ ($j = 1, 2, \ldots, N_h$; and $h = 1, 2, \ldots, L$), where $\alpha_j^{[h]}$ is related to the sigmoid activation function applied to neuron *j* at layer *h* of an *L*-layer MFNN.

Step 2: Starting with a set of weights and biases with small random values, train the network using the conventional backpropagation algorithm to obtain a continuous solution for the given problem, i.e., to obtain a set of continuous weights and biases which can achieve

$$\sum_{k=1}^{K} e_k < E_0 \tag{3.13}$$

where e_k is as defined in (2.8), *K* is the number of pattern pairs in the training set, and E_0 is a predetermined error level.

Step 3: Find the maximum absolute value, $w_{j\text{-max}}^{[h]}$, among weights $w_{ij}^{[h]}$ ($i = 1, 2, \ldots, N_{h-1}$)

$$w_{j\text{-max}}^{[h]} = \max_i \{ |w_{ij}^{[h]}| \;/\; i = 1, \ldots, N_{h-1} \} \tag{3.14}$$

for $j = 1, \ldots, N_h$ and $h = 1, \ldots, L$.

the interval of [-1, +1]

$$w'^{[h]}_{ij} = \frac{w^{[h]}_{ij}}{w^{[h]}_{j\text{-max}}} \tag{3.15}$$

for $i = 1, \ldots, N_{h-1}$, $j = 1, \ldots, N_h$ and $h = 1, \ldots, L$.

Also scale biases $\Theta_j^{[h]}$ by the same factor $w_{j\text{-max}}^{[h]}$

$$\theta^{[h]}_j = \frac{\theta^{[h]}_j}{w^{[h]}_{j\text{-max}}} \tag{3.16}$$

for $j = 1, \ldots, N_h$ and $h = 1, \ldots, L$.

Step 5: Adjust parameter $\alpha_j^{[h]}$ accordingly as follows

$$\alpha^{[h]}_j = w^{[h]}_{j\text{-max}}\, \alpha^{[h]}_j \tag{3.17}$$

for $j = 1, \ldots, N_h$ and $h = 1, \ldots, L$.

Step 6: Quantize normalized weights $w'^{[h]}_{ij}$ to single term powers-of-two weights $w^{*[h]}_{ij}$

$$w^{*[h]}_{ij} = Q(w'^{[h]}_{ij}) \tag{3.18}$$

for $i = 1, \ldots, N_{h-1}$, $j = 1, \ldots, N_h$ and $h = 1, \ldots, L$. Where the function

Step 7: Substitute current STPT weights $w^{*}_{ij}{}^{[h]}$ and new values of $\alpha_j^{[h]}$ into the network. Calculate the squared output error over all training pattern pairs as

$$TSE=\sum_{k=1}^{K} e_k \tag{3.19}$$

If $TSE < E_0$, stop; otherwise, proceed to Step 8.

Step 8: Calculate $\Delta w^{*}ij^{[h]}$, $\Delta b_j^{[h]}$, and $\Delta\alpha_j^{[h]}$, which are changes in $w^{*}_{ij}{}^{[h]}$, $b_j^{[h]}$, and $\alpha_j^{[h]}$ respectively, using the relationships developed in Sections 2.2, 2.3, and 3.1.

Step 9: Update weights $w^{*}_{ij}{}^{[h]}$ with changes Δw obtained in Step 8 and then quantize them to STPT values as in Step 6. If this update results in a reduced TSE, accept the new weights; otherwise, discard the changes and keep previous weights.

Step 10: Update parameters $b_j^{[h]}$ and $\alpha_j^{[h]}$ with changes $\Delta b_j^{[h]}$ and $\Delta\alpha_j^{[h]}$ obtained in Step 8.

Step 11: Go to Step 7.

It needs to be pointed out that, as in the original BP algorithm, convergence cannot be guaranteed. If convergence cannot be achieved, it may be necessary to restart the algorithm from the beginning with a new and

network topology, adding more neurons and/or layers to the existing network. However, as shown in the next section, in most cases MFNNs with STPT weights can reach convergence for the same problem as original MFNNs, without increasing the number of neurons or layers.

3.3 Simulation Results

Simulations have been conducted to verify the effectiveness of the proposed design algorithm. The first example is a simple but very important benchmark problem, XOR.

3.3.1 A Benchmark Problem

XOR (exclusive OR) problem has been considered as a benchmark problem in neural network history. It is of great importance to test the mapping ability of neural networks. XOR was first cited by Minsky [Minsky and Papert, 1969] in 1969 to criticize the capability of neural networks, which caused an interruption in neural network research for about ten years. In 1986, in their famous books on Parallel Distributed Processing, Rumelhart *et al* have demonstrated that MFNNs are capable of such mapping, which marked the revival of neural network research.

input		output
0	0	0
0	1	1
1	0	1
1	1	0

According to [Rumelhart et al., 1986], the smallest MFNNs for the XOR problem consists of two input neurons, one hidden layer with two hidden units, and one output neuron. MFNNs with STPT weights have demonstrated that they are capable of solving this problem without increasing the size of the network from the smallest topology. Since this is a binary (0/1) input-output mapping problem, the binary form of sigmoid activation function given by (3.20) is used at each neuron. Other parameters used in the simulations are listed in Table 3.1.

$$F(x) = \frac{1}{1 + e^{-\alpha x}} \tag{3.20}$$

Parameters	Symbols	Values
Learning rate of weights	ϵ	0.5
Range of initial weights	W_0	[-1.0, 1.0]
Learning rate of biases	ϵ_b	0.1
Range of initial biases	B_0	[-0.1, 0.1]
Step size of slope changes	ϵ_a	0.15
Momentum of slope changes	μ_a	0.05
Quantization levels of weights	M	4
Error level for training	E_0	0.1

MFNNs with STPT weights have shown very good performance for the XOR problem, i.e., convergence was always reached within a limited number of epochs. One typical example is given below:

NO. of epochs needed in stage 1: 730

NO. of epochs needed in stage 2: 9

Weights:

$w^{[1]}_{11} = 1.00000$ $w^{[1]}_{12} = 1.00000$

$w^{[1]}_{21} = 1.00000$ $w^{[1]}_{22} = 1.00000$

$w^{[2]}_{11} = 1.00000$

$w^{[2]}_{12} = -1.00000$

Slopes:

$a^{[1]}_1 = 5.82774$ $a^{[1]}_2 = 1.93203$

$a^{[2]}_1 = 9.00515$

BIASES:

$\Theta^{[2]}_1 = -.34642$

The simulation results of the XOR problem impliy that MFNNs with STPT weights are able to achieve the same mapping capabilities as the conventional MFNNs and the use of powers-of-two weights does not necessarily mean an increase in the size (number of neurons or layers) of the network.

3.3.2 More simulations

More simulations have been carried out by using 10 numerals, each represented by a 10x10 pixel matrix as shown in Fig.3.3. The corresponding targets were given below each pattern. The MFNN used in the simulations had one input layer with 100 units, one output layer with 4 neurons, and one or more hidden layers with various numbers of hidden neurons. Different combinations of the number of hidden layers, the number of hidden neurons, and the number of weights quantization levels were used in simulations to test the performance of the proposed design procedures.

Two aspects of performance, i.e., the convergence and generalization properties of the algorithm, have been observed in simulations. For each topology of the network and the number of quantization levels, the network

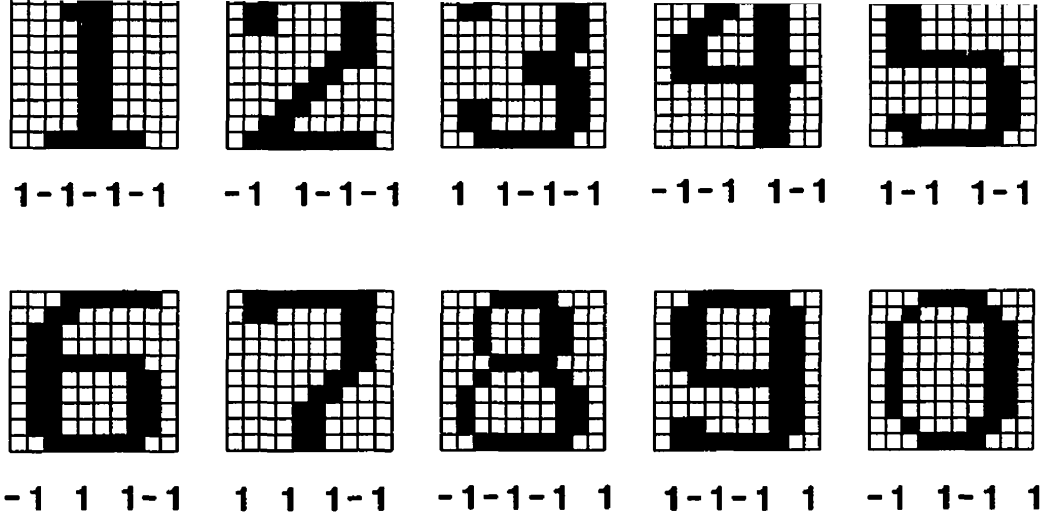


Figure 3.3 10 numeral training patterns

was first trained with the given 10 pattern pairs to obtain both continuous and quantized solutions, i.e., weights; then a set of noisy patterns (original patterns corrupted by noise) was fed to both continuous-weight network and the powers-of-two-weight network to test the generalization abilities. A bipolar form of inputs and outputs was used. Noisy patterns were constructed by randomly inverting a percentage of total elements in training patterns. The generalization performance was measured by the recall accuracy (the percentage of correct recalls) which was obtained by feeding 100 noisy versions of each training pattern to the network and taking the average.

Table 3.2 Convergence Speed (In Number of Epochs) for CMFNN and STPT MFNN (100 Inputs, 4 Outputs, and 1 Hidden Layer)

No. of Hidden Neurons	CMFNN	STPT MFNN		
		M=2	M=4	M=8
10	61.6	/	55.2	19.6
20	42.8	1366.6	3.8	3.4
40	32.4	34.0	2.0	2.0
60	28.0	9.4	2.0	2.0
80	25.4	13.0	2.0	2.0
100	24.0	2.2	2.0	2.0

Table 3.3 Generalization Capabilities (In Percentage of Correct Recalls) for CMFNN and STPT MFNN (100 Inputs, 4 Outputs, and 1 Hidden Layer)

No. of Hidden Neurons	CMFNN	STPT MFNN		
		M=2	M=4	M=8
10	99.58%	/	98.92%	98.98%
20	99.46%	96.14%	99.16%	99.24%
40	99.46%	98.94%	99.28%	99.34%
60	99.48%	99.08%	99.46%	99.50%
80	99.60%	99.00%	99.40%	99.40%
100	99.52%	99.04%	99.42%	99.42%

Table 3.4 Convergence Speed for Networks with Different Number of Hidden Layers When M=4 (100 Inputs and 4 Outputs)

No. of Hidden Neurons	One Hidden Layer		Two Hidden Layers	
	CMFNN	STPT MFNN	CMFNN	STPT MFNN
10	61.6	55.2	131.2	208.6
20	42.8	3.8	64.6	10.2
40	32.4	2.0	41.0	2.8
60	28.0	2.0	32.0	2.0
80	25.4	2.0	26.4	2.0
100	24.0	2.0	22.6	2.0

Table 3.5 Generalization Capabilities for Networks with Different Number of Hidden Layers When M=4 (100 Inputs and 4 Outputs)

No. of Hidden Neurons	One Hidden Layer		Two Hidden Layers	
	CMFNN	STPT MFNN	CMFNN	STPT MFNN
10	99.58%	98.92%	99.14%	98.42%
20	99.46%	99.16%	99.52%	99.30%
40	99.46%	99.28%	99.34%	99.04%
60	99.48%	99.46%	99.44%	99.18%
80	99.60%	99.40%	99.60%	99.36%
100	99.52%	99.42%	99.62%	99.42%

the proposed design algorithm have been confirmed by the simulation results because convergence was reached in almost all runs. It can be seen from the above simulation results that MFNNs with STPT weights can retain similar generalization capability as the conventional MFNNs without an increase in the size of the network. Furthermore, in order to achieve good performance, parameters like the number of hidden neurons and the number of quantization levels should be chosen carefully. Small number of hidden neurons combined with very small number of quantization levels of weights should be avoided. Some redundancy in the topology of the network is recommended in order to reduce the number of quantization levels of weights and consequently to reduce the number of bits to be shifted in hardware implementation.

3.4 Comparison with Existing Models

With the introduction of adaptive slopes of activation functions for post-quantization network fine-tuning, our above proposed algorithm has shown to be very effective in the STPT-network error reduction. It has significant advantages in convergence speed and the magnitude of error that can be reduced when compared with the weight-adjust-only algorithm, e.g., the one used in [Marchesi et al., 1993]. The comparison of the two algorithms over the performance of their post-quantization network tuning capabilities are shown

of hidden neurons varies from 10 to 60.

In each diagram, curve "weight-slope" represents the result using the algorithm proposed in this chapter, i.e., adaptation of weights and slope of activation functions for post-quantization network tuning, while curve "weight-only" refers to the result based on the algorithm of adjusting weights only with adaptive learning rate parameter as proposed in [Marchesi et al., 1993]. It can be seen that in all cases our proposed algorithm is more capable of post-quantization network error reduction.

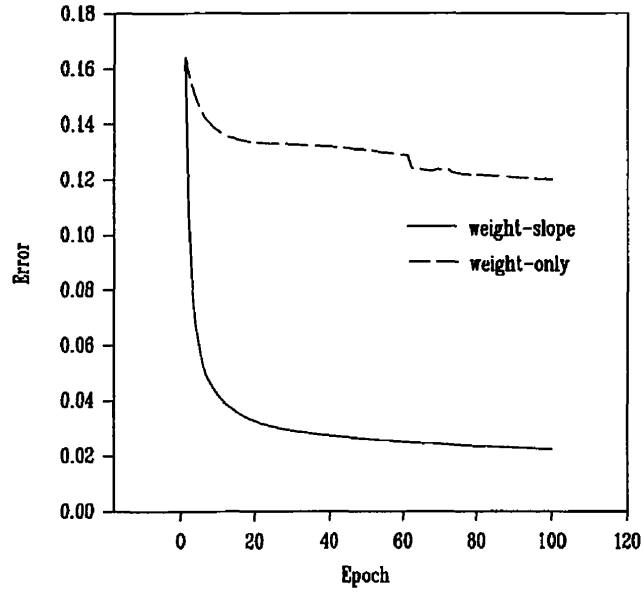


Figure 3.4 Error curve when $N_h = 10$

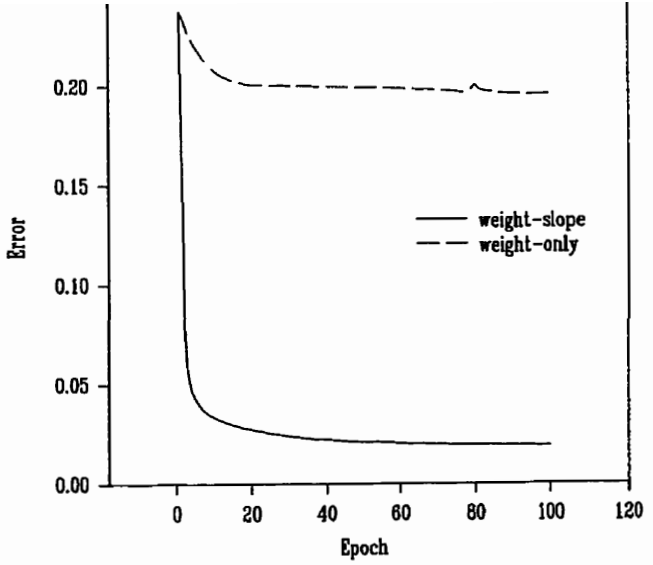


Figure 3.5 Error curve when $N_h = 20$

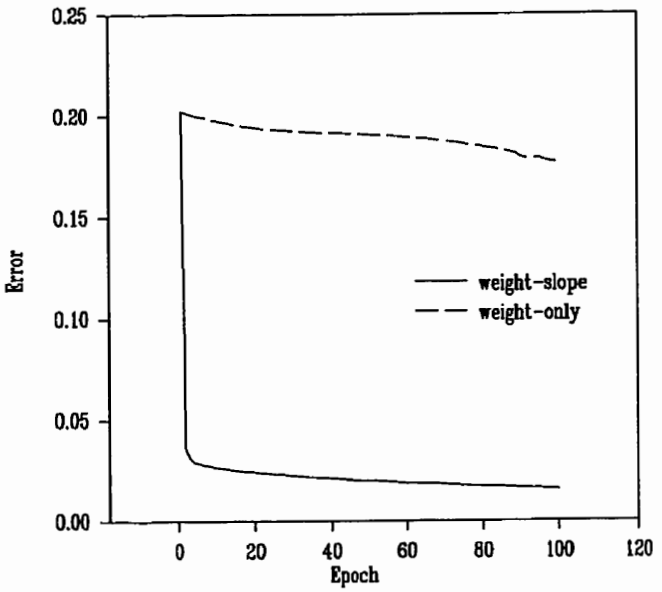


Figure 3.6 Error curve when $N_h = 40$

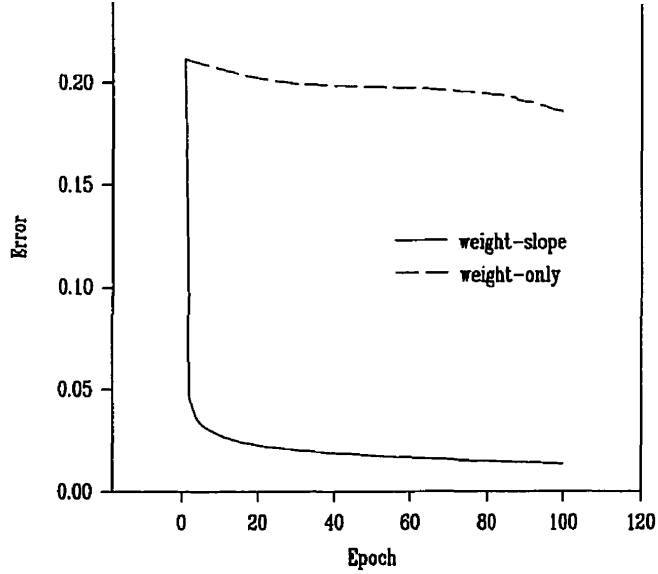


Figure 3.7 Error curve when $N_h = 60$

3.5 Advantages for hardware implementation

The basic operation of MFNNs is to pass a weighted sum of inputs through a non-linear activation function. Therefore, the digital hardware implementation of MFNNs will consist of several major functional blocks, including multiplications, summations, and calculation of non-linear functions. Among them, summations can be implemented by using adders and accumulators, while non-linear calculations are usually done by look-up tables. However, the multiplications between inputs and weights are not favored by digital technology since the multipliers required for these operations are slow

in a digital VLSI implementation.

This computational burden in digital implementation can be eliminated if STPT weights are used in MFNNs as proposed previously in this chapter. There will be significant gain in operation speed and saving in silicon area when multiplications being replaced by shift operations. A shift operation can easily be implemented using either MUX's or simple combinational logic. A block diagram illustrating the operation of a shifter is shown in Fig.3.8. The corresponding symbol of the shifter is drawn in Fig.3.9, where "A" is the input vector, "CT" is the control vector which determines the number of bits to be shifted, and "Z" is the output vector, which is a shifted version of the input vector "A". To be fitted into a neuron's operation in MFNNs with STPT weights, the pin "A" is connected to a particular input to the neuron, while "CT" is mapped to the corresponding STPT weight. Considering that STPT weights are all with negative powers, the shifter has shift-right operations only. The operation of the shifter can further be illustrated as follows. Assuming M=4, the possible STPT choices are 2^0, 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, 0 and corresponding control vectors of the shifter are 10000, 01000, 00100, 00010, 00001, and 00000, respectively. If input "A" is an 8 bit vector, $A_7...A_0$, the operation can be described by Table 3.6. A VHDL description of such an operation with STPT parameter M=4 can be found in Table D.1 of Appendix D.

Table 3.6 Description of the operation of the shifter

CT	A	Z
00000	$A_7 \ldots A_0$	000000000000
10000	$A_7 \ldots A_0$	$A_7 \ldots A_0 0000$
01000	$A_7 \ldots A_0$	$A_7 A_7 \ldots A_0 000$
00100	$A_7 \ldots A_0$	$A_7 A_7 A_7 \ldots A_0 00$
00010	$A_7 \ldots A_0$	$A_7 A_7 A_7 A_7 \ldots A_0 0$
00001	$A_7 \ldots A_0$	$A_7 A_7 A_7 A_7 A_7 \ldots A_0$

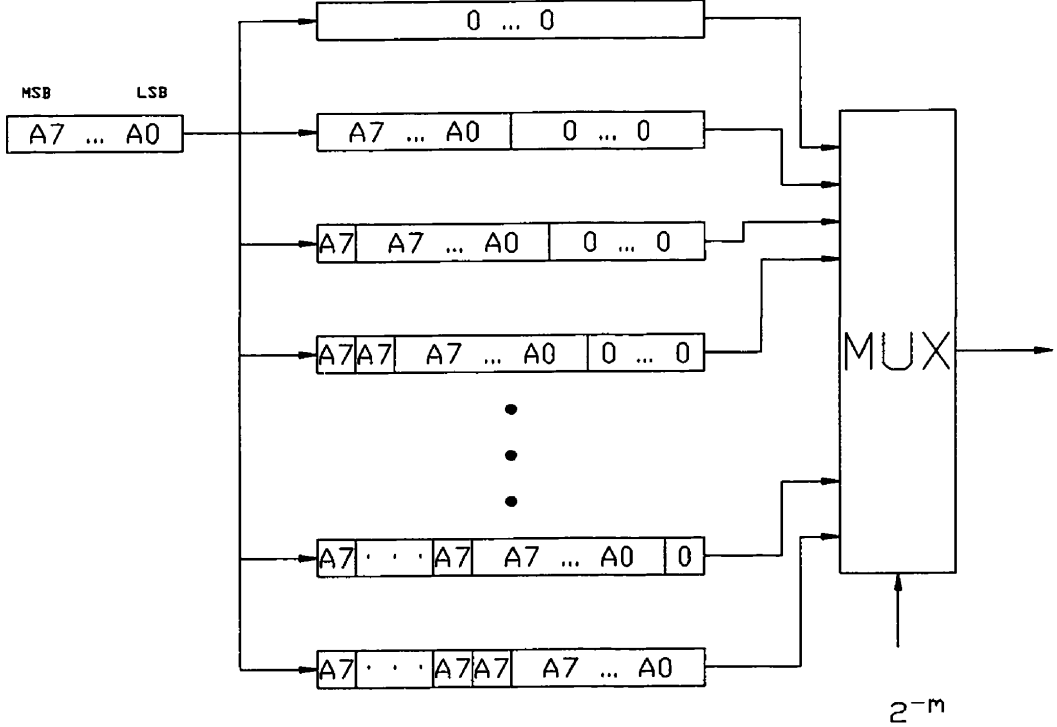


Figure 3.8 Illustration of the shift operation

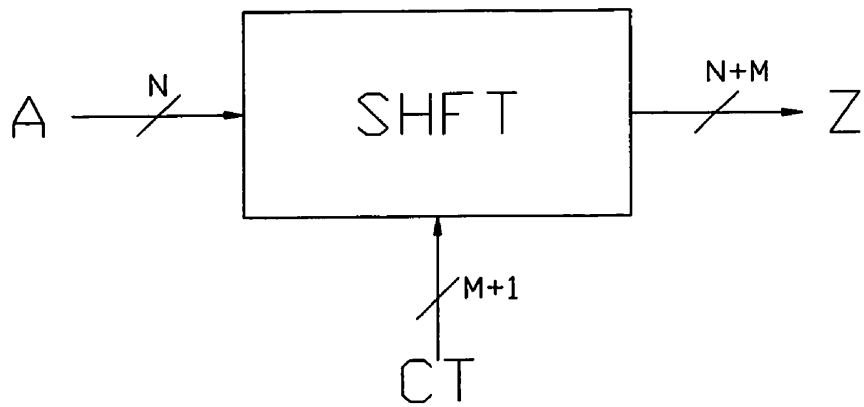


Figure 3.9 A shifter

design has an area of 474 design units (158 equivalent gates) with a maximum delay of 1.35 ns in LSI Logic 0.6 um 3.3V CMOS 600K ASIC technology.

If we consider an 8x8 2's complement multiplication with carry-save array in the same technology, the multiplier will require 2525 design units (842 gates) and have a maximum delay of 11.30 ns.

Table 3.7 illustrates the advantages of the MFNN model of STPT weights over the original MFNN model for digital hardware implementation of the multiplication in the weighted sum calculations. The technology used here is LSI Logic Corporation 0.6 um 3.3V CMOS 600K ASIC technology.

Table 3.7 Hardware advantage of MFNN with STPT weights

	MFNN	STPT-MFNN
Calculation	$z_j^{[h]} * w_{ij}^{[h]}$	$z_j^{[h]} * w_{ij}^{[h]}$
Implementation	multiplier (8x8)	shifter
Area (# gates)	842	158
Delay (ns)	11.30	1.35

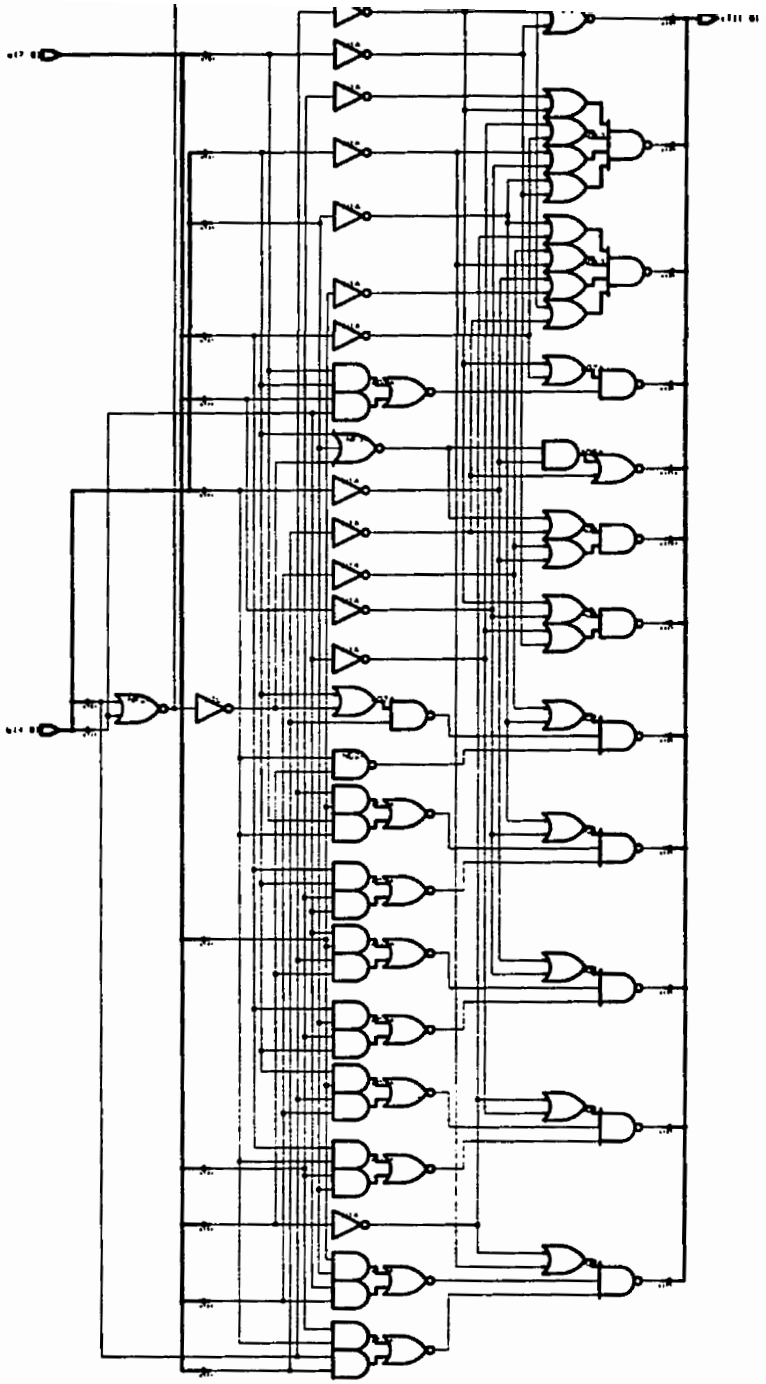


Figure 3.10 Schematic of the shifter used in MFNN with STPT weights

significantly by replacing multipliers with shifters in the MFNN design.

As for the nonlinear activation functions, they are usually implemented by using look-up tables. In the original MFNN models, every neuron in the network has the same activation function. For the proposed model of MFNN with STPT weights, due to the adjustment of the slope of activation functions, each neuron in the network will have different parameter for activation function and result in a different look-up table. In the first glance, it seems that the implementation of activation function in the new model will be more complicated than in the original model where a single global look-up table may be used.

However, one of the most attractive features of artificial neural networks is the parallel distributed computation. In order to realize parallel processing, each neuron will need a local look-up table for its activation function instead of a global one. Based on this consideration, the proposed model will have no extra hardware requirements by implementing a different activation function at each neuron.

3.6 Concluding Remarks

The model of multilayer feedforward neural networks with single-term

algorithm featuring adaptive slope of activation functions was developed and corresponding design procedure was established. Due to the use of STPT weights, the multiplications required in the weighted sum operations were able to be replaced by shift operations, which resulted in a substantial improvement in both silicon area and operation speed in digital VLSI implementation of multilayer feedforward neural networks. Meanwhile, the MFNNs with STPT are capable of achieving almost the same generalization performance as the original multilayer feedforward networks without increasing the network sizes. The results presented here demonstrated the feasibility of the proposed model in multilayer feedforward neural network applications.

Chapter 4

MULTILAYER FEEDFORWARD NEURAL NETWORKS WITH QUANTIZED NEURONS

In the previous chapter, an MFNN model using STPT weights has been proposed to alleviate the computational burden of multiplication. By using that model, the multipliers will be replaced by shift registers in digital hardware implementation. In this chapter, a new model, which can solve the same problem but through a different approach, will be developed.

4.1 Introduction

Under normal circumstances, the outputs of activation functions in an MFNN and connection weights obtained by using the backpropagation algorithm are continuously valued such that multiplications are inevitable in the calculation of weighted sums, the basic operations involved in MFNNs. This implies that the current architecture of MFNNs is not suitable for digital implementation. Some research activities have been invoked in view of this

STPT weights has been proposed to deal with this problem.

Actually, the two factors involved in a typical multiplication in MFNNs are connection weight and the output of a related neuron. If either of them takes the form of powers of two, then the multiplication is able to be replaced by a shift operation. Based on this observation, a new model of MFNNs using quantized neurons will be proposed in this Chapter. The outputs of such quantized neurons can take only powers-of-two values so that multiplication operations can also be avoided even though weights are still continuous ones. Both strategies of using powers of two weights and quantized neurons in MFNNs can alleviate the computational burden of multiplications; however, quantized neurons offer a number of advantages over powers of two weights. First, using quantized neurons makes the realization of activation functions (usually by look-up-table technique) much easier. Moreover, continuous weights are used with quantized neurons such that the network will have more freedom to be adapted to diverse problems than in the case of using powers of two weights.

The remaining part of this Chapter is organized as follows. Section 4.2 is dedicated to the quantized neuron model and the modified learning algorithm suitable for MFNNs with quantized neurons. In Section 4.3, the detailed

neurons is described. The mapping capability of MFNNs with quantized neurons will be studied in Section 4.4. Simulation results will be provided in Section 4.5. Finally, Section 4.6 is dedicated to hardware implementation.

4.2 Quantized Neurons

As mentioned before, the purpose of introducing quantized neurons is to avoid multiplications by forcing their outputs to be of powers of two values. Thus, the most important feature of a quantized neuron is its output form. In this model, single term powers of two format will be used, i.e., a quantized neuron can generate only outputs from the following set

$$\{ \pm 1, \pm 2^{-1}, \pm 2^{-2}, \ldots, \pm 2^{-M}, 0 \} \tag{4.1}$$

where M determines the number of quantization levels.

Except for its output format, a quantized neuron operates in a similar manner as an ordinary neuron, i.e., it takes a weighted sum and passes it through a nonlinear activation function. The powers of two output format can be realized by adopting a multi-step function as its activation function which can be defined as:

$$G(x) = sgn(x) \begin{cases} 1 & |x| \geq C_1 \\ 2^{-m} & C_{m+1} \leq |x| \leq C_m \\ 0 & |x| < C_{M+1} \end{cases} \tag{4.2}$$

quantized neuron is depicted in Fig.4.1. Since the inputs to this neuron are outputs of other quantized neurons, the operation of taking the weighted sum becomes much easier and no multipliers are needed.

Although a neuron with hardlimit activation function can be considered as a special case of quantized neurons with $M=0$, quantized neurons provide a more generalized definition of neurons with discrete outputs. Since quantized neurons have more output levels than hardlimit function, the network using quantized neurons will have more flexibility in adapting to various problems and will be easier to train.

Although multiplications can be avoided by using quantized neurons in MFNNs, the training of such networks is another problem. Since the activation function of a quantized neuron is a multi-step function, the derivatives of such a function are either zero or undefined. Therefore, the original form of the backpropagation algorithm can not be applied directly. In the following, a modification to the backpropagation algorithm will be proposed to make it suitable for training MFNNs with quantized neurons.

The major obstacle preventing the original BP algorithm from being applicable to MFNNs with quantized neurons is that there is no appropriately

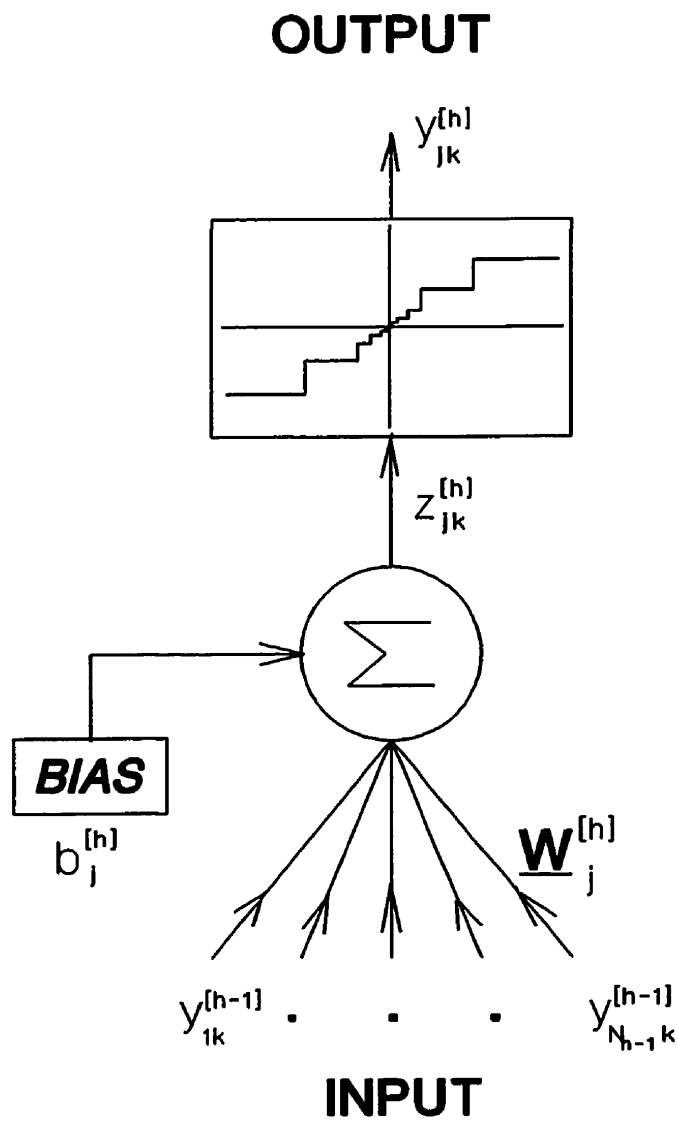


Figure 4.1 A Quantized Neuron

learning will occur. To overcome this difficulty, an appropriate nonzero derivative will be assigned to each interval (C_m, C_{m-1}) of the activation function (the function is constant within the interval). Consider the sigmoid function given as

$$F(x) = g\frac{1-e^{-\alpha x}}{1+e^{-\alpha x}} \tag{4.3}$$

Where $g \geq 1.0$ is a gain factor. This F(x), which can be used as activation function for ordinary MFNNs, is depicted in Fig.4.2. The multi-step activation function can be obtained by the following procedures.

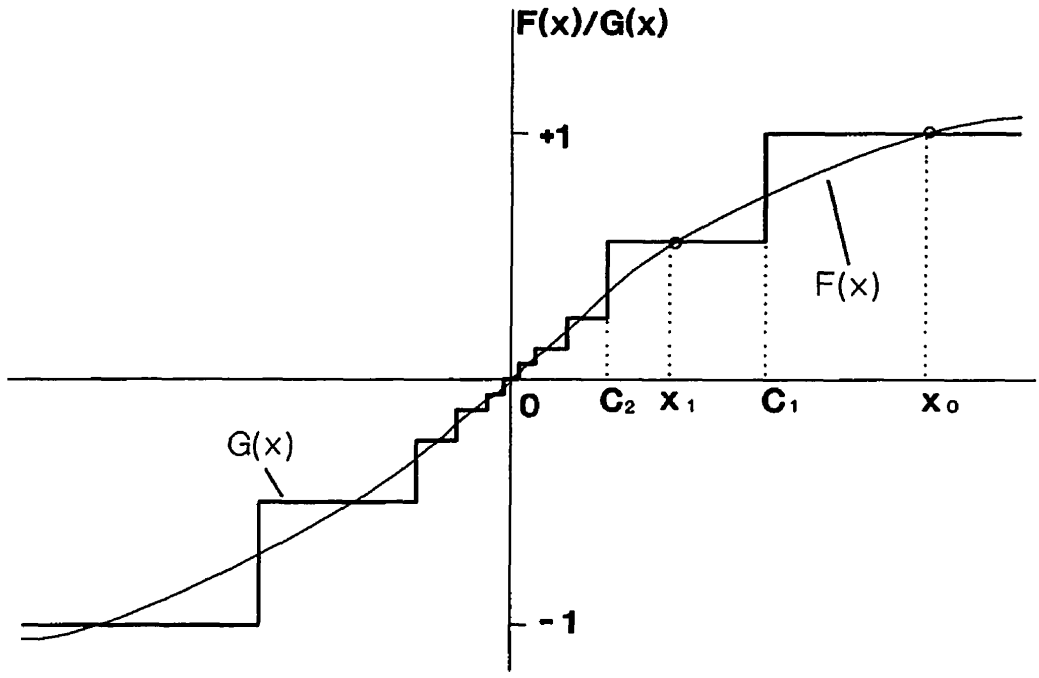


Figure 4.2 Original and Quantized Activation Functions

they range from -1 to +1, each level will have an intersection with F(x) defined above. These intersections can be obtained by solving the following set of simultaneous equations

$$\begin{aligned} y &= g\frac{1-e^{-\alpha x}}{1+e^{-\alpha x}} \\ y &= 2^{-m} \qquad m=0,1,\ldots M \end{aligned} \tag{4.4}$$

Denoting x coordinates corresponding to these intersections as x_0, x_1, ... x_M, then, the extreme points of each interval of G(x) can be defined as

$$\begin{aligned} C_m &= \frac{x_m + x_{m-1}}{2} \qquad m=1,2,\ldots,M \\ C_{M+1} &= \frac{x_M}{2} \end{aligned} \tag{4.5}$$

An illustration is provided in Fig.4.2. Furthermore, the derivative of F(x) evaluated at an intersection will be used to approximate the derivative of G(x) within the entire interval corresponding to that intersection, that is

$$G'(x) = \begin{cases} F'(x_0) & |x| \geq C_1 \\ F'(x_m) & C_{m+1} \leq |x| < C_m \\ F'(0) & |x| < C_{M+1} \end{cases} \tag{4.6}$$

This definition settles down the problem with the derivative of the multi-step activation functions. With these modifications, the principle of the

applied to MFNNs with quantized neurons.

4.3 Design Procedures for MFNNs with Quantized Neurons

In this section, the detailed design procedures of using quantized neurons in MFNNs will be presented. The proposed algorithm consists of three basic steps. Firstly, the network is trained by using the standard backpropagation algorithm outlined in Chapter 2. After convergence, all neurons are then replaced by quantized ones, which are introduced in Section 4.2. Finally, weights are adjusted by adopting the modified backpropagation algorithm described in Section 4.2 to reduce the increased output error.

A. Training the Network Using the Standard Backpropagation Algorithm

For a given problem, assuming that the topology of the network has been established, we first initialize all connection weights to small random numbers, then apply the standard backpropagation algorithm to the network until convergence is reached, i.e., the output error falls below a predetermined level E. For bipolar input and output of +1 and -1, the following sigmoid activation function is used

$$F(x) = g\frac{1-e^{-\alpha x}}{1+e^{-\alpha x}} \tag{4.7}$$

order to avoid the possiblility of undesired saturation. Another consideration on the use of g is that intersections of F(x) and the quantized levels of ± 1 are needed as discussed in Section 4.2 in order to obtain non-zero derivatives of G(x). Upon convergence to a predetermined error level E, a network with continuous weights and sigmoid activation functions is obtained which is denoted as Net#1.

B. Replacing Original Neurons with Quantized Neurons

All neurons in Net#1 will be replaced by quantized neurons while the topology of the network and the connection weights among neurons are unchanged. As stated previously, a quantized neuron differs from an original neuron mainly in its activation function which is a multi-step function as defined in (4.2). After the replacement of neurons, all training patterns will be presented to the network and the output error will be checked. If the sum of squared output errors remains below the error level E, i.e.,

$$\sum_{k=1}^{K} e_k \leq E \tag{4.8}$$

the algorithm terminates at this point. The desired MFNN with quantized neurons, denoted as Net#2, is obtained. Otherwise proceed to the next step.

Normally, replacing original neurons with quantized ones will cause an increase in the output error of the network. If this increased output error jumps above the predetermined level E, then it must be brought down by adjusting the parameters of the network. Since all weights are continuous, the modified BP algorithm proposed in the first part of this section can be employed to fulfill such adjustment.

Besides, biases can also be adapted because they are continuous and not involved in any multiplications. The procedures for bias adaptation are similar to those described previously. The weights and biases will be repeatedly updated until the sum of squared output errors falls below the level E, i.e., (4.8) is satisfied. Now the obtained network, denoted as Net#2, is the desired MFNN with quantized neurons.

The above three-stage design method is referred as Scheme 1. Besides, it is also possible to use quantized neurons right from the very beginning of the training process especially when on-chip (on-line) learning is desired. This plan will be referred as Scheme 2. However, it is expected that the network obtained by using Scheme 2 will lose some generalization capability while it will gain speed in training. In section 4.5.2, the simulations results for both schemes will be provided. Because the training is performed off-line here, the

the other hand, a higher generalization capability is what we always want to achieve.

4.4 Mapping Abilities of MFNNs with Quantized Neurons

The conventional multilayer feedforward neural networks have been shown to be universal approximators [Hornik et al., 1989][Hornik, 1991], that means MFNNs with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden neurons are available.

For MFNNs with quantized neurons, the activation functions are quantized and no longer continuous. Consequently, the mapping ability of such networks needs to be examined. In [Hornik, 1991], it was shown that whenever function $F(\bullet)$ is bounded and nonconstant, then, for arbitrary input environment measures μ, multilayer feedforward networks with activation function F() can approximate any function in $L^p(\mu)$ (the space of all functions on R^k such that $\int_{R^k} |f(x)|^p \, d\mu(x) < \infty$) arbitrarily well provided that sufficiently many hidden units are available if closeness is measured by $\rho_{p,\mu}$ as

$$\rho_{p,\mu}(f,g) = \left[\int_{R^k} |f(x) - g(x)|^p d\mu(x)\right]$$

where $1 \leq p \leq \infty$, the most popular choice being $p = 2$, corresponding to mean square error. Since the activation functions used in MFNNs with quantized neurons can meet the bounded and nonconstant condition, MFNNs with quantized neurons are still capable of approximating any discrete mapping. The simulation results presented below will also show that it is not always necessary to use more hidden units in MFNNs with quantized neurons than in conventional MFNNs in order to get the same mapping capabilities.

4.5 Simulation Results

4.5.1 Benchmark Problems

The XOR problem was again applied to MFNNs with quantized neurons. Since the input/output patterns are binary (0/1) form, the activation function will also take binary form which can be defined as

$$F_{0/1}(z) = \frac{F_{\pm 1}(z) + 1}{2} \tag{4.10}$$

Here $F_{0/1}(z)$ remains a multistep function with STPT values because the constant 1/2 can be combined into biases and the factor 1/2 does not change the single term powers of two format of the activation function.

with two neurons, and one output neuron, which is the smallest MFNN that can solve the XOR problem. Quantized neurons are used for the complete training, i.e., there is only one stage instead of two. Simulations have been carried out with different initial conditions and for each simulation convergence reached within limited number of epochs. One example is given below:

Parameters used in training:

Learning rate of weights: 0.5

Range of initial weights: [-1.0, +1.0]

Learning rate of biases: 0.1

Range of initial biases: [-0.1, +0.1]

NO. of quantization levels: M=4

Error level of training: TSEL=0.01

Results:

NO. of epochs needed in training: 187

Weights:

$w^{[1]}_{11} = 2.380167$ $w^{[1]}_{12} = 1.489372$

$w^{[1]}_{21} = 2.388525$ $w^{[1]}_{22} = 1.496423$

$w^{[2]}_{11} = 3.639545$

$w^{[2]}_{12} = -3.674386$

$\Theta^{[1]}_1 = -1.027880$ $\Theta^{[1]}_2 = -1.806875$

$\Theta^{[2]}_1 = -1.105733$

An FPGA design of an MFNN with quantized neurons has been implemented for the XOR problem (see Appendix D for details) using Xilinx 4000 series technology, which verified the feasibility of using quantized neurons in MFNN designs.

Another benchmark problem is the parity problem, in which the output required is 1 if the input pattern contains an odd number of 1s and 0 otherwise. This is a very difficult problem because the most similar patterns (those which differ by a single bit) require different answers.

A four-bit parity was used to test the MFNNs with quantized neurons. The problem can be described as in Table 4.1. The network used in this problem has four inputs, one hidden layer with four hidden neurons, and one output unit, which is the smallest size needed to solve this parity problem with conventional MFNNs. Convergence was reached in all simulation runs. One of them is shown below.

Parameters of training:

Learning rate of weights: 0.01

Learning rate of biases: 0.01

Range of initial biases: ± 0.1

NO. of quantization levels: $M = 4$

Error level of training: $TSEL = 0.0$

Results:

NO. of epochs needed in training: 24620

Weights:

$w^{[1]}_{11} = 0.67983$ $w^{[1]}_{12} = 0.68972$ $w^{[1]}_{13} = 8.98207$ $w^{[1]}_{14} = 1.19062$

$w^{[1]}_{21} = -0.68769$ $w^{[1]}_{22} = -0.67753$ $w^{[1]}_{23} = -9.57031$ $w^{[1]}_{24} = -1.18309$

$w^{[1]}_{31} = 0.68069$ $w^{[1]}_{32} = 0.69107$ $w^{[1]}_{33} = 8.80989$ $w^{[1]}_{34} = 1.19142$

$w^{[1]}_{41} = -0.68304$ $w^{[1]}_{42} = -0.67973$ $w^{[1]}_{43} = -9.28224$ $w^{[1]}_{44} = -1.18275$

$w^{[2]}_{11} = -4.10414$

$w^{[2]}_{21} = -3.39181$

$w^{[2]}_{31} = 4.70325$

$w^{[2]}_{41} = 4.50950$

BIASES:

$\Theta^{[1]}_{1} = -0.32909$ $\Theta^{[1]}_{2} = -0.35629$ $\Theta^{[1]}_{2} = -1.37785$

$\Theta^{[2]}_1 = -2.73303$

Table 4.1 Description of Parity Problem

INPUT	OUTPUT
0000	0
0001	1
0010	1
0011	0
0100	1
0101	0
0110	0
0111	1
1000	1
1001	0
1010	0
1011	1
1100	0
1101	1
1110	1
1111	0

Based on the above simulation results, it can be seen that MFNNs with quantized neurons are able to achieve the same mapping performance as the conventional MFNNs without an increase in network size.

Simulations have been conducted to verify the design procedures proposed above. An example is given below. The training patterns were the 26 letters of the alphabet, each represented by a 10x10 pixel matrix as shown in **Fig. 4.3**. Black pixels correspond a value to 1 and white pixels are assigned the value -1. The targets were bipolar codes as given below each training pattern. The feedforward neural network used in simulations had 100 inputs, 5 outputs, and one or two hidden layers with various number of neurons. Two aspects of behaviour, i.e., the convergence and generalization properties of the algorithm, have been examined in the simulation. For each topology of the network, Scheme 1 of the design algorithms proposed in section 4.3 was applied to obtain Net#1 and Net#2, a set of noisy patterns (original patterns corrupted by noise) was then fed to Net#2 to test the generalization capability. Noisy patterns were constructed by randomly inverting a percentage of total elements in training patterns. In simulations presented in this section, this percentage was ranging from 5% to 20%. Example of noise patterns are shown in Fig.4.4. The recall accuracy was obtained by taking the average results of 100 noisy versions of each of the training patterns used.

Tables 4.2 through 4.5 show the simulation results under different conditions of the number of hidden layers, hidden neurons, and quantization

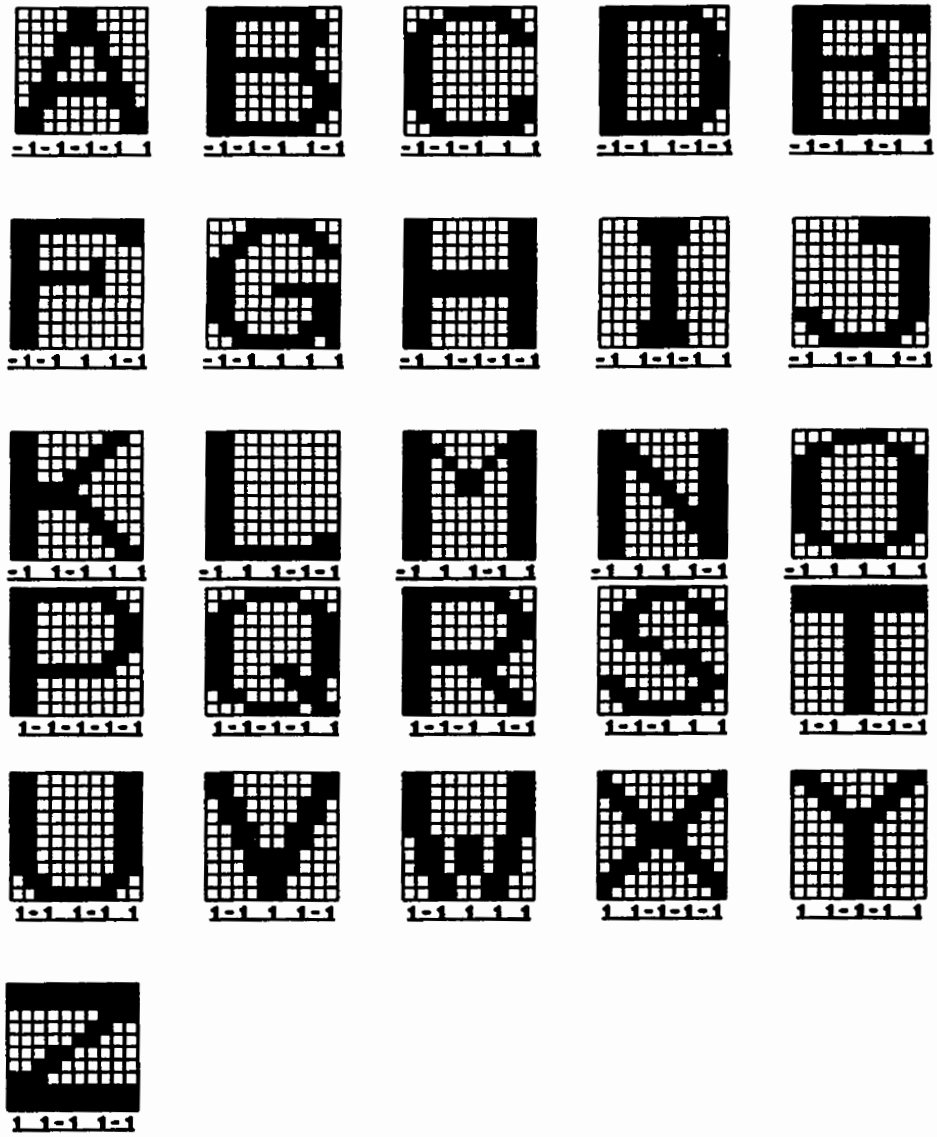


Figure 4.3 Training patterns of the 26 letters of the alphabet

are also given in these tables. For the case of two hidden layers, an identical size for both layers is assumed, i.e., both layers have the same number of neurons. All data given in these tables are averages of five runs of the algorithm, starting with different initial weights which were set as random numbers uniformly distributed in [-0.1, +0.1]. Parameters of F(x) when used to find out C_m's were $g=1.1$ and $\alpha=2.0$. Other parameters used were:

Learning rate for weights: $\epsilon = 0.01$

Predetermined error level: $E = 0.1$.

For the purpose of comparison, also listed in Tables 4.2-4.5 under the column CMFNN are corresponding results of conventional MFNNs using the standard backpropagation algorithm. The parameters of sigmoid activation functions are $g=1.0$ and $\alpha=2.0$.

Table 4.2 Convergence performance in number of training epochs (one hidden layer)

N_H	Scheme 1			Scheme 2		CMFNN
	Net #1	Net #2				
		M=4	M=2	M=4	M=2	
20	453.8	1.0	1.0	35.8	38.2	244.2
40	551.8	2.0	1.6	31.2	32.4	166.4
60	754.8	4.6	3.4	31.8	31.2	141.6
80	1078.8	6.4	5.4	31.0	31.6	146.0
100	1190.6	9.8	8.6	29.4	30.0	126.2

Table 4.3 Generalization capability in percentage of correct recalls (one hidden layer, 5% noise level)

N_H	Scheme 1		Scheme 2		CMFNN
	M=4	M=2	M=4	M=2	
20	94.52	92.76	93.24	90.75	95.70
40	95.13	93.32	93.00	91.71	95.84
60	93.54	91.69	91.41	90.74	94.82
80	92.28	90.53	90.44	88.84	94.09
100	92.44	90.50	90.52	88.27	94.17

Table 4.4 Convergence performance in number of training epochs (two hidden layers)

N_H	Scheme 1			Scheme 2		CMFNN
	Net #1	Net #2				
		M=4	M=2	M=4	M=2	
20	303.4	21.4	17.8	44.6	41.6	198.6
40	314.4	14.4	15.8	36.2	35.0	131.2
60	429.4	15.2	13.8	34.4	33.2	109.2
80	669.0	12.2	13.0	35.4	34.0	86.2
100	827.6	12.8	15.4	36.0	35.6	69.2

Table 4.5 Generalization capability in percentage of correct recalls (two hidden layers, 5% noise level)

N_H	Scheme 1		Scheme 2		CMFNN
	M=4	M=2	M=4	M=2	
20	94.54	93.04	93.53	91.35	95.44
40	94.21	93.03	93.02	91.83	95.39
60	94.09	92.74	93.13	92.10	95.32
80	93.32	92.54	92.78	90.64	95.17
100	93.65	92.16	91.66	90.86	94.30

This test procedure is repeated 100 times at each noise level, in 5% increments ranging from 0% to 20%. At 5% noise, the patterns are still recognizable, as shown by the examples in the middle part of Fig.4.4. At 20% noise, the patterns are not very recognizable, as shown by the examples in the right part of Fig.4.4. The correct recall rate as a function of noise added for MFNNs with 20 hidden neurons is shown in Fig.4.5, where QMFNN refers to the MFNN with quantized neurons and M=4, while the curve CMFNN represents the performance of the conventional MFNN with continuous weights. It can be seen that the MFNN with quantized neurons has no deterioration from original MFNN performance.

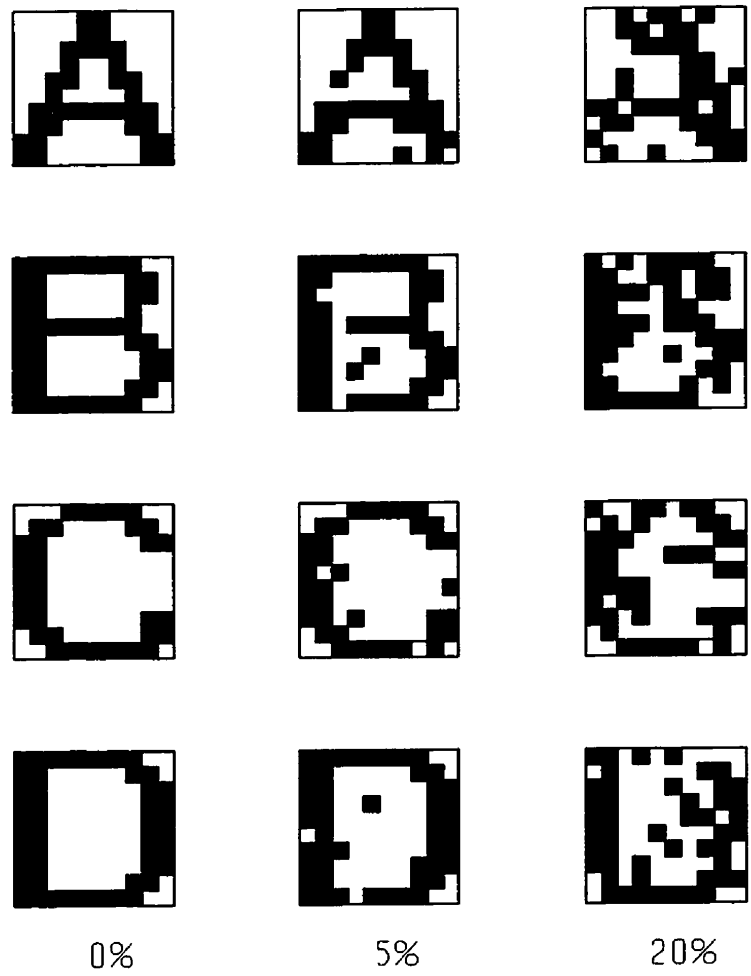


Figure 4.4 Example of noise patterns

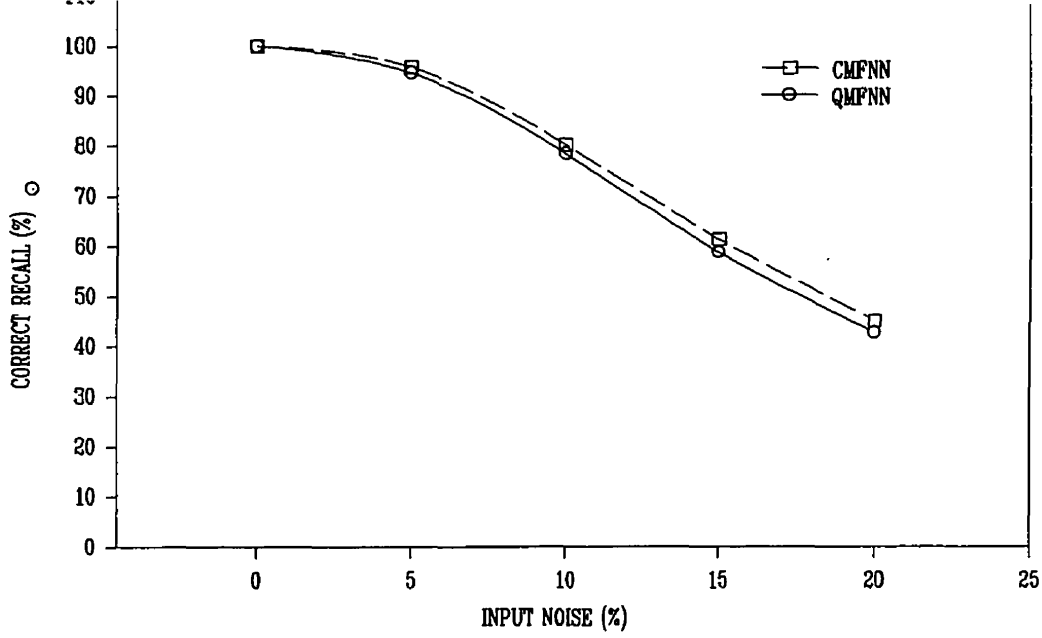


Figure 4.5 Recall accuracy as a function of input noise (20 hidden neurons)

From the simulation results, we can see that, first, convergence was always reached in all runs. This implies the effectiveness of the proposed algorithm and the possibility of implementing feedforward neural networks with quantized neurons. Next, the multilayer feedforward neural networks with quantized neurons can achieve similar recall accuracy as the conventional multilayer feedforward neural networks. Furthermore, Scheme 1 can get higher recall accuracy than Scheme 2 while the latter converges faster in training, which agrees with the prediction in Section 4.3. It is also noted that the recall performance deteriorates with decreasing number of quantization levels. This

4.6 ADVANTAGES FOR HARDWARE IMPLEMENTATION

For MFNNs with quantized neurons, the powers-of-two factors involved in multiplications are outputs of neurons. Since these outputs vary from pattern to pattern, there is no such cases that direct wiring can be applied. However, the shifter using simple combinational logic as suggested in Chapter 3 is still applicable and can improve significantly the area-delay product over multiplications. The difference is that the weight is connected to pin "A" of the shifter as input vector and the input to the neuron (the output from another neuron) functions as the control vector. This configuration is illustrated in Fig. 4.6. Table 4.6 illustrates the advantages of the MFNN with quantized neuron model over the original MFNN model for VLSI implementation of the multiplication in the weighted sum calculations. An 8x8 multiplier is assumed for MFNN model while the MFNN with quantized neurons model is as presented previously in this Chapter with M=4. Again, the LSI Logic 600k CMOS ASIC technology is used in the example. The VHDL description of the shifter is listed in Table D.1 of Appendix D.

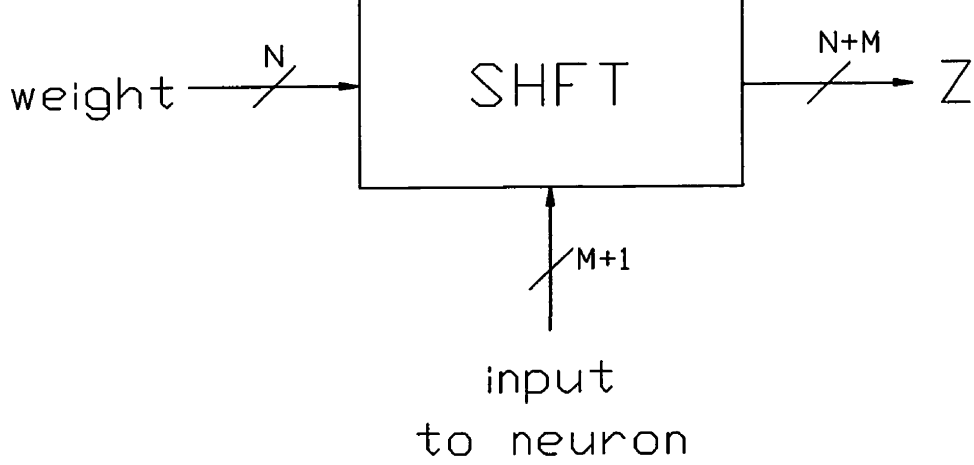


Figure 4.6 A shifter used in MFNNs with quantized neurons

Table 4.6 Hardware advantage of MFNN with quantized neurons

	MFNN	STPT-MFNN
Calculation	$z_j^{[h]} * w_{ij}^{[h]}$	$z_j^{[h]} * w_{ij}^{[h]}$
Implementation	multiplier (8x8)	shifter
Area (# gates)	842	158
Delay (ns)	11.30	1.35

that it simplifies the implementation of the nonlinear activation function. As described previously in this chapter, activation functions used in MFNNs with quantized neurons are multistep functions with STPT outputs. This simplified STPT multistep activation function can be realized easily using either comparators or other combinational logic. For any input to the function, the STPT output value can be determined by simply compare the input with the thresholds related to each output level. This procedure is illustrated in Fig.4.7, where only one bit in the output vector will be one, all others are zero. A further description can be found in Table 4.7, where $TH_1 > TH_2 > \ldots > TH_n$ are thresholds and $C_i = 1$ when $IN \geq TH_i$ and $C_i = 0$ otherwise. A VHDL description of such an implementation when $M = 4$ is listed in Table D.2. The input is assumed to be 16 bits.

Table 4.7 Description of the decoder for STPT multistep activation function

OUTPUT FROM COMPARATORS (INPUT TO THE DECODER) $C_n \ldots C_1$	OUTPUT FROM THE DECODER (OUTPUT OF THE FUNCTION) $Q_n \ldots Q_1$
00...0000	0000...00
11...1111	1000...00
11...1110	0100...00
11...1100	0010...00
.....	.....
10...0000	0000...01

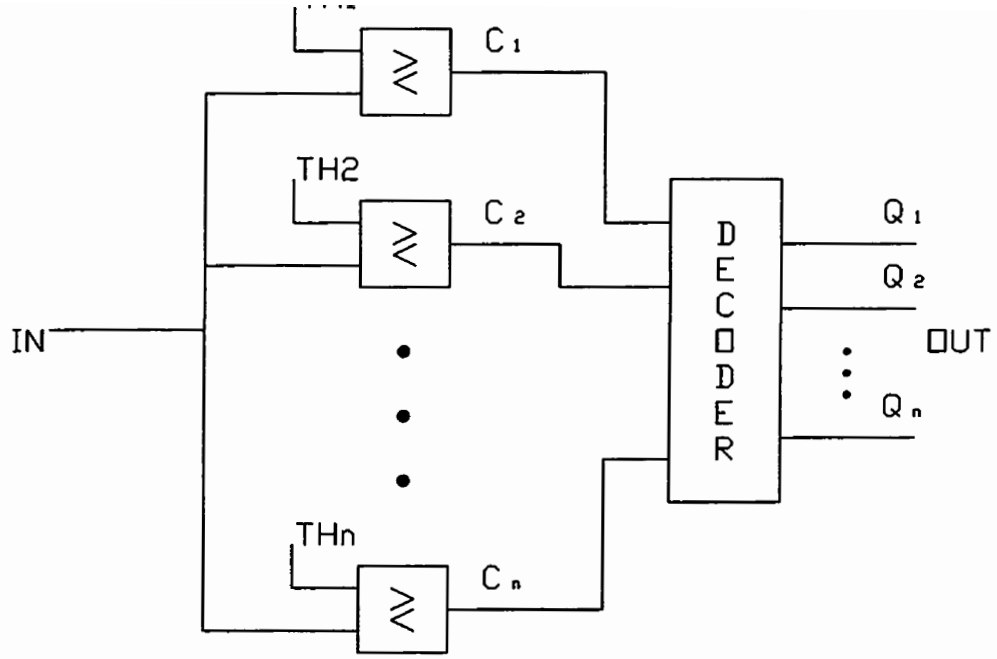


Figure 4.7 Block diagram of the STPT multistep activation function

The synthesized circuit in LSI Logic 600k CMOS ASIC technology is shown in Fig.4.8. A more detailed schematic is drawn in Fig.4.9 with an area of 1937 design units (646 gates). The complexity of this design is only comparable to the address decoder (an n to 2^n decoder) part of a memory based look-up-table implementation, as shown in Fig.4.10, which is the commonly used solution of the nonlinear activation function. A 128 words 8-bit look-up-table implementation in LSI 600k technology requires approximately 2560 gates.

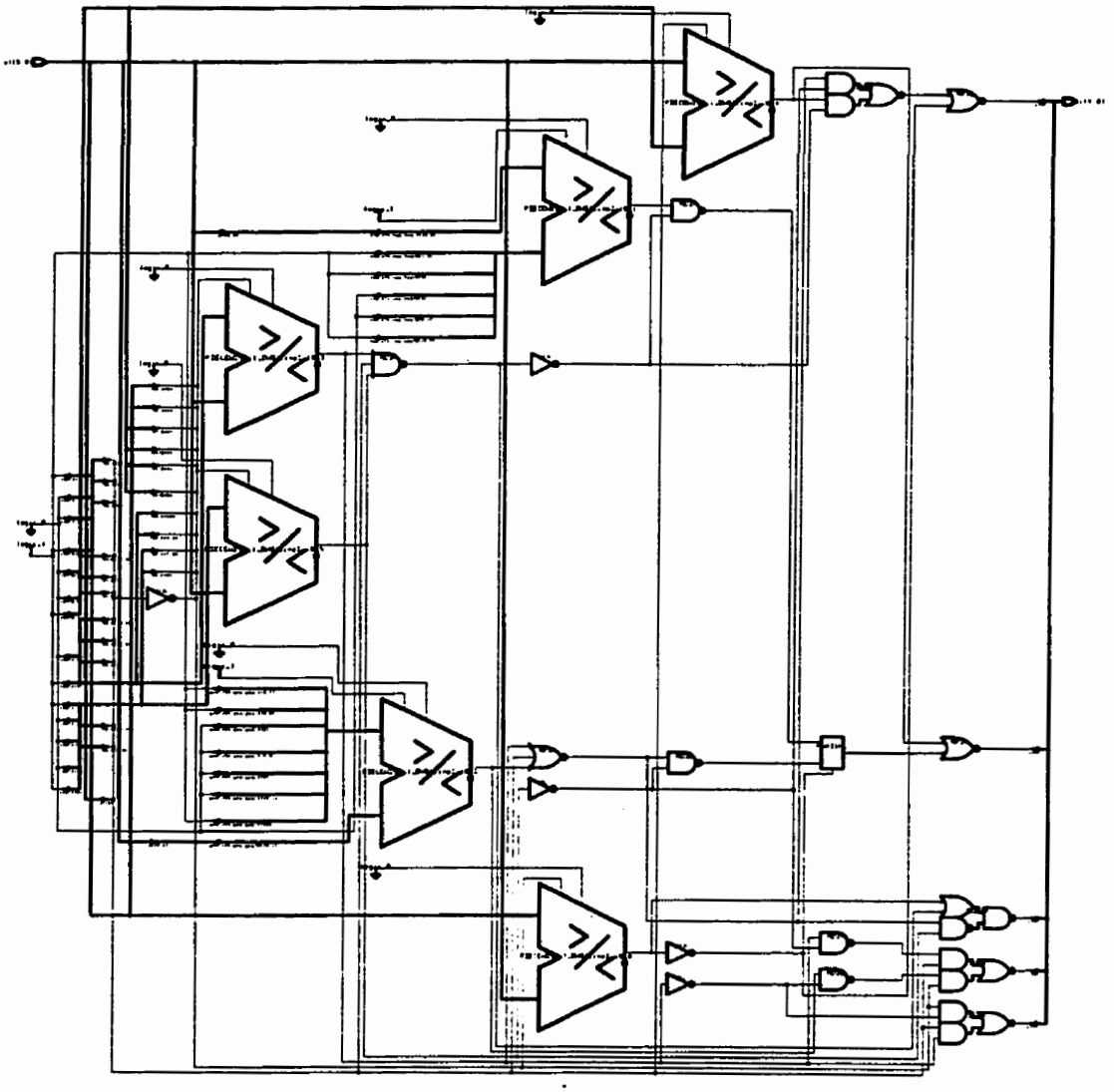


Figure 4.8 Multistep activation function circuitry

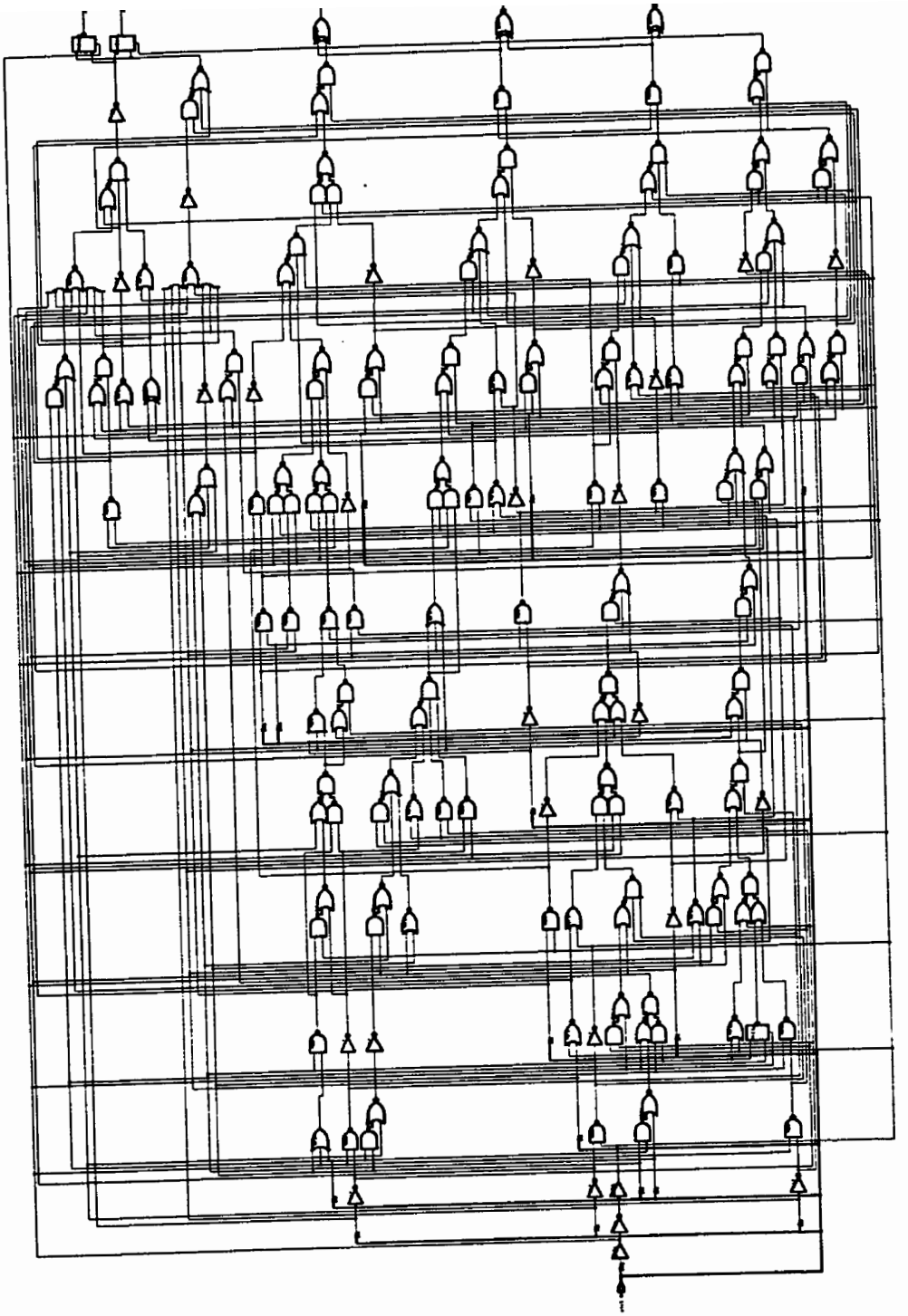


Figure 4.9 Schematic of the multistep activation function

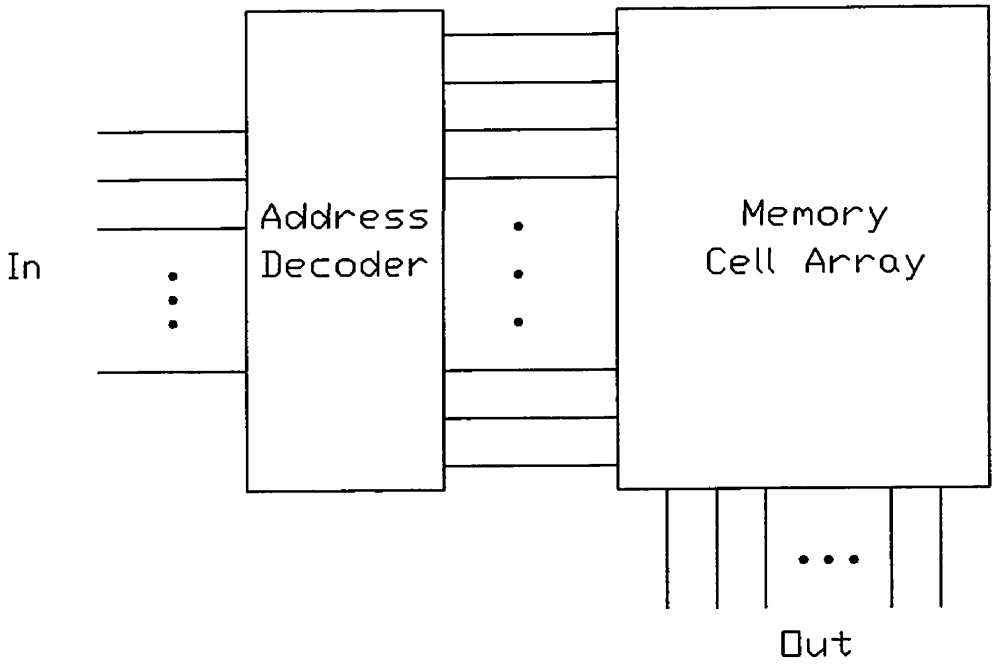


Figure 4.10 Structure of a look-up-table

4.7 Concluding Remarks

The concept of quantized neurons was introduced in this chapter to alleviate the computational burden of massive multiplications in multilayer feedforward neural networks. A modified backpropagation algorithm was developed to meet the training requirements of MFNNs with quantized neurons.

network, an MFNN with quantized neurons has significant advantages for digital hardware implementation. First of all, a reduced chip area and an increased computational speed can be achieved due to the fact that multiplications are replaced by shift operations only, which is a similar feature as for the model of MFNNs with STPT weights proposed in Chapter 3. Yet one more advantage of using quantized neurons in MFNNs is that the complexity of the implementation of nonlinear activation functions, which is another shortcoming in digital techniques, can be reduced substantially. More on the issue of implementation of activation functions will be discussed in the next chapter.

MORE MFNN MODELS FOR DIGITAL IMPLEMENTATIONS

In the previous chapters, MFNN models with STPT weights and quantized neurons were proposed. These models have been shown to be effective in alleviating some of the computational burden in digital implementation of neural networks. Based on the ideas in the development of those two models, along with the introduction of a new form of sigmoid activation function, further MFNN models will be developed in this chapter for alternative digital hardware implementation.

5.1 A Simplified Sigmoid Activation Function (SSAF)

The sigmoid function as defined in equation (3.2) is the most popular nonlinear activation function used in artificial neural networks. However, this sigmoid function is not suitable for direct digital implementation as it consists of an infinite exponential series. A look-up-table has been a traditional method for implementing the sigmoid function for which the amount of hardware

overcomes many of the limitations of single-layer perceptron so that MFNNs can approximate any input-output functions. The sigmoid function provides, at the output of a neuron, a nonlinearity that has a *tanh*-like transition between the lower and upper saturation regions. In practice, any nonlinear function which possesses a similar transition region may be expected to achieve similar performance in MFNNs. In this section, it is shown that a simple second-order piecewise nonlinear function exists which can be used as an activation function in MFNNs. The proposed piecewise activation function can be implemented directly using digital techniques.

5.1.1 Second-Order Approximation

Consider the following second-order piecewise nonlinear function, which has a tanh-like transition between an interval [-L, L]

$$H(x) = \begin{cases} x(\beta - \theta x) & for \quad 0 \le x \le L \\ x(\beta + \theta x) & for \quad -L \le x \le 0 \end{cases} \tag{5.1}$$

where ß and Θ determine the slope and the gain of the function. Consequently, a sigmoid-like bipolar function can be realized by

$$G(x) = \begin{cases} 1 & for\ L \le x \\ H(x) & for\ -L \le x \le L \\ -1 & for\ x \le -L \end{cases} \tag{5.2}$$

$$G_{0/1}(x) = \begin{cases} 1 & \text{for } L \le x \\ \frac{1}{2}H(x) + \frac{1}{2} & \text{for } -L \le x \le L \\ 0 & \text{for } x \le -L \end{cases} \tag{5.3}$$

To determine the parameters ß and Θ, the following condition can be used

$$H'(x)|_{x=L} = 0 \tag{5.4}$$

hence

$$\beta = 2\theta L \tag{5.5}$$

Also, based on the condition H(L) = 1, we obtain

$$\beta L - \theta L^2 = 1 \tag{5.6}$$

From equations (5.5) and (5.6), the following relationships can be obtained

$$\begin{cases} \beta = \frac{2}{L} \\ \theta = \frac{1}{L^2} \end{cases} \tag{5.7}$$

where L determines the saturation point of the function.

conventional sigmoid function given below

$$F(x) = \frac{1 - e^{-\alpha x}}{1 + e^{-\alpha x}} \tag{5.8}$$

The parameters used are $L = 2$ and $\alpha = 2$. It can be seen that these two curves are very close. The maximum difference, which is about 4%, occurs around the saturation points. With such a close approximation, similar performance can be expected when the piecewise activation function is used with MFNNs.

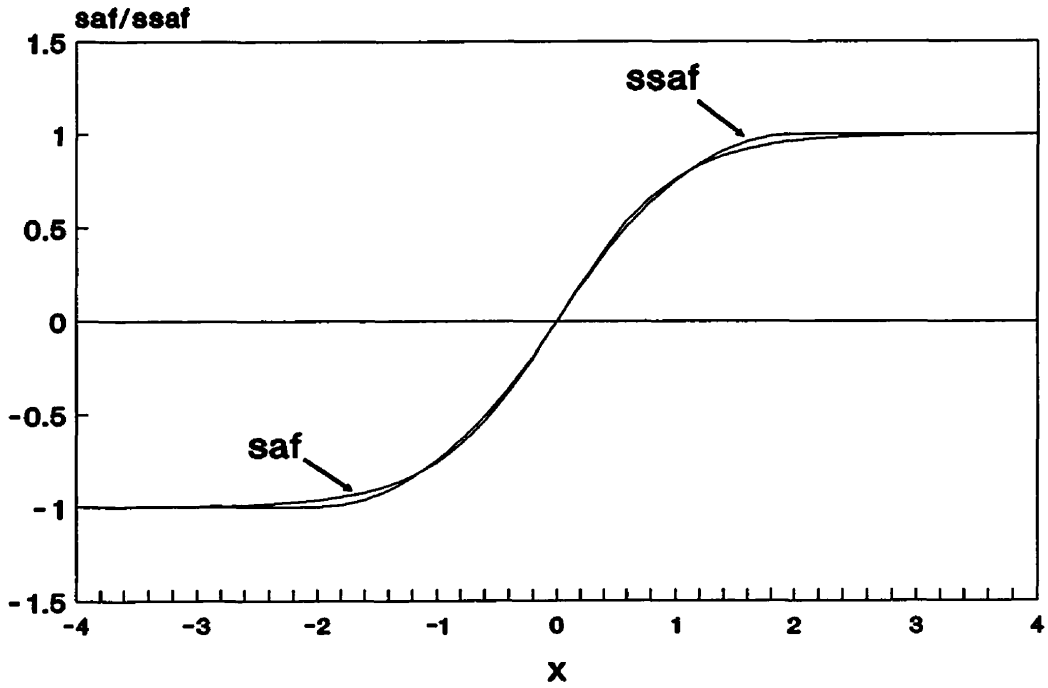


Figure 5.1 Sigmoid Activation Function (SAF) and Simplified Sigmoid Activation Function (SSAF)

Although the piecewise function G(x) is a very good approximation to the sigmoid activation function F(x), there is a major difference between the two functions. It can be seen that the piecewise function has a zero derivative beyond the saturation points $\pm L$, as shown in equation (5.9), which is not suitable for training with the backpropagation algorithm. This problem needs to be solved before the piecewise function G(x) can be used as activation functions in MFNNs.

$$G'(x) = \begin{cases} 0 & \text{for} \quad L \leq x \\ H'(x) & \text{for} \quad -L \leq x \leq L \\ 0 & \text{for} \quad x \leq -L \end{cases} \tag{5.9}$$

If the derivative of the piecewise activation function is used directly with the algorithm, the learning process will get stuck when it happens to be in the wrong saturation region. It needs a little push to bring the learning out of the premature saturation region. A small positive value of the derivative may serve the purpose. By introducing a small positive value δ into G'(x) in both saturation regions, we have

$$G'(x) = \begin{cases} H'(x) & \text{for } -(L-\Delta) \le x \le L-\Delta \\ \delta & \text{for } -(L-\Delta) \ge x \end{cases} \quad (5.10)$$

The learning process can be carried out using this version of the derivative of G(x). While the newly defined G'(x) will serve in the backward operations during training, the piecewise activation function G(x) is still used in the forward operations. Given a small positive value of δ, the offset Δ can be determined by setting $H'(L-\Delta) = \delta$, i.e.,

$$\beta - 2\theta(L-\Delta) = \delta \quad (5.11)$$

Solving the above equation and taking into account equation (5.7), we have

$$\Delta = \frac{L^2}{2}\delta \quad (5.12)$$

A direct digital hardware implementation of H(x) can be carried out according to the signal flow graph as shown in **Fig.5.2**. This implementation can be simplified when L takes a value in single term powers-of-two format. In this situation, both Θ and ß are also single term powers-of-two values according to Eq.(5.7) such that H(x) can be implemented by one multiplication together with one shift and one addition as illustrated by **Fig.5.3**. A VHDL description of the implementation of the simplified sigmoid activation function

circuitry is drawn in **Fig. 5.4**, which clearly indicates that there are one multiplier, one adder/subtracter, and two comparators in the implementation. The detailed schematic of this design is drawn in **Fig.5.5** and has an area of 2661 design units (887 gates) and a maximum delay of 13.14 ns in LSI Logic 600K ASIC technology. In comparison, a 128x8-bit look-up-table, as discussed in section 4.6, requires 2560 gates in the same technology.

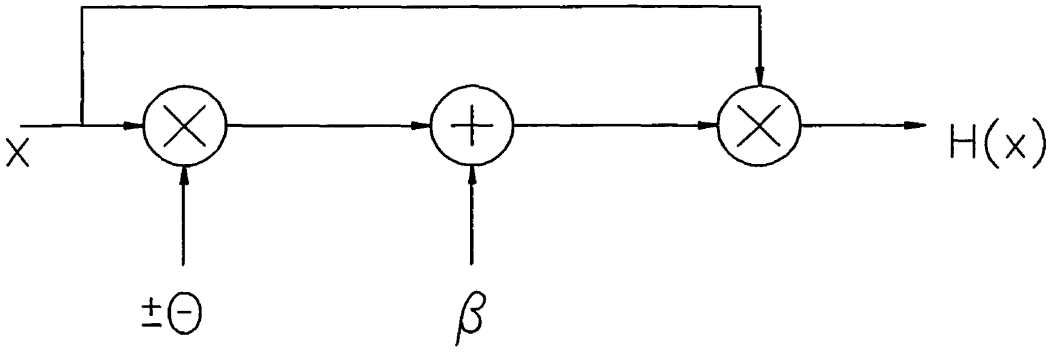


Figure 5.2 Block Diagram for Implementation of H(x)

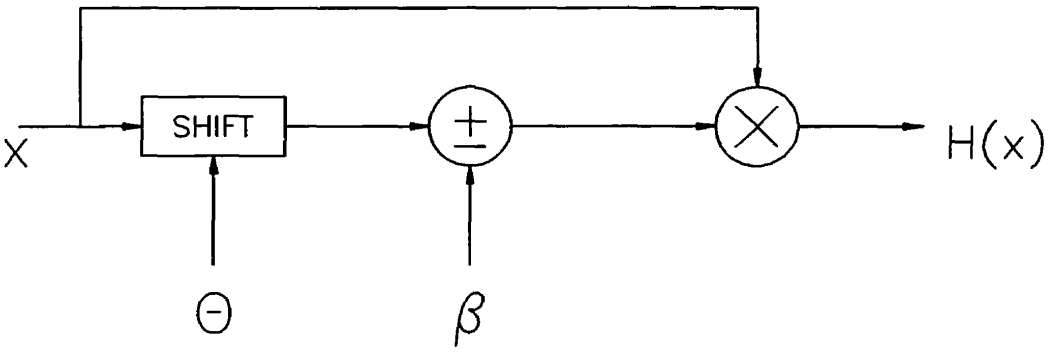


Figure 5.3 H(x) with STPT L

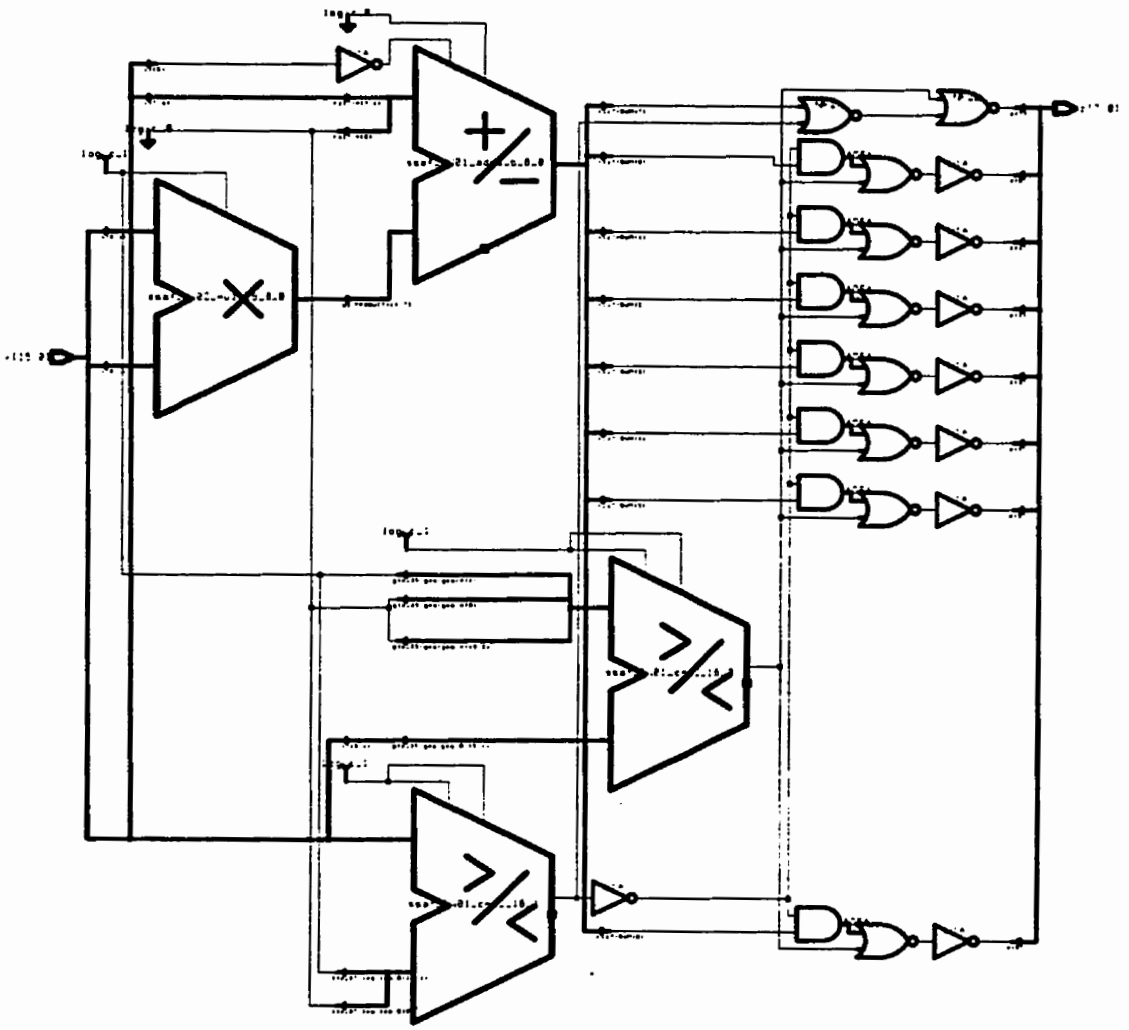


Figure 5.4 Implementation of the simplified sigmoid activation function

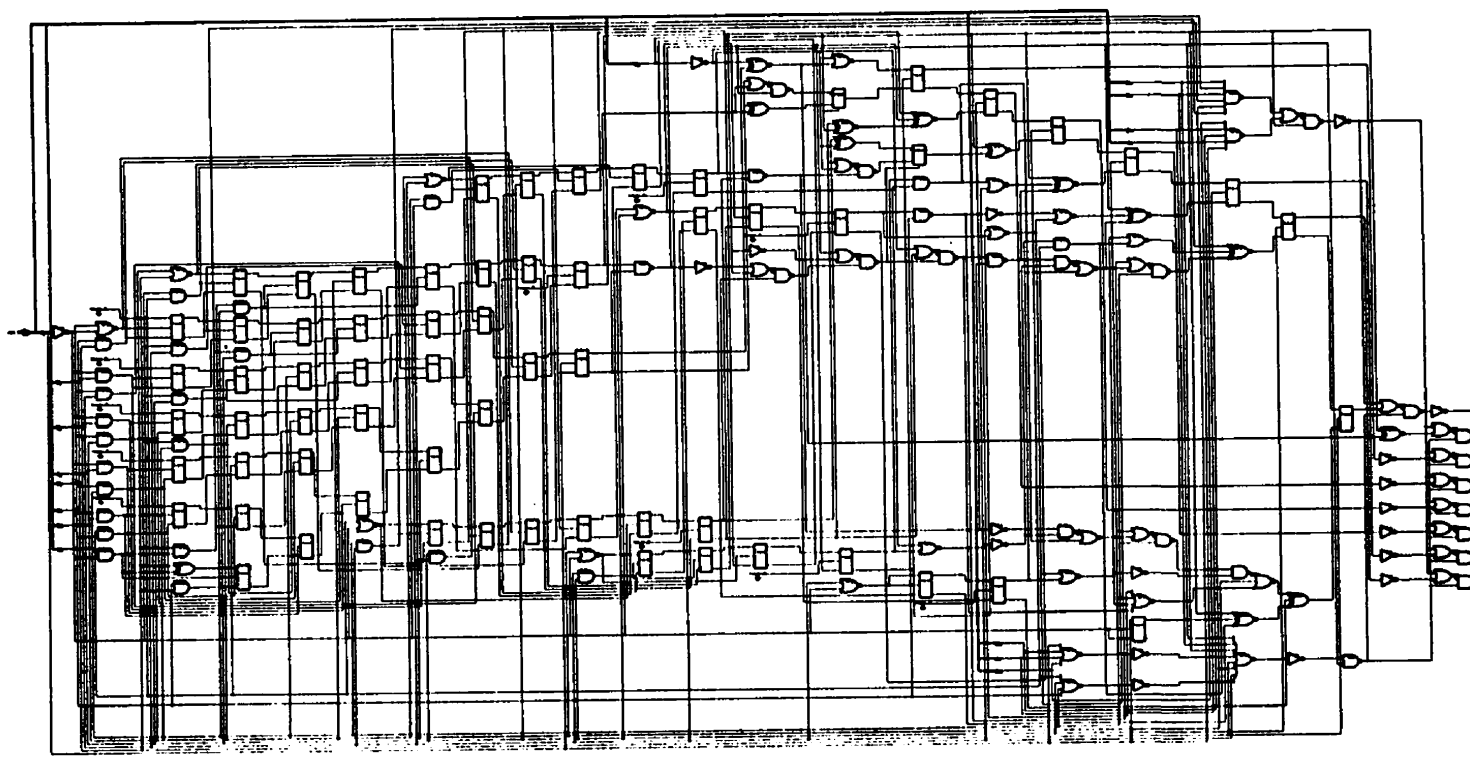


Figure 5.5 Schematic of the Simplified Sigmoid activation function

In this section, simulation results will be presented in which the performance of the proposed piecewise activation function G(x) is compared with the performance of the traditional sigmoid activation function as shown below

$$F(x) = \frac{1 - e^{-\alpha x}}{1 + e^{-\alpha x}} \tag{5.13}$$

The two activation functions were used to train a two-layer and a three-layer feedforward neural network using the backpropagation algorithm. The following parameters of the functions were used:

G(x): $L = 2$

F(x): $\alpha = 2$

Two curves of activation functions with these parameters are shown in **Fig.5.1.** The training pattern-pairs of 10 numerals, which are the same as those used in Chapter 3, were used in simulations. The feedforward neural networks have 100 input nodes, 4 output neurons, and one or two hidden layers each with 40 hidden neurons. The learning rate parameter used was 0.01 in all simulations. For each combination of function and network, three set of initial random weights uniformly distributed in the interval [-0.1, 0.1] were used. For recall

constructed by inverting 5% of its original pixels randomly. The results on the number of training epochs to reach a sum of squared output errors of 0.1 over the entire training pattern set and the generalization capabilities, which were measured as the percentage of correct recalls over all patterns, are summarized in Table 5.1. The simulation results show that the proposed piecewise activation function can achieve a similar performance as that of the traditional sigmoid activation function.

Table 5.1 Performances of SSAF and SAF for Two- and Three-Layer FNNs

	F(x)		G(x)	
	Training (Epochs)	Generalization (%)	Training (Epochs)	Generalization (%)
Two-layer network	33	99.6	29	99.6
	32	99.1	27	99.1
	32	99.0	28	99.0
Three-layer network	39	99.5	35	99.4
	41	99.9	38	99.9
	38	99.9	35	99.9

The simplified sigmoid activation function (SSAF) proposed in the last section has nearly identical features to the traditional sigmoid activation function, but can be implemented easily using one multiplier. When used in combination with STPT weights, the implementation of network can be further simplified. In this section, a method for designing MFNNs using SSAFs and STPT weights will be developed.

5.2.1 Design Algorithm

Our objective is to design an MFNN with SSAF at the output of each neuron and STPT weights of the form of $\{\pm 1, \pm 2^{-1}, \ldots, \pm 2^{-M}, 0\}$. The design procedure can be carried out as follows.

Step 1: Starting with small random weights and zero biases, train an MFNN with SSAFs and continuous weights using the backpropagation algorithm without adjusting biases until convergence to an predetermined error level E_0, i.e.,

$$\sum_{k=1}^{K} e_k < E_0 \qquad (5.14)$$

Step 2: Find the maximum absolute value in each layer among weights $w_{ij}^{[h]}$, which have been obtained in Step 1.

$$w_{max}^{[h]} = \max_{i,j} \{ |w_{ij}^{[h]}| \} \tag{5.15}$$

Then find the smallest STPT value which is greater than or equal to $w_{max}^{[h]}$ and denote it as $2^{p[h]}$. Now set $w_{max}^{[h]}$ to this STPT value $w_{max}^{[h]} = 2^{p[h]}$.

Step 3: Normalize weights $w_{ij}^{[h]}$ as

$$w'^{[h]}_{ij} = 2^{-p[h]} w_{ij}^{[h]} \tag{5.16}$$

Step 4: Adjust the parameters in SSAFs at each layer accordingly as

$$\begin{aligned} \beta^{[h]} &= 2^{p[h]} \beta^{[h]} \\ \theta^{[h]} &= 2^{2p[h]} \theta^{[h]} \end{aligned} \tag{5.17}$$

such that they remain STPT values.

Step 5: Quantize the normalized weight $w'^{[h]}_{ij}$ to its nearest STPT value

$$w^{*[h]}_{ij} = sgn(w'^{[h]}_{ij}) \begin{cases} 0 & \text{if } |w'^{[h]}_{ij}| < C_M \\ 2^{-m} & \text{if } C_m < |w'^{[h]}_{ij}| < C_{m-1} \end{cases} \qquad (5.18)$$

where $sgn(w'_{ij}{}^{[h]})$ stands for the sign of $w'_{ij}{}^{[h]}$ and C_m is defined as

$$C_m = \frac{3}{4}2^{-m} \qquad m = 0, 1, \ldots, M \qquad (5.19)$$

Step 6: Calculate the TSE using current STPT weights and SSAFs. If TSE is less than the predefined level E_0, i.e.,

$$\sum_{k=1}^{K} e_k < E_0 \qquad (5.20)$$

stop; otherwise, proceed to Step 7.

Step 7: Compute changes in weights $w^*_{ij}{}^{[h]}$ and biases $b_j^{[h]}$, respectively, according to the equations in the BP algorithm.

Step 8: Update weights and quantize them as in Step 5. Denote the old set of weights as $\{w^*_{ij}{}^{[h]}\}_{old}$ and the new set as $\{w^*_{ij}{}^{[h]}\}_{new}$, and then calculate TSE using both sets of weights, if

accept new weights by setting

$$\{W^{*[h]}_{ij}\} = \{W^{*[h]}_{ij}\}_{new} \tag{5.22}$$

otherwise, discard them and keep old weights.

Step 9: Update biases $b_j^{[h]}$ and go back to Step 6.

5.2.2 Simulation Results

The 10 numeral patterns, each with 10x10 bipolar pixels, shown in Fig.3.4 are used in the simulations. Each MFNN was trained to obtain both continuous and STPT weights. Sets of noisy patterns were fed to each continuous-weight MFNN and its corresponding STPT-weight MFNN to test their generalization capabilities in terms of recall accuracy (in percentage of correct recalls). A noisy version of each of the 10 training patterns was constructed, as before, by inverting randomly 5% of the original elements. The recall accuracy was obtained by taking the average of the results among 100 noisy versions of each of the 10 training patterns. Simulation results are summarized in Table 5.2 and Table 5.3, where CMFNN and STPTMFNN represent, respectively, the MFNNs with continuous and STPT weights. For

All data given in Tables 5.2 and 5.3 were averages of five designs, starting with different initial random weights uniformly distributed in [-0.1, 0.1]. Other parameters used were $\epsilon = 0.01$; $\epsilon_b = 0.1$; $M = 4$; $E_0 = 0.01$; $E = 0.2$; $D = 2$; and $\delta = 0.01$.

It can be seen that convergence was reached in all designs. The designed MFNNs with STPT weights can retain the generalization capability of the corresponding MFNN with continuous weights as the degradations in performance were at most 0.42% over all designs.

One Hidden Layer

Number of Hidden Neurons	Training (Epochs)		Generalization (%)	
	CMFNN	STPTMFNN	CMFNN	STPTMFNN
10	96.6	72.2	99.44	99.04
20	350.2	1.0	99.72	99.50
40	300.0	2.2	99.50	99.40
60	294.6	1.0	99.58	99.48
80	61.6	1.0	99.56	99.50
100	45.0	1.0	99.54	99.32

Table 5.3 Convergence Speed and Generalization Capabilities of MFNNs with Two Hidden Layers

Number of Hidden Neurons	Training (Epochs)		Generalization (%)	
	CMFNN	STPTMFNN	CMFNN	STPTMFNN
10	385.2	71.0	99.42	99.00
20	846.0	381.8	99.46	99.34
40	712.2	3.4	99.40	99.38
60	1021.2	1.0	99.52	99.48
80	483.4	1.0	99.56	99.50
100	307.6	1.0	99.60	99.46

In this section, the ideas of STPT weights, quantized neurons, and simplified sigmoid activation functions will be combined to generate an MFNN model with no weight multiplications for continuous input-output mapping.

The designed MFNNs will have following features:

- STPT weights in the input layer and continuous weights in all other layers
- Simplified sigmoid activation functions at output neurons
- three-level activation functions (3-LAFs) at hidden neurons

A three-level activation function is a special case of the quantized neuron presented in Chapter 4 when M = 0 and can be expressed as

$$F_3(x) = \begin{cases} 1 & \text{for } t \le x \\ 0 & \text{for } -t < x < t \\ -1 & \text{for } x \le t \end{cases} \qquad (5.23)$$

where t is a positive threshold value. The derivative of $F_3(x)$ can be determined by using the method described in Chapter 4, i.e., finding the three intersection of $F_3(x)$ and the following function

$$F(x) = g\frac{}{1+e^{-\alpha x}}$$

where $g > 1$ is a gain factor. The derivative of F(x) at these three intersections will be used as the approximation of $F'_3(x)$ in three different regions defined in equation (5.23), respectively, during training.

5.3.1 Design Algorithm

Step 1: Prepare a set of random weights and zero biases.

Step 2: Starting with the latest weights and zero biases, train the network using the backpropagation algorithm, with the SSAFs at the output neurons and the 3-LAFs at the hidden neurons. The weights will keep updated until the TSE becomes less than a prespecified error level E_0. The obtained network is denoted as Net 1.

Step 3: Find the maximum absolute value w_{max} among the weights in the first layer and normalize these weights by w_{max}.

Step 4: Scale the threshold value t of 3-LAFs applied to hidden neurons by

Step 5: Quantize those normalized weights in the first layer to their nearest STPT values from the set of $\{\pm 1, \pm 2^{-1}, \ldots, \pm 2^{-M}, 0\}$.

Step 6: Calculate the TSE. If $TSE < E_0$, stop and denote the network obtained here as Net 2; otherwise, proceed to Step 7.

Step 7: Re-adapt all continuous weights in all layers rather than the first one, and biases in all neurons using the backpropagation algorithm.

Step 8: Go back to Step 6.

5.3.2 Simulation Results

Simulation results are provided in **Table 5.4.** Two normalized orthogonal continuous real vector sets, one as input pattern set and the other as target set, were used for training and recall. Each vector set consists of 10 vectors and each vector consists of 25 continuous real elements, which are generated by using a method described in Reference [Kwan et al., 1993]. The network was used as a pattern associator, which had 25 inputs, 25 outputs, and one

versions of each of the 10 input vectors were presented to the network to test the recall performance. The noisy vectors were constructed by adding random noise within the interval of $\pm R$ to each element of each input vector. R represents a percentage of the maximum element value among all the 10 input vectors. In the simulations presented here, R was 10% or 20%. The output vector was identified based on its crosscorrelations with all ideal output vectors. The ideal output vector with maximum crosscorrelation was selected as the recall vector. For comparison, the simulation results of the corresponding continuous MFNN (CMFNN), which had the same topology but continuous weights and bipolar sigmoid activation functions at both layers, were also obtained. The data summarized in **Table 5.4** represent the average of five designs, starting with different initial random weights uniformly distributed within ± 0.1. The learning rate parameter for weights was $\epsilon = 0.01$, the step size for bias adjustment was $\epsilon_b = 0.01$, and other parameters used were $\delta = 0.01$, $\alpha = 2$, $D = 2$, $M = 4$, and $E_0 = 10^{-6}$.

It can be seen that the proposed MFNNs with SSAFS, 3-LAFs, and STPT weights have a similar recall performance as the original MFNNs with SAFs and continuous weights at a cost of additional training epochs.

Table 5.4 Summary of Simulation Results

Number of Hidden Neurons	Training (Epochs)			Recall (%)			
	CMFNN	MMFNN		10% noise		20% noise	
		Net 1	Net 2	CMFNN	MMFNN	CMFNN	MMFNN
10	587.0	771.6	2690.0	100.0	97.48	100.0	94.28
15	380.2	427.8	313.4	100.0	99.96	100.0	99.68
20	339.8	256.6	211.4	100.0	100.0	100.0	99.84
25	221.6	124.8	99.2	100.0	100.0	100.0	100.0
30	174.8	117.2	70.0	100.0	100.0	100.0	100.0

5.4 Multiplierless MFNNs for Discrete Input-Output Mapping

When the input and output patterns are of discrete format (binary or bipolar), some limitations in the design of MNFFs without weight multiplications can be removed and the design will be more flexible.

In this section, a method for designing 2-layer feedforward neural networks suitable for bipolar (± 1) input to output mapping will be presented, which uses simplified sigmoid activation functions at hidden neurons, step activation functions at output neurons, continuous valued weights in the first layer, and single-term powers-of-two weights in the second layer such that multipliers can be eliminated from the resultant networks. The designed network will have the following properties:

- bipolar (± 1) input and output
- one hidden layer
- continuous weights at first layer and single-term powers-of-two weights at the second layer
- SSAFs at the hidden layer
- SAFs at the output layer for training and step functions at the output layer for recall

5.4.1 Design Algorithm

Step 1: Prepare a set of random weights and zero biases, with sigmoid activation functions at the output layer and simplified sigmoid activation functions (with a single-term powers-of-two L) at the hidden layer.

Step 2: Starting with the latest weights and zero biases, train the network using the BP algorithm without adjusting biases until

$$\sum_{k=1}^{K} e_k < E \tag{5.25}$$

where E is a prespecified error level. The network obtained at this point is denoted as Net#1.

Step 3: Find the maximum absolute value $w_{max}^{[2]}$ among the weights in the output layer and normalize these weights by $w_{max}^{[2]}$.

Step 4: Adjust the parameter α of the sigmoid activation functions applied at the output neurons as $\alpha w_{max}^{[2]}$.

term powers-of-two values chosen from the following set:

$$\{ \pm 1, \pm 2^{-1}, \pm 2^{-2}, \cdots, \pm 2^{-S}, 0 \} \qquad (5.26)$$

Step 6: Calculate the TSE. If TSE$<$E is not satisfied, proceed to step 7; otherwise, go to step 8.

Step 7: Adapt all continuous weights of the first layer and biases of neurons at both layers using the equations given in Section II until either Eq.(13) is satisfied or convergence is reached in which no further improvement in SSE can be obtained.

Step 8: Find the maximum absolute value $w_{max}^{[1]}$ among the weights in the first layer, and set $w_{max}^{[1]} = 2^{p[1]}$, where $2^{p[1]}$ is the smallest single-term powers-of-two value greater than or equal to $w_{max}^{[1]}$.

Step 9: Normalize the weights in the first layer by $2^{p[1]}$ and set parameters ß and Θ of the SSAFs applied at hidden neurons as $ß = 2^{p[1]}ß$ and $\Theta = 2^{2p[1]}\Theta$, respectively, such that they remain single-term powers-of-two.

Step 10: Replace the sigmoid activation functions at the output layer by

as Net#2.

5.4.2 Simulation Results

Simulations have been conducted to verify the proposed design algorithm. The input patterns used in training were 10 numerals as shown in Fig.3.4, each represented by 10x10 bipolar pixels. The corresponding desired output patterns were 4-bit codes given below each input pattern. Thus, the network had 100 inputs, 4 outputs, and one hidden layer with various number of neurons. After training, 100 noisy versions of each of the 10 input patterns, in total 1000, were presented to test the recall accuracy of the network obtained. A noisy pattern was constructed by inverting randomly a percentage (in this paper it was 5%) of elements of the original pattern. The recall accuracy was obtained by taking the average over all 1000 testing patterns. Simulation results are summarized in Tables 5.5 and 5.6. All data given in these tables were averages of five designs, starting with different initial random weights uniformly distributed in [-0.1, +0.1]. For the purpose of comparison, the results of corresponding continuous-weight MFNN (CMFNN), which had the same topology but continuous weights and sigmoid activation functions at both layers, were also obtained and included in these tables. The total number of epochs under MMFNN is the sum of epochs required to obtain both Net#1 and

target patterns. The other parameters used for simulations were: $\epsilon = 0.01$, $\epsilon_b = 0.1$, E = 0.01, $\alpha = 2$, D = 2, and $\delta = 10^{-2}$.

Based on the data in Tables 5.5 and 5.6, we can see that the convergence was always reached in the training of MMFNN and there was only slight degradation in the recall performance of MMFNN compared with CMFNN.

Table 5.5 CONVERGENCE SPEED (IN NO. OF EPOCHS) OF CMFNNs AND MMFNNs

Number of hidden neurons	CMFNN	MMFNN		
		S = 2	S = 4	S = 8
10	202.8	721.2	629.0	696.4
20	98.0	182.6	172.2	172.2
40	78.2	155.8	141.8	141.2
60	71.4	141.2	146	146.8
80	67.4	120.2	126.6	126.6
100	56.0	99.2	99.8	100.6

Table 5.6 RECALL PERFORMANCE (IN PERCENTAGE OF CORRECTNESS) OF CMFNNs AND MMFNNs

Number of hidden neurons	CMFNN	MMFNN		
		S=2	S=4	S=8
10	99.50	99.43	99.08	98.96
20	99.72	99.60	99.68	99.66
40	99.56	99.46	99.56	99.58
60	99.54	99.56	99.60	99.60
80	99.62	99.58	99.60	99.60
100	99.62	99.46	99.58	99.58

5.5 Concluding Remarks

A simplified sigmoid activation function (SSAF) has been proposed in this chapter for direct digital implementation. This presented model is a piecewise function which has a very close approximation to the original sigmoid function and performs equally when used in multilayer feedforward neural networks. The advantage of SSAF for hardware implementation was demonstrated by the fact that it requires much less silicon area than the commonly used look-up-table method.

Based on the SSAF model and combined with the ideas of STPT weights

feedforward neural networks architectures suitable for digital hardware implementation were developed under different conditions. While having advantages for digital implementation approach, all these models can retain the performance of the original multilayer feedforward networks as shown by the simulation results.

Chapter 6

CONCLUSIONS AND SUGGESTIONS

6.1 Conclusions

This dissertation has made original contributions to the development of artificial neural network models for digital hardware implementations.

First, a new model of multilayer feedforward neural network with single term powers-of-two weights is proposed in Chapter 3 along with a dedicated design algorithm. The adaptive sigmoid activation function has been introduced for fine-tuning the network to compensate the errors caused by weight quantization. This method gives the network more dimensions of freedom in addition to weight adjustment to adapt to a given problem. The proposed algorithm turns out to be effective in designing MFNNs with STPT weights. MFNNs with STPT weights have substantial advantages over original MFNNs in digital hardware implementation. By using STPT weights, multiplications are eliminated such that only shift operations are required. This has resulted in

feasibility of the proposed model and algorithm was demonstrated by simulation results. STPT-weight networks can retain a similar performance to the original continuous-weight networks while avoiding weight multiplications in digital hardware implementations.

The STPT weights were introduced in an attempt to alleviate the computational burden of multiplications, no extra effort has been made to ease the interconnection problem in digital implementation. However, if we take into account the effect of the increased number of zero weights and the reduced bit width of non-zero weights as a result of the adoption of STPT weights, we can still see some reduction in the density of interconnections, although this impact is limited and no substantial improvement is expected.

A new model for MFNNs with quantized neurons is proposed in Chapter 4. The concept of a quantized neuron is introduced and its structure is demonstrated. The output of a quantized neuron is restricted to STPT format with a multistep activation function. The BP algorithm has been modified to handle the training of quantized neurons. A methodology for designing MFNNs with quantized neurons is presented and has been proved to be very effective through simulations. The advantages of using quantized neurons in MFNNs for digital hardware implementation include elimination of weight multiplications

functional blocks have been proposed and significant improvement in terms of speed and silicon area has been achieved. In conclusion, MFNNs with quantized neurons have shown great advantages over digital hardware implementation with little degradation in the network performance when compared with original MFNNs.

A simplified sigmoid activation function is proposed in Chapter 5, which is a very close approximation to the original sigmoid activation function and has the same performance in simulations. A corresponding training algorithm has been developed and a cost effective direct hardware implementation is presented. More multiplierless MFNN models are also developed in Chapter 5 based on the idea of STPT weights and the simplified sigmoid activation function. The effectiveness of these models are verified via computer simulations.

Real world applications may require very large neural networks with hundreds of thousands neurons, or even more. Increased complexity of ANN will definitely result in a high cost of hardware implementation, which could limit the wide application of ANNs. Thus, there is an urgent need in cost effective implementation of ANNs and the strategies proposed in this dissertation are able to serve this purpose well.

While the proposed models and algorithms have been shown to be successful in designing multilayer feedforward neural networks, the following open problems still need further investigation.

The original multilayer feedforward neural networks are universal approximators. Do MFNNs with powers-of-two weights still have this property? It would be very interesting to see whether it is possible to find any direct analytical solution in this regard. Although it is expected that the analysis may be quite complex in nature.

Our simulation results show that MFNNs with STPT weights or quantized neurons can achieve almost the same performance as the original MFNNs without an increase in the size of the network. To what extent this result can still hold is a good direction for future mathematical analysis.

The models proposed in this dissertation can eliminate multiplication in the feedforward operations only. That means multiplications are still inevitable in the learning phase. The learning algorithm without multiplications is needed if on-chip learning is to be implemented digitally.

Appendix A

DERIVATION OF THE BP ALGORITHM

The backpropagation (BP) algorithm is a gradient descent method, which makes the change in a weight to be proportional to the negative derivative of a cost function with respect to that weight. If the total squared error (TSE) defined in equation (2.9) is used as the cost function, then the change in weight $w_{ij}^{[h]}$ due to pattern k can be calculated as

$$\Delta_k w_{ij}^{[h]} = -\varepsilon \frac{\partial e_k}{\partial w_{ij}^{[h]}} \qquad \text{(A.1)}$$

By using the chain rule,

$$\frac{\partial \theta_k}{\partial w_{ij}^{[h]}} = \frac{\partial \theta_k}{\partial z_{jk}^{[h]}} \frac{\partial z_{jk}^{[h]}}{\partial w_{ij}^{[h]}}$$

$$= \frac{\partial \theta_k}{\partial z_{jk}^{[h]}} \frac{\partial}{\partial w_{ij}^{[h]}} \left[\sum_{i=1}^{N_{h-1}} w_{ij}^{[h]} y_{ik}^{[h-1]} + b_j^{[h]} \right] \quad \text{(A.2)}$$

$$= -\frac{\partial \theta_k}{\partial z_{jk}^{[h]}} y_{ik}^{[h-1]}$$

define

$$\delta_{jk}^{[h]} = -\frac{\partial \theta_k}{\partial z_{jk}^{[h]}} \quad \text{(A.3)}$$

then

$$\Delta_k w_{ij}^{[h]} = \varepsilon \, \delta_{jk}^{[h]} y_{ik}^{[h-1]} \quad \text{(A.4)}$$

for the output layer, i.e., h = L,

$$\theta_k = \sum_{j=1}^{N_L} (t_{jk} - y_{jk}^{[L]})^2 \quad \text{(A.5)}$$

$$y_{jk}^{[L]} = F(z_{jk}^{[L]}) \quad \text{(A.6)}$$

hence

$$\delta_{jk}^{[L]} = -\frac{\partial \theta_k}{\partial y_{jk}^{[L]}} \frac{\partial y_{jk}^{[L]}}{\partial z_{jk}^{[L]}} = (t_{jk} - y_{jk}^{[L]})\, F'(z_{jk}^{[L]}) \tag{A.7}$$

and

$$\Delta_k w_{ij}^{[L]} = \varepsilon (t_{jk} - y_{jk}^{[L]})\, F'(z_{jk}^{[L]})\, y_{ik}^{[L-1]} \tag{A.8}$$

for $h < L$

$$\delta_{jk}^{[h]} = -\frac{\partial \theta_k}{\partial z_{jk}^{[h]}} = -\sum_{l=1}^{N_{h+1}} \frac{\partial \theta_k}{\partial z_{lk}^{[h+1]}} \frac{\partial z_{lk}^{[h+1]}}{\partial z_{jk}^{[h]}} \tag{A.9}$$

the term $\partial z_{ik}^{[h+1]} / \partial z_{jk}^{[h]}$ can be expressed as

$$\frac{\partial z_{ik}^{[h+1]}}{\partial z_{jk}^{[h]}} = \frac{\partial z_{ik}^{[h+1]}}{\partial y_{jk}^{[h]}} \frac{\partial y_{jk}^{[h]}}{\partial z_{jk}^{[h]}} = F'(z_{jk}^{[h]}) \frac{\partial}{\partial y_{jk}^{[h]}} \left[\sum_{l=1}^{N_h} w_{li}^{[h+1]} y_{lk}^{[h]} + b_i^{[h+1]} \right] = F'(z_{jk}^{[h]})\, w_{ji}^{[h+1]} \tag{A.10}$$

and by definition

$$-\frac{\partial \ldots}{\partial z_{ik}^{[h+1]}} = \delta_{ik} \qquad \text{(A.11)}$$

therefore

$$\delta_{jk}^{[h]} = F'(z_{jk}^{[h]}) \sum_{i=1}^{N_{h+1}} \delta_{ik}^{[h+1]} w_{ji}^{[h+1]} \qquad \text{(A.12)}$$

Appendix B

DERIVATION OF THE ALGORITHM FOR ADAPTATION OF ACTIVATION FUNCTIONS

Based on the gradient descent method, the change in the parameter α of the sigmoid activation function of the jth neuron at the hth layer in an MFNN can be expressed as

$$\Delta_k \alpha_j^{[h]} = -\frac{\partial \theta_k}{\partial \alpha_j^{[h]}} \tag{B.1}$$

for h = L

$$\begin{aligned} \Delta_k \alpha_j^{[L]} &= -\frac{\partial \theta_k}{\partial y_{jk}^{[L]}} \frac{\partial y_{jk}^{[L]}}{\partial \alpha_j^{[L]}} \\ &= 2(t_{jk} - y_{jk}^{[L]}) F'_{\alpha}(z_{jk}^{[L]}, \alpha_j^{[L]}) \end{aligned} \tag{B.2}$$

where $F'_{\alpha}(z,\alpha)$ is the partial derivative of the activation function with respect to parameter α.

$$\Delta_k^{[h]} = -\frac{\partial \theta_k}{\partial \alpha_j^{[h]}} = -\sum_{l=1}^{N_{h+1}} \frac{\partial \theta_k}{\partial z_{lk}^{[h+1]}} \frac{\partial z_{lk}^{[h+1]}}{\partial \alpha_j^{[h]}} = -\sum_{l=1}^{N_{h+1}} \frac{\partial \theta_k}{\partial z_{lk}^{[h+1]}} \frac{\partial z_{lk}^{[h+1]}}{\partial y_{jk}^{[h]}} \frac{\partial y_{jk}^{[h]}}{\partial \alpha_j^{[h]}} \tag{B.3}$$

Since

$$-\frac{\partial \theta_k}{\partial z_{lk}^{[h+1]}} = \delta_{lk}^{[h+1]} \tag{B.4}$$

then

$$\frac{\partial z_{lk}^{[h+1]}}{\partial y_{jk}^{[h]}} = \frac{\partial}{\partial y_{jk}^{[h]}} \left[\sum_{l=1}^{N_h} w_{lj}^{[h+1]} y_{lk}^{[h]} + b_j^{[h+1]} \right] = w_{jl}^{[h+1]} \tag{B.5}$$

and

$$\frac{\partial y_{jk}^{[h]}}{\partial \alpha_j^{[h]}} = F'_{\alpha}(z_{jk}^{[h]}, \alpha_j^{[h]}) \tag{B.6}$$

$$\Delta_k \alpha_j^{[h]} = F'_{\alpha}(z_{jk}^{[h]}, \alpha_j^{[h]}) \sum_{l=1}^{N_{h+1}} \delta_{lk}^{[h+1]} w_{jl}^{[h+1]} \qquad \text{(B.7)}$$

AN FPGA IMPLEMENTATION OF MFNNS WITH QUANTIZED NEURONS

In this appendix, the FPGA design of a multilayer feedforward neural network with quantized neurons for XOR problem will be presented.

C.1 Design Overview

The overall design of an all-digital implementation of MFNN with quantized neurons can be divided into several major parts, including accumulation, shift operation, activation function, and timing control. The design will be a partly parallel, partly serial operated architecture. In other words, the operations within the same layer of the network are parallel, and the inter-layer operations are performed in serial from the first (or input) layer to the output layer because the outputs of the current layer usually are inputs to the next layer. The architecture of each layer are similar in MFNNs. A block diagram of the implementation structure of a typical MFNN is illustrated in **Fig.C.1**, where only one layer is shown.

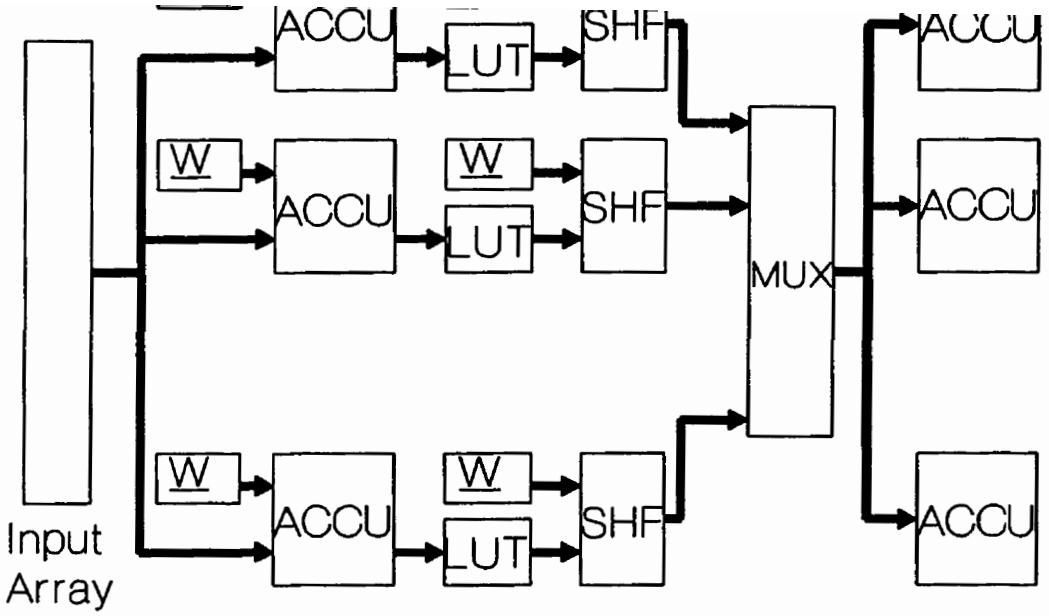


Figure C.1 Block Diagram of Digital Implementation Structure of an MFNN with quantized neurons

In the above structure, the ACCUs stand for functional blocks of accumulation; the SHFs are functional blocks of shift operation; the LUTs stand for the functional blocks of activation function; MUX is a multiplexer; and Ws are weights.

The operation of accumulation can be realized using an accumulator. The function of accumulator is to add up all input to the neuron. Each neuron will have a dedicated accumulator. The accumulator will have add/subtract

bias inputs, synchronous reset for new patterns, and clock enable to control the operation flow. The function of a shift block is to shift a weight by a certain number of bits, which is determined by the activation of the corresponding neuron. This block will also have parallel data loading, synchronous reset, and clock enable. Because for a particular connection weight, the number of bits to be shifted varies from pattern to pattern, the shift block should be able to detect how many bits will be shifted for each weights under different input patterns and control the shift operation as required. From Chapter 4, it can be seen that the maximum magnitude of the output of a neuron cannot exceed 1, therefore only right shift is involved in the operation. However, the output of each neuron is changing with different pattern presented to the network. Consequently, the number of bits to be shifted for a particular weight is not known in advance. The circuit has to be able to deal with this demand. As for the nonlinear activation function, it is usually implemented by look-up table using memories. Since in the model of MFNNs with quantized neurons, the activation function is a multi-step function, the implementation of this function can be simplified significantly. Due to its multi-step format, a group of magnitude comparators and a simple combinational logic can be put together to realize the desired feature of such activation functions. Weights can be stored in memories.

design package using the Unified Component Libraries. Creating FPGA designs with Viewlogic involves the following steps:

1. Enter the design with Viewdraw schematic editor, observing the Xilinx design requirements.

2. Test the functionality of the design. Run XSimMake to generate the ViewSim functional simulation netlist (VSM) file. After verifying that the logic design is functionally correct, proceed with the third step, design implementation.

3. Implement the FPGA design. Generate the placed and routed design automatically by executing the XMake program for an FPGA design or translate the design manually.

4. Simulate the timing of the design. Generate a ViewSim timing netlist by running the XSimMake program on the LCA (Logic Cell Array) file. Use the VSM output file for timing simulation.

The Viewlogic design methodology for FPGAs is illustrated by the flowchart in **Fig.C.2.**

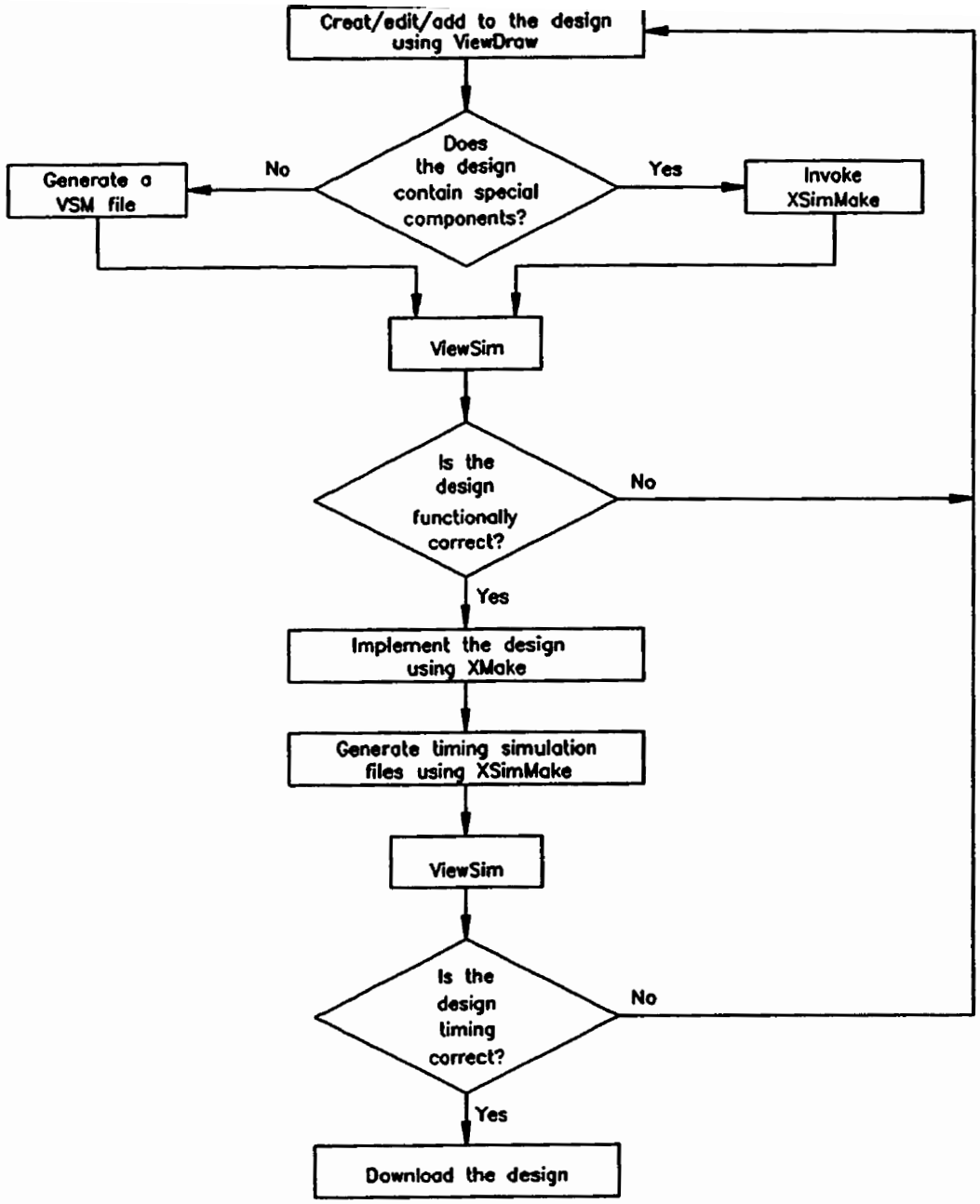


Figure C.2 Viewlogic Design Methodology for FPGAs

Fig.C.3 shows a top level schematic of an MFNN with 2 inputs, one (1) output, and one (1) hidden layer of two (2) hidden neurons for solving XOR problem. Several functional blocks were used in the design. The description of these blocks will be provided in the following sections. Some of them are user defined symbols, while the others are components directly from Xilinx libraries. In **Table C.1**, all symbols used in the top level design are listed with a brief explanation. Xilinx FPGA device 4013MQ208-5 was used in the design. The final design occupies 346 CLBs, that counts for 60% of the maximum number of 576 available CLBs.

The circuit was designed to function in the following way. After a pattern is presented to the input of the network, the bias of each neuron will be loaded into the corresponding accumulator; then the first layer accumulators will be working in parallel to add up all weighted inputs. Since the inputs are digital signals of 0 or 1, they can control the clock enable of the accumulator to determine whether the corresponding weight will be added or not. The elements of the input pattern are selected one by one in sequence by a multiplexer.

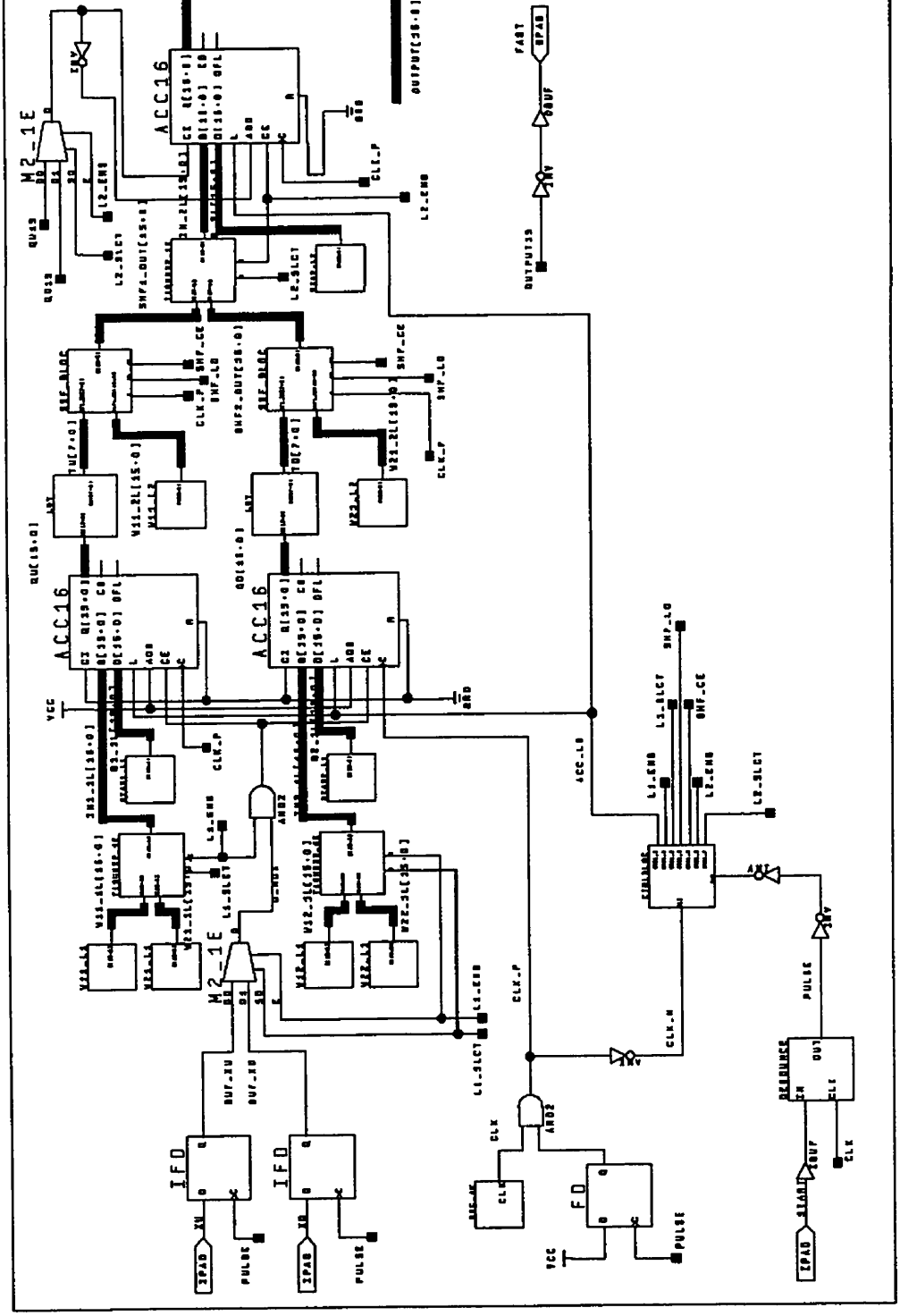


Figure C.3 Top level schematic for XOR problem

Symbols	Explanation
M2_1E	Two to one multiplexer
IFD	Input D Flip-Flop
IPAD	Input pad
$W_{ij}\_L_h$	Weight $w_{ij}^{[h]}$
$BIAS_j\_L_h$	Bias $b_j^{[h]}$
T16MUX2_1E	16 bit two to one multiplexer
AND2	Two inputs AND gate
ACC16	16 bit accumulator
LUT	Activation function
SHF_BLOC	Shift operation block
OSC_4K	Internal clock generator
DEBOUNCE	Start pulse generator
CTRLBLOC	Timing control block
OBUF	Output buffer
OPAD	Output pad
IBUF	Input buffer

After the accumulations at the first layer are finished, the sums of accumulators will be passed to the activation functions (block LUT) to produce the output of neurons. Because quantized neurons are used here, the output of an activation function has single term powers-of-two format, which will be used as an input to the block of shift operation (SHF_BLOC) to control the shift

to neurons at the next layer. The sign bit of the accumulator at the first layer will be used to select add/sub of the accumulator at the second layer. Due to the similarity of layers in an MFNN model, the same operations will be repeated in the consecutive layers.

In the following section, the details of major sub-circuit blocks will be described.

C.3 Sub-Circuit Blocks

In this section, the schematic design of the major functional blocks will be described. All of these blocks have been constructed using the primitive components from the Xilinx XC4000 library.

C.3.1 Accumulator - ACC16

ACC16, shown in **Fig.C.4**, is a 16-Bit Loadable Cascadable Accumulator with Carry-In, Carry-Out, and Synchronous Reset. The function of ACC16 is to take a sum of weighted inputs to each neuron, that means the output of the accumulator is the net input to the corresponding neuron.

Each neuron needs a dedicated accumulator such that parallel operation

the first layer, ACC16 operates only in add mode due to 0/1 inputs. Therefore, ADD will always be HIGH and CI will be LOW. Weights and biases, both in 16-bit 2's complement, are connected to B and D, respectively.

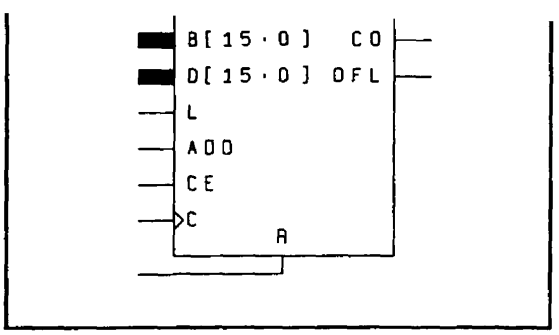


Figure C.4 16-Bit Accumulator ACC16

When used in a layer other than the first one, ACC16 will operate in add mode as well as subtract mode. The operation mode will be determined by the output of neurons in the previous layer together with the corresponding weights. If the sign bit of the output of the neuron in the previous layer is negative, then the value of the shifted version of the corresponding weight will be subtracted from the contents of the accumulator; otherwise, it will be added to that accumulator.

C.3.2 LUT - Implementation of the Activation Function

The input to the LUT is the output from the accumulator. The output of the LUT is the activation of the quantized neuron in STPT format, i.e., there is only one bit is "on" (logic high) in the output. This functional block has been

Fig. C.5 is the schematic of LUT when M=4. The schematic for symbol CMPRTR is shown in **Fig. C.6**. The five comparators on the top are used for comparisons between input data and positive thresholds, while the other five at the bottom are for comparison with negative thresholds.

The input-output relationship of the combinatorial logic can be described as in **Table C.2**. The five positive and negative thresholds are obtained by using the method described in Chapter 4 and listed in **Table C.3**.

Table C.2 Combinatorial Logic in LUT Block

INPUTS			OUTPUTS
IN15	P4-P0	N4-N0	Position of "1"
0	11111	xxxxx	
1	xxxxx	11111	OUT7
0	01111	xxxxx	
1	xxxxx	01111	OUT6
0	00111	xxxxx	
1	xxxxx	00111	OUT5
0	00011	xxxxx	
1	xxxxx	00011	OUT4
0	00001	xxxxx	
1	xxxxx	00001	OUT3

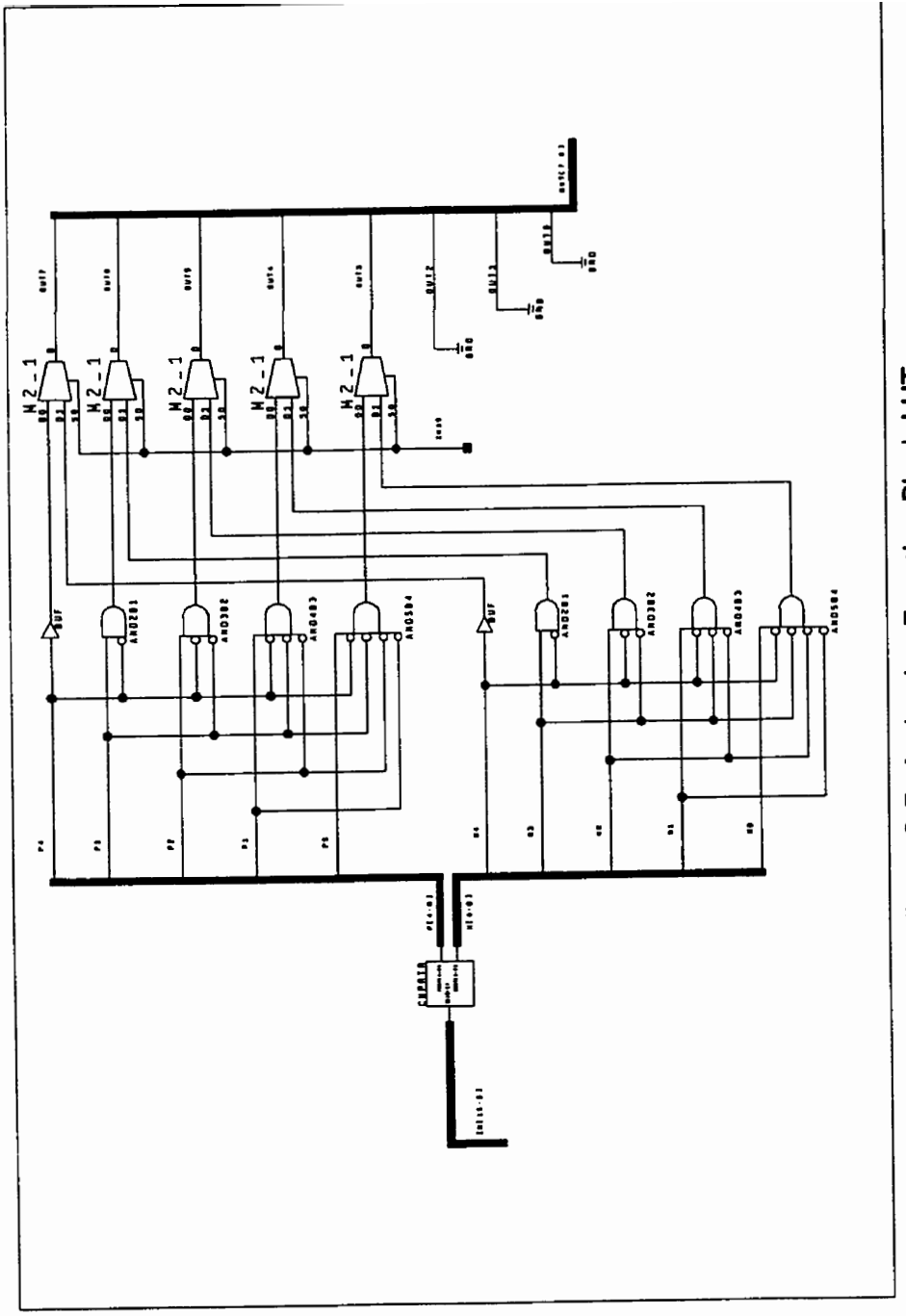


Figure C.5 Activation Function Block LUT

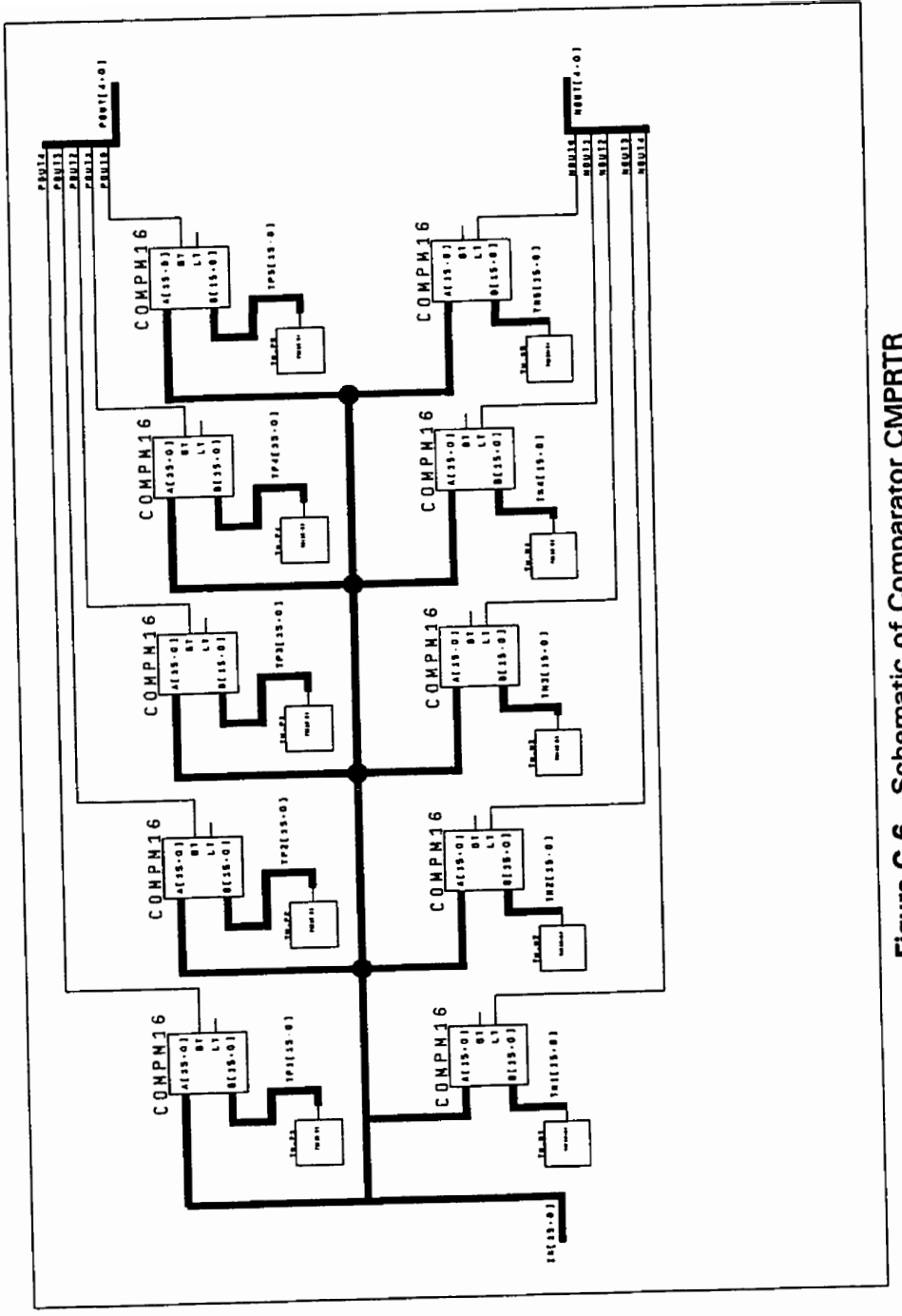


Figure C.6 Schematic of Comparator CMPRTR

THRESHOLDS	2'S COMPLEMENT VALUES
TH_P1	0001000000011010
TH_P2	0000010111000110
TH_P3	0000001011000011
TH_P4	0000000101011110
TH_P5	0000000001110101
TH_N1	1110111111100110
TH_N2	1111101000111010
TH_N3	1111110100111101
TH_N4	1111111010100010
TH_N5	1111111110001011

C.3.3 SHF_BLOC - Implementation of Shift Operation

The function of the shift block is to shift a weight according to the activation of the corresponding neuron, which is of the STPT format, instead of doing multiplication. As mentioned before, because the output of a neuron is different under different input patterns, the number of bits to be shifted in the corresponding weight varies from one pattern to another. Therefore, the shift block must be able to detect and control the actual number of bits to be shifted. This is done by using two shift registers, one is used to control how many bits will be shifted while the other is used to do the actual shift operation of weights. **Fig. C.7** shows the implementation of the shift block where the

enable (L), shift left/right (LEFT), and synchronous reset (R).

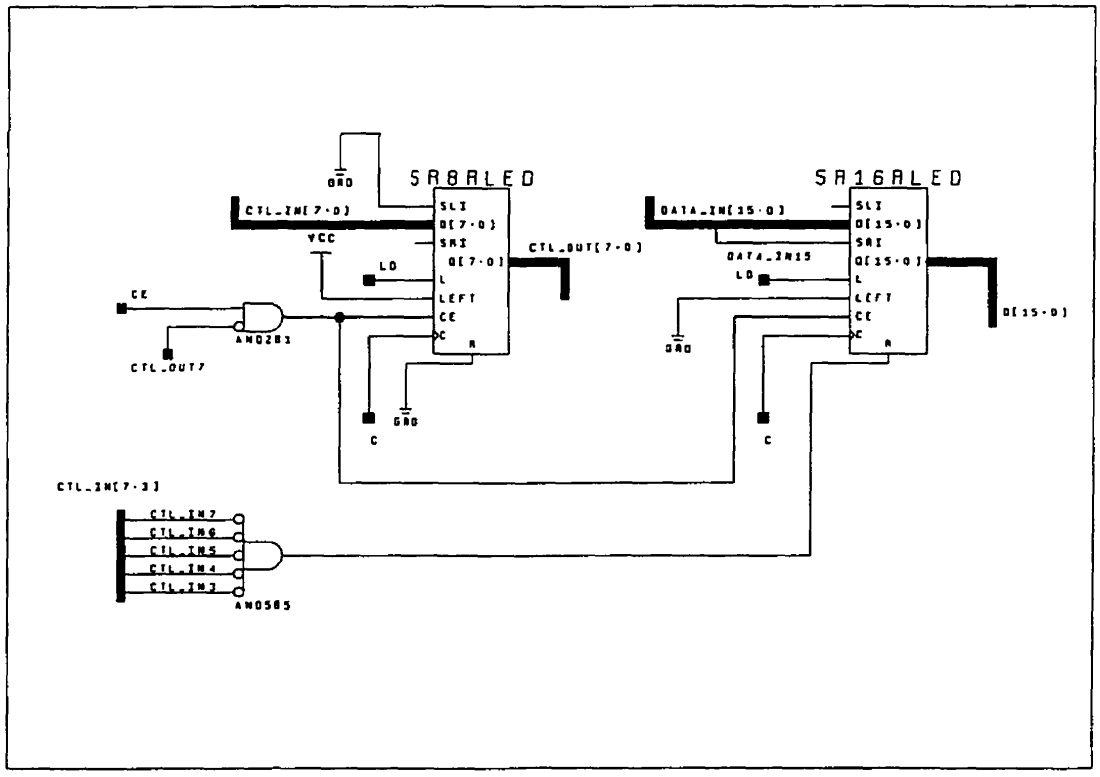


Figure C.7 Implementation of Shift Operation

The 8-bit shift register SR8RLED is used to control the shift operation while the 16-bit shift register SR16RLED conducts the actual weight shifting. The SR8RLED will keep shifting left while the SR16RLED is shifting right until the most significant bit (MSB) in the SR8RLED becomes 1. At that point, both

SR8RLED is all zero, the output of the [illegible] synchronously at the next clock pulse.

C.3.4 CTRLBLOC - Implementation of the Control Block

The function of the control block CTRLBLOC is to provide timing signals to the circuit. These signals usually will control the Enable input, the Data Load input, and Set/Reset input of functional components. Because an MFNN operates from the input layer to the output layer, the CTRLBLOC will allow each layer to operate only when all information have come available to that layer.

Fig. C.8 shows the implementation of the control block CTRLBLOC, where CB4CE is a 4-stage, 4-bit, synchronous, clearable, cascadable binary counter.

From the top level design given in Section C.2, it can be seen that the whole circuit operates synchronously, i.e., the states of the circuit make changes only during clock transitions. However, the sequence of operations and the time at which a block can operate will be determined by control signals. These signals include ACC_LD, L1_ENB, L1_SLCT, SHF_LD, SHF_CE, L2_ENB, and L2_SLCT, which are summarized in **Table C.3.**

SIGNALS	OUTPUTS OF CTRLBLOC	FUNCTIONS
ACC_LD	CTRL_1	Accumulators load signal
L1_ENB	CTRL_2	Layer 1 operations enable
L1_SLCT	CTRL_3	Layer 1 multiplexer select signal
SHF_LD	CTRL_4	Shift blocks load signal
SHF_CE	CTRL_5	Shift blocks clock enable
L2_ENB	CTRL_6	Layer 2 operations enable
L2_SLCT	CTRL_7	Layer 2 multiplexer select signal

All control signals are active High. Each block can act only during the active period of the corresponding control signal. The order of appearance of these signals is the same as the sequence in which they are listed in **Table C.3.**

C.3.5 Weights and Biases

Weights and biases used in the FPGA design are listed in **Table C.4,** along with their 16-bit fixed-point 2's complement representations.

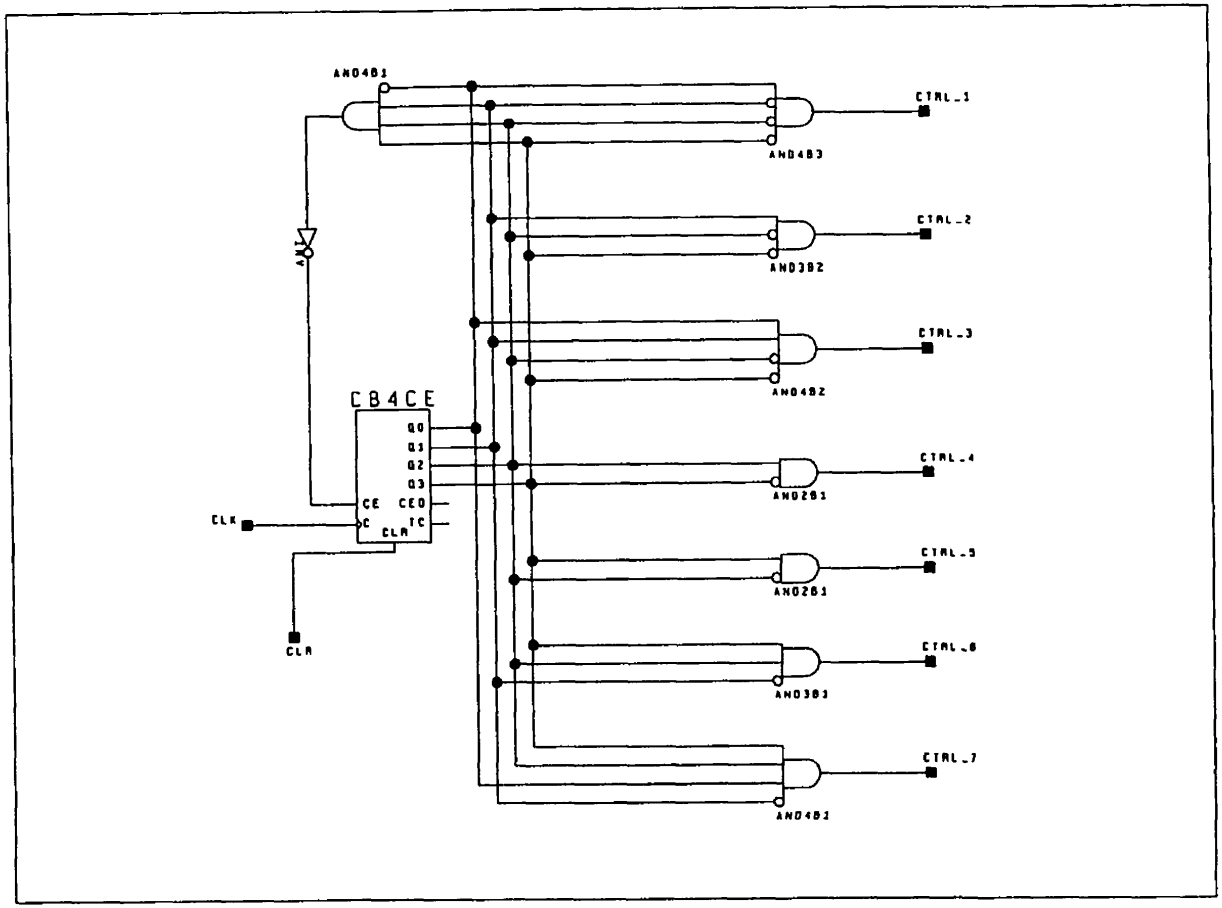


Figure C.8 Schematic of Control Block CTRLBLOC

Table C.4 Representations of Weights and Biases

Symbols	Values	2's Complement Representations
w11[1]	2.36474609375000	0010010111010110
w12[1]	1.56982441875000	0001100100011110
w21[1]	2.42138671875000	0010011010111110
w22[1]	1.56982421875000	0001100100011110
w11[2]	3.65307617187500	0011101001110011
w21[2]	-3.72949218750000	1100010001010100
b1[1]	-1.03979492187500	1110111101011101
b2[1]	-1.86059570312500	1110001000111011
b1[2]	-1.11254882812500	1110111000110011

C.4 Design Simulation

Both functional and timing simulations have been conducted on the top level schematic design. Functional simulation is used to verify the logic relationship under normal unit delay of each component while timing simulation is used to verify the logic correctness under the worst situation of gate delays. Functional simulations use unrouted design and timing simulation based on the routed design.

11. The desired output is 0, 1, 1, and 0, respectively. For a neuron at the output layer, if the net input before activation is negative, then the output of that neuron will be zero because a hardlimiter function can be applied to produce binary output. On the other hand, if the net input before activation is positive, the output of that neuron will be logic one.

C.4.1 Functional Simulation Results

Design simulations have been conducted using Xilinx's ViewSim. The results are provided below.

1) XU = 0, XD = 0

When both inputs are 0, the signal OUTPUT[15:0] = 1110111000110011 which is less than TH_N1 = 1110111111100110. Thus, the output of the network is logic zero (Low), the same as expected. The simulation waveform is depicted in **Fig. C.9.**

2) XU = 0, XD = 1

In this case, two inputs are different, the output of the network should be logic one (High). Look at the simulation waveform given in **Fig. C.10,** the signal OUTPUT[15:0] = 0001001001000101 which is greater than TH_P1 = 0001000000011010. Therefore, the output of the network is indeed

3) XU = 1, XD = 0

Similar to the above case 2), the two inputs are new different, so the output of the network should be logic one (High). From **Fig. C.11**, the simulation waveform, it can be seen that the signal OUTPUT[15:0] = 0001001001000101 which is greater than TH_P1 = 0001000000011010. Therefore, the output of the network is indeed logic one (High).

4) XU = 1, XD = 1

Now both inputs are 1, which is a similar situation to case 1), the output of the network is expected to be logic zero (Low). Observe the simulation waveform in **Fig.C.12**, the signal OUTPUT[15:0] = 1110110011111001 which is still less than TH_N1 = 1110111111100110. With this OUTPUT[15:0] as the net input to the output neuron before activation, the output of the network is logic Low (zero), showing the correct result.

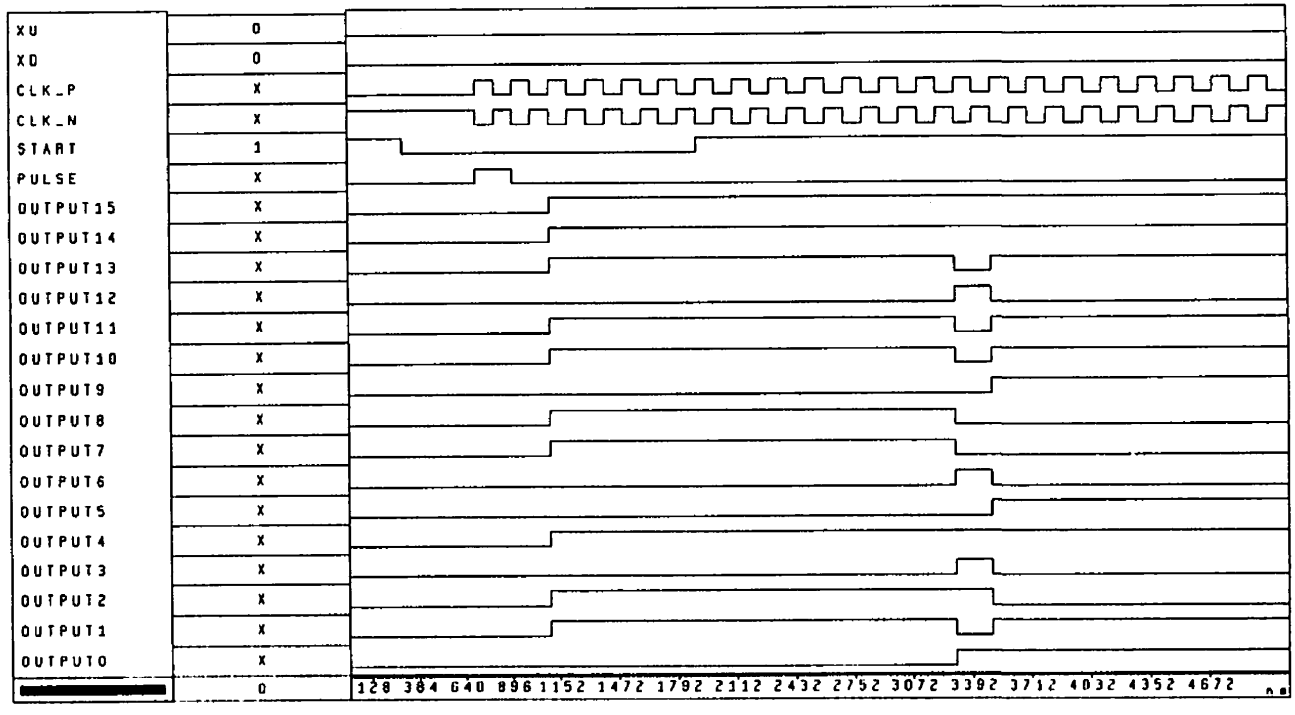


Figure C.9 Functional Simulation Result When XU = 0 and XD = 0

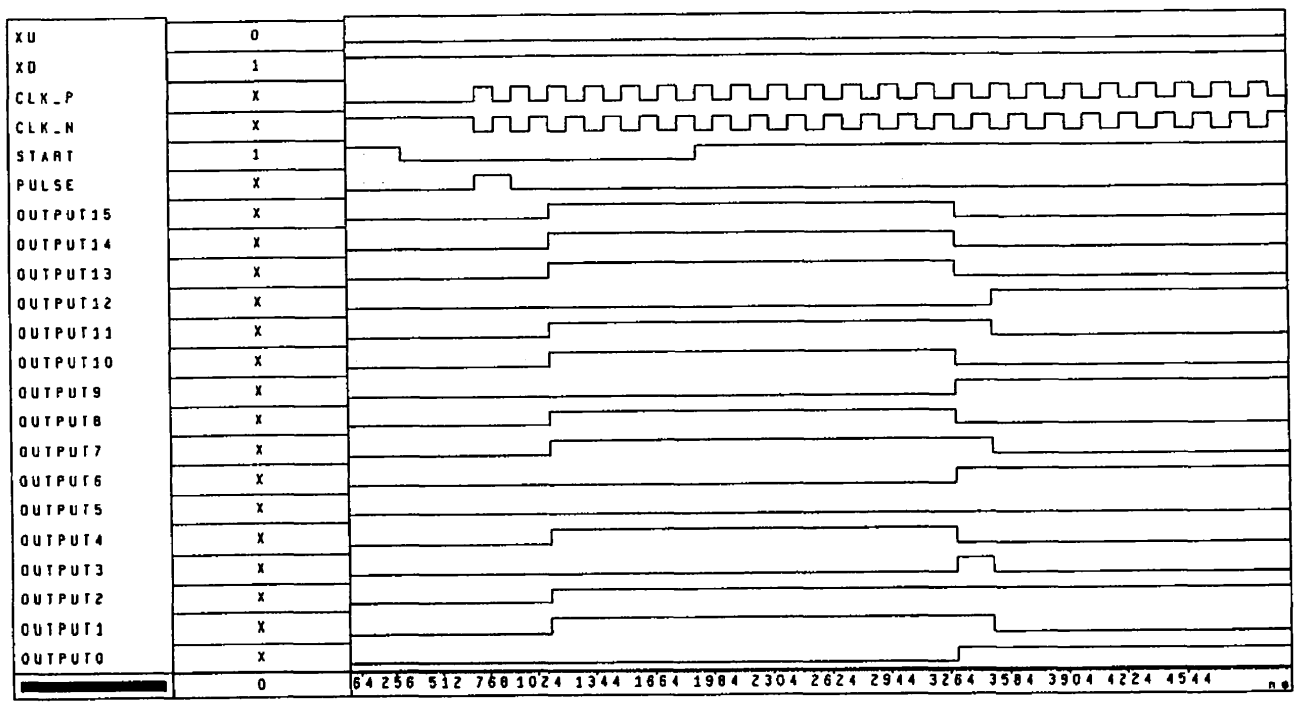


Figure C.10 Functional Simulation Result When XU = 0 and XD = 1

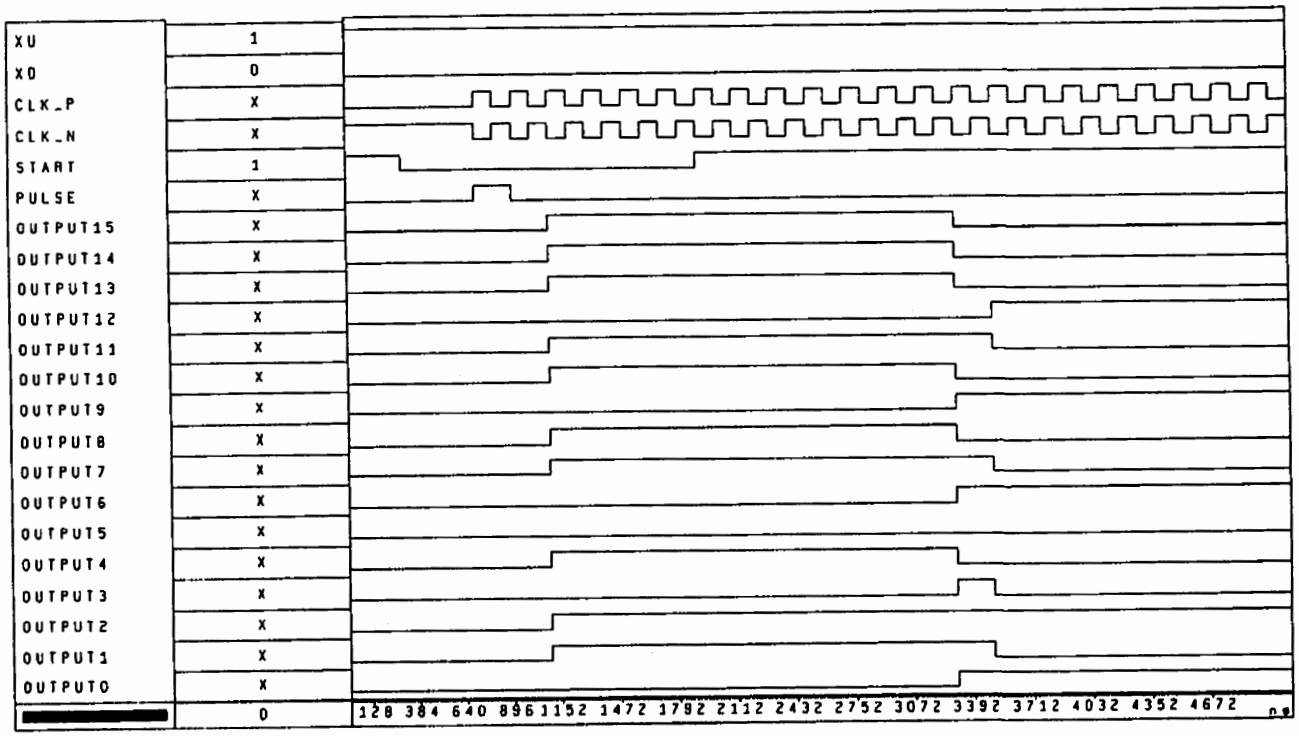


Figure C.11 Functional Simulation Result When XU = 1 and XD = 0

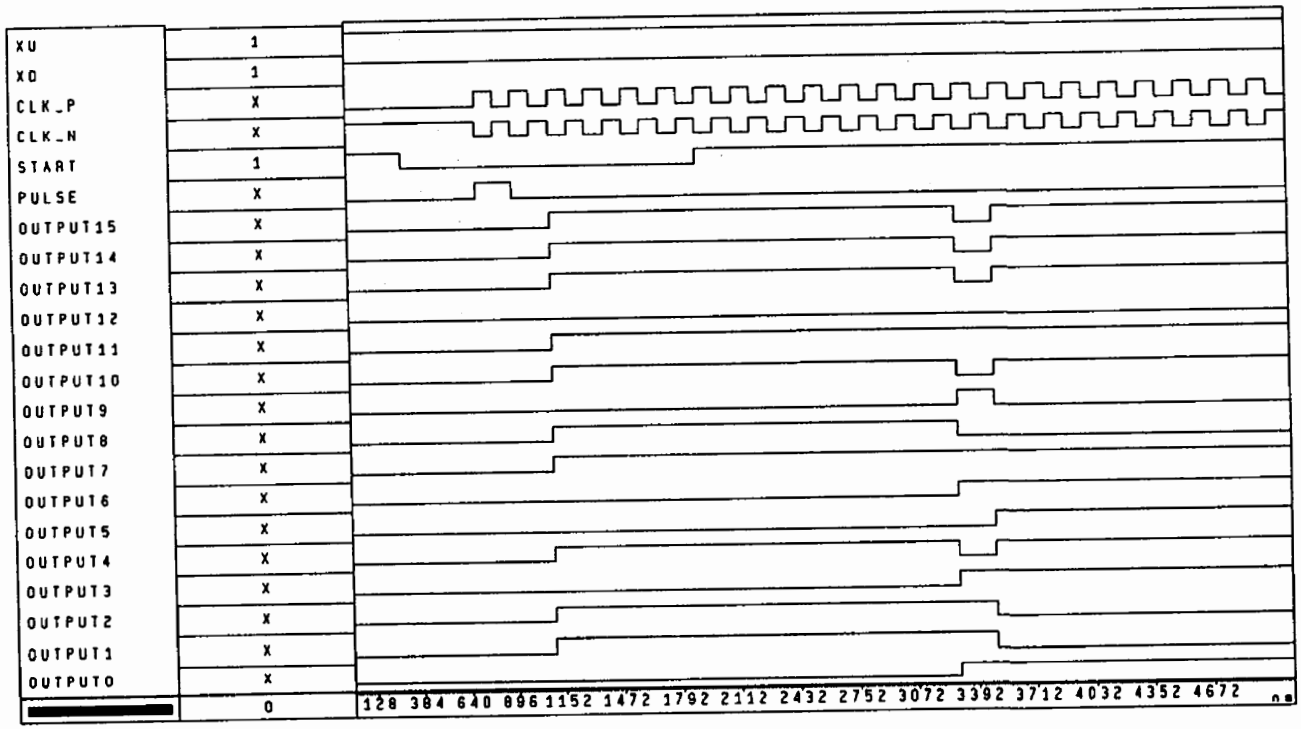


Figure C.12 Functional Simulation Result When XU = 1 and XD = 1

Timing simulations have been carried out [illegible]

The same logic results as in the functional simulations were obtained and are shown in **Fig.C.13-C.16.** The only difference is the amount of delays.

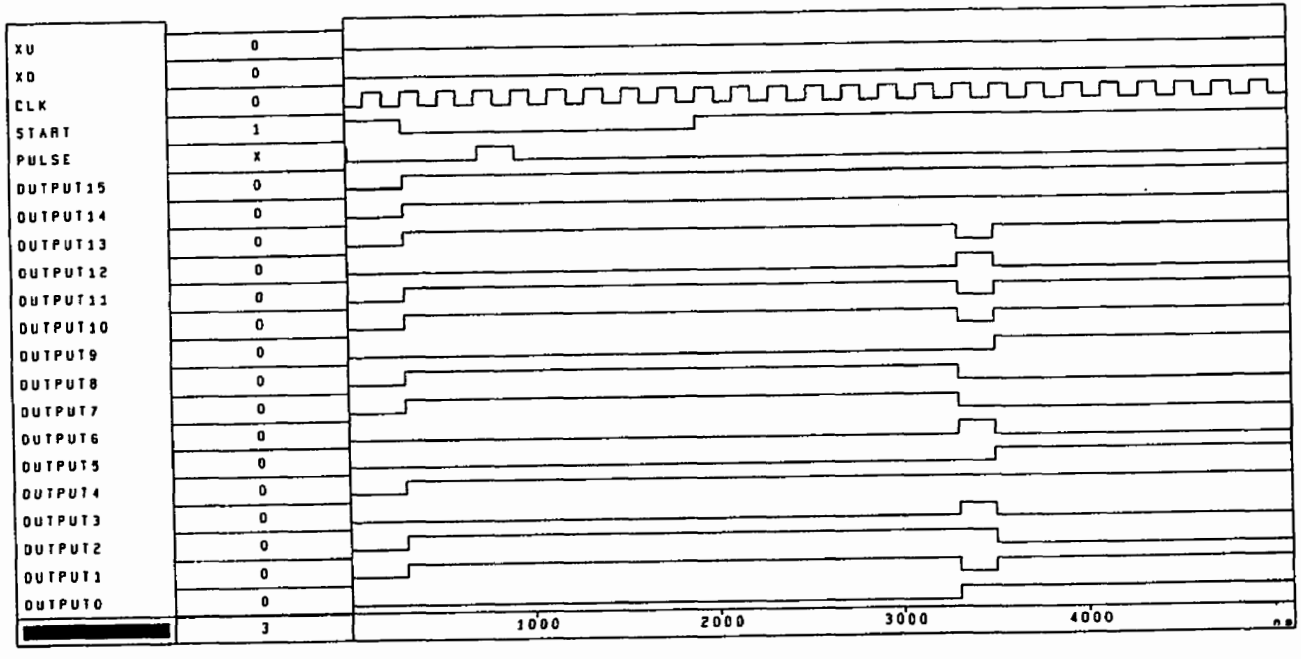


Figure C.13 Timing Simulation Result When XU = 0 and XD = 0

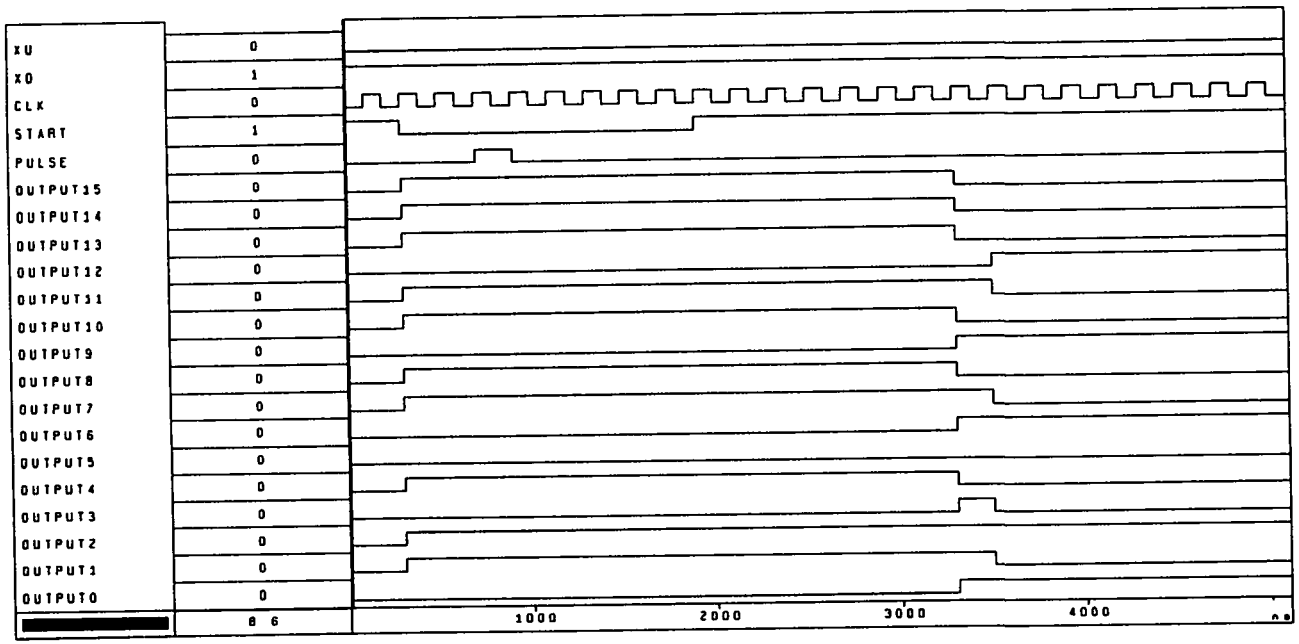


Figure C.14 Timing Simulation Result When XU = 0 and XD = 1

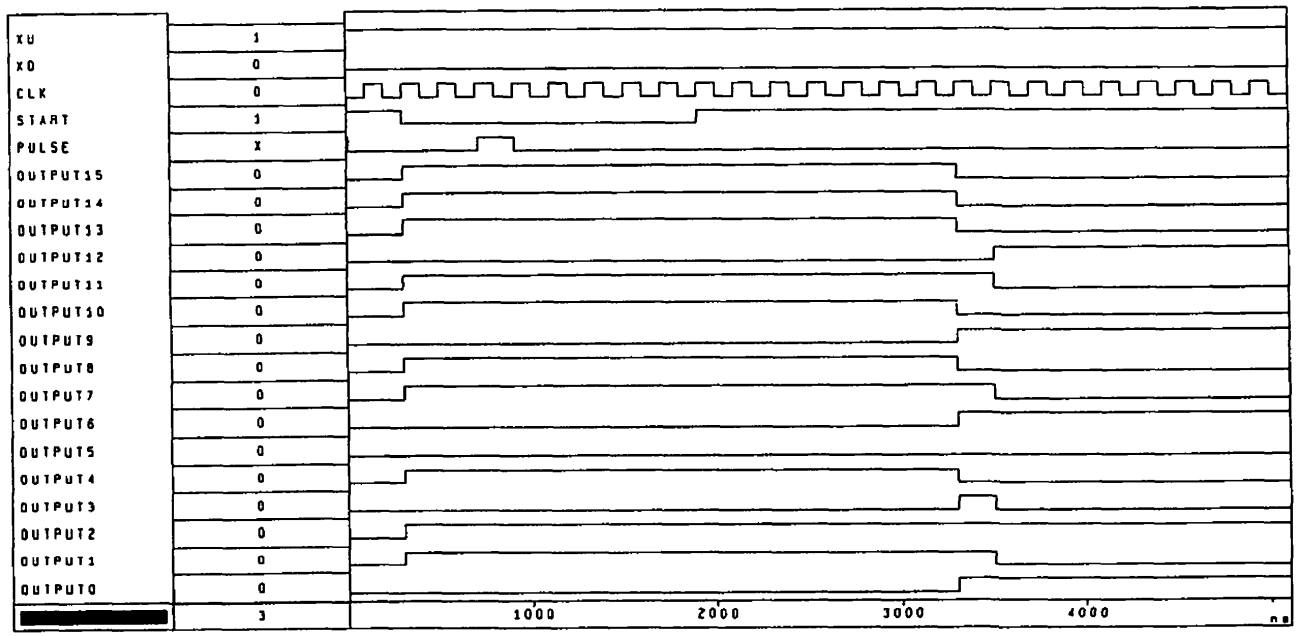


Figure C.15 Timing Simulation Result When XU = 1 and XD = 0

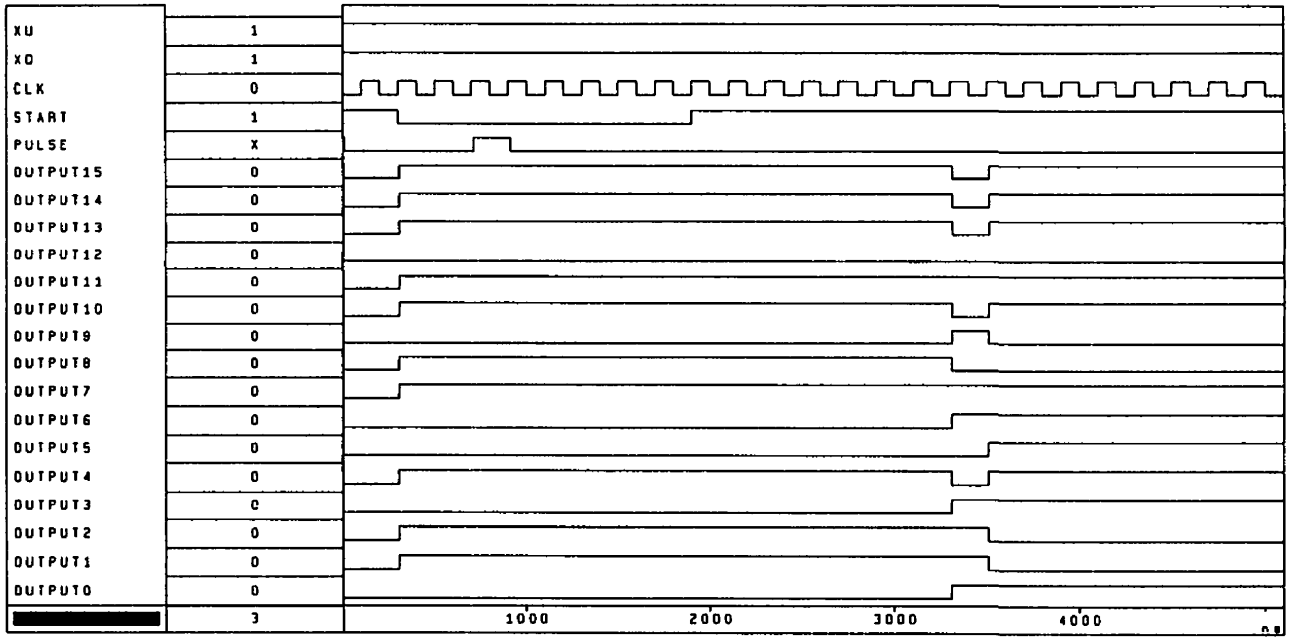


Figure C.16 Timing Simulation Result When XU = 1 and XD = 1

Appendix D

VHDL CODES FOR HARDWARE IMPLEMENTATION SCHEMES

Table D.1 VHDL code for shift operation

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

entity SHIFTER is
        port( a:    in    std_logic_vector(7 downto 0);
              ct:   in    std_logic_vector(4 downto 0);
              z:    out   std_logic_vector(11 downto 0));
end SHIFTER;

architecture BEHAVIORAL of SHIFTER is
begin
        process(a, ct)
        begin
                case ct is
                        when "00000" =>
                                z<="000000000000";
                        when "00001" =>
                                z<=a(7)&a(7)&a(7)&a(7)&a(7 downto 0);
                        when "00010" =>
                                z<=a(7)&a(7)&a(7)&a(7 downto 0)&'0';
```

```
                z<=a(7)&a(7)&a(7 downto 0)&"00";
            when "01000" =>
                z<=a(7)&a(7 downto 0)&"000";
            when "10000" =>
                z<=a(7 downto 0)&"0000";
            when others =>
                z<="----------";
        end case;
    end process;

end BEHAVIORAL;
```

Table D.2 VHDL description of the multistep activation function used in MFNNs with quantized neurons

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity PIECEWISE is
        port(  a:       in      std_logic_vector(15 downto 0);
               c:       out     std_logic_vector(4 downto 0));
end PIECEWISE;

architecture BEHAVIORAL of PIECEWISE is

constant p0:        std_logic_vector(15 downto 0):="0001000000011010";
constant p1:        std_logic_vector(15 downto 0):="0000010111000110";
constant p2:        std_logic_vector(15 downto 0):="0000001011000011";
constant p3:        std_logic_vector(15 downto 0):="0000000101011110";
constant p4:        std_logic_vector(15 downto 0):="0000000001110101";
constant n0:        std_logic_vector(15 downto 0):="1110111111100110";
constant n1:        std_logic_vector(15 downto 0):="1111101000111010";
constant n2:        std_logic_vector(15 downto 0):="1111110100111101";
constant n3:        std_logic_vector(15 downto 0):="1111111010100010";
constant n4:        std_logic_vector(15 downto 0):="1111111110001011";

begin

        process(a)
        begin
                case a(15) is
                        when '0' =>
                                if(a>=p0) then
```

```
                elsif(a>=p1) then
                        c<="01000";
                elsif(a>=p2) then
                        c<="00100";
                elsif(a>=p3) then
                        c<="00010";
                elsif(a>=p4) then
                        c<="00001";
                else
                        c<="00000";
                end if;
        when others =>
                if(a<=n0) then
                        c<="10000";
                elsif(a<=n1) then
                        c<="01000";
                elsif(a<=n2) then
                        c<="00100";
                elsif(a<=n3) then
                        c<="00010";
                elsif(a<=n4) then
                        c<="00001";
                else
                        c<="00000";
                end if;
        end case;
end process;

end BEHAVIORAL;
```

Table D.3 VHDL description of the simplified sigmoid activation function

```
library IEEE, DW02;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use DW02.DW02_components.all;

entity ssaf is
        port(x:         in std_logic_vector(15 downto 0);
             z: out std_logic_vector(7 downto 0));
end ssaf;

architecture behavior of ssaf is

signal x_shft: std_logic_vector(15 downto 0);
signal x_shft_low: std_logic_vector(7 downto 0);
signal x_shft_low_sqr: std_logic_vector(15 downto 0);
signal z_long: std_logic_vector(8 downto 0);
signal control: std_logic;


begin
        control <= '1';
        x_shft <= x(15)&x(15 downto 1);
        x_shft_low <= x_shft(7 downto 0);
        -- x_shft_low_sqr <= x_shft_low * x_shft_low;
  U1: DW02_mult
      generic map(A_width => 8, B_width => 8)
      port map(A => x_shft_low, B => x_shft_low,
               TC => control, PRODUCT => x_shft_low_sqr);

piecewise: process(x, z_long)
```

```
        if(x > =2) then
                z< ="01111111";
        elsif(x< =-2) then
                z< ="10000000";
        else
                z < = z_long(7 downto 0);
        end if;
end process;

second_order: process(x_shft, x_shft_low_sqr, x)

begin
        case x(15) is
                when '0' = >
                        z_long < = x_shft(7 downto 0)&'0' -
                                x_shft_low_sqr(15 downto 7);
                when others = >
                        z_long < = x_shft(7 downto 0)&'0' +
                                x_shft_low_sqr(15 downto 7);
        end case;
end process;

end behavior;
```

Ackley, D.H., D.E., Hinton, G.E., and William, R.J., A learning algorithm for Boltzmann machines, in Anderson, J.A. and Rosenfeld, E. (Eds.), *Neurocomputing*, Cambridge, MA: MIT Press, 1988.

Alippi, C., Piuri, V., and Sami, M., Sensitivity to errors in artificial neural networks: A behavioral approach, *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, Vol. 42, No. 6, June 1995, pp. 358-361.

Ansari, N., Hou, E.S.H., and Yu, Y., A new method to optimize the satellite broadcasting schedules using the mean field annealing of a Hopfield neural network, *IEEE Trans. Neural Networks*, vol.6, no.2, pp.470-483, Mar. 1995.

Antognetti, P., and Milutinović, V. (Eds.), *Neural Networks: Concepts, Applications, and Implementations*, Vol.I, III, IV, Englewood Cliffs, NJ: Prentice-Hall, 1991.

Atlas, L.E and Suzuki, Y., Digital systems for artificial neural networks, *IEEE Circuits and Devices Magazine*, pp. 20-24, Nov. 1989.

Baldi, P., Gradient descent learning algorithm overview: A general dynamical systems perspective, *IEEE Trans. Neural Networks*, Vol. 6, pp. 182-195, Jan. 1995.

Benvenuto, N., Marchesi, M., Orlandi, G., Piazza, F., and Uncini, A., Design of multi-layer neural networks with powers-of-two weights, *Proc. IEEE Int. Symposium on Circuit and Systems*, pp. 2951-2954, New Orleans, May 1990.

Carpenter, G.A., and Grossberg, S., A massively parallel architecture for a self-organizing neural pattern recognition machine, *Computer Vision, Graphics, and Image Processing*, vol. 37, pp. 54-115, 1983.

recognition codes for analog output patterns, *Applied Optics*, vol. 26, pp. 4919-4930, Dec. 1, 1987.

Carpenter, G.A., and Grossberg, S., Art 3 hierarchical search: Chemical transmitters in self-organizing pattern recognition architectures," in *Proc. Int. Joint Conf. on Neural Networks*, vol. 2, pp. 30-33, Washington, D.C., Jan. 1990.

Carpenter, G.A., Grossberg, S., Markuzon, N., Reynolds, J.H., and Rosen, D.B., Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps, *IEEE Trans. Neural Networks*, vol.3, no.4, pp.698-713, 1992.

Carpenter, G.A., Grossberg, S., and Reynolds, J.H., A fuzzy ARTMAP nonparametric probability estimator for nonstationary pattern recognition problems, *IEEE Trans. Neural Networks*, vol.7, no.6, pp.1330-1336, 1996.

Cavilia, D.D., Valle, M., and Bisio, G.M., Effects of weight discretization on the back propagation learning method: algorithm design and hardware realization, *Proc. Int. Joint Conf. on Neural Networks*, vol. 2, pp. 631-637, San Diego, CA, 1990.

Choi, J., Bang, S.H., and B.J. Sheu, A programmable analog VLSI neural networks processor for communication receivers, *IEEE Trans. Neural Networks*, vol.4, no.3, pp.484-495, May 1993.

Choong, P.L., deSilva, C.J.S., Dawkins, H.J.S., and Sterrett, G.F., Entropy maximization networks: An application to breast cancer prognosis, *IEEE Trans. Neural Networks*, vol.7, no.3, pp.568-577, May 1996.

Ergezinger, S. and Thomsen, E., An accelerated learning algorithm for multilayer perceptrons: Optimization layer by layer, *IEEE Trans. Neural Networks*, vol.6, no.1, pp.31-42, Jan. 1995.

Freeman, J.A., and Skapura, D.M., *Neural Networks: Algorithms, Applications,*

Company, 1991.

Fukushima, K., Cognitron: A self-organizing multilayered neural network, *Biolog. Cybernetics*, vol. 20, pp.121-136, 1975.

Fukushima, K., Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, *Biolog. Cybernetics*, vol. 36, pp. 193-202, 1980.

Grossman, T., Meir, R., and Domany, E., Learning by choice of internal representations, in *Advances in Neural Information Processing Systems I*, pp.73-80, Touretzky, D. (Eds.), Morgan Kaufmann, San Mateo, 1989.

Hebb, D.O., *The Organization of Behavior*, New York: John Wiley & Sons, 1949.

Hinton, G.E., and Sejnowski, T.J., Learning and relearning in Boltzmann machines, in Rumelhart, D.E. and McClelland, J.L. (Eds.), *Parallel Distributed Processing*, vol.1, ch.7, Cambridge, MA: MIT Press, 1986.

Hopfield, J.J., Neural networks and physical systems with emergent collective computational abilities, *Proc. Natl. Acad. Sci. USA*, vol. 79, pp.2554-2558, April 1982.

Hopfield, J.J., Neurons with graded response have collective computational propoerties like those of two-state neurons, *Proc. Natl. Acad. Aci. USA*, vol.81, 3088-3092, May 1984.

Hornik, K., Stinchcombe, M., and White, H., Multilayer feedforward networks are universal approximators, *Neural Networks*, vol.2, pp. 359-366, 1989.

Hornik, K., Approximation capabilities of multilayer feedforward networks, *Neural Networks*, vol. 4, pp. 251-257, 1991.

Jacobs, R.A., Increased rates of convergence through learning rate adaptation,

Jenkins, J.H., *Designing with FPGAs and CPLDs*, Englewood Cliffs, NJ: PTR Prentice Hall, 1994.

Kechriotis, G., Zervas, E., and Manolakos, E.S., Using recurrent neural networks for adaptive communication channel equalization, *IEEE Trans. Neural Networks*, vol.5, no.2, pp.267-278, Mar. 1994.

Kim, Y.C. and Shanblatt, M.A., An Implementable Digital Multilayer Neural Network, *Proc. IJCNN*, Baltimore, vol.2, pp.594-600, 1992.

Kirkpatrick, S., Jr., Gelatt, C.D., and Vecchi, M.P., Optimization by simulated annealing, in Anderson, J.A. and Rosenfeld, E. (Eds.), *Neurocomputing*, MIT Press, Cambridge, MA, pp. 554-568, 1988.

Kohonen, T., Self-organized formation of topologically correct feature maps, *Biolog. Cybernetics*, vol. 43, pp. 59-69, 1982.

Kohonen, T., *Self-Organization and Associative Memory*, Springer-Verlag, Berlin, 1984.

Kosko, B., Adaptive bidirectional associative memories, *Appl. Optics*, vol. 26, pp. 4947-4960, Dec. 1, 1987.

Kosko, B., Bidirectional associative memories, *IEEE Tran. Systems, Man, and Cybernetics*, vol. 18, no.1, pp.49-60, 1988.

Kruschke, J.K., and Movellan, J.R., Benefits of gain: Speeded learning and minimal hidden layers in back-propagation networks, *IEEE Trans. Systems, Man, and Cybernetics*, vol. 21, pp. 273-280, Mar. 1991.

Kung, S.Y., *Digital Neural Networks*, Englewood Cliffs, NJ: PTR Prentice Hall, 1993.

Kung, S.Y. and Taur, J.S., Decision-based neural networks with signal/image classification applications, *IEEE Trans. Neural Networks*, vol.6, no.3,

Kwan, H.K., Simple sigmoid-like activation function suitable for digital hardware implementation, *IEE Electronics Letters*, vol. 28, no. 15, pp. 1379-1380, July 16, 1992.

Kwan, H.K. and Chan, C.L., Design of multidimensional spherically symmetric and constant group delay recursive digital filters with sum of powers-of-two coefficients, *IEEE Trans. on Circuits and Systems*, vol. 37, pp. 1027-1035, August 1990 and pp. 1580, December 1990.

Kwan, H.K. and Chan, C.L., Circularly symmetric two-dimensional multiplierless FIR digital filter design using an enhanced McClellan transformation, *Proc. Inst. Elec. Eng. (part G: Circuits, Devices, and Systems)*, vol. 136, no. 3, pp. 129-134, June 1989.

Kwan, H.K., and Tang, C.Z., Multilayer feedforward neural networks with powers-of-two weights and piecewise activation functions, *Proc. 1992 Canadian Conference on Electrical and Computer Engineering*, Toronto, Sept. 1992, pp. TM6.14.1-4.

Kwan, H.K., and Tang, C.Z., Designing multilayer feedforward neural networks using simplified sigmoid activation functions and one-powerss-of-two weights, *IEE Electronics Letters*, vol. 28, no.25, pp. 2343-2345, Dec. 3, 1992.

Kwan, H.K., and Tang, C.Z., A design method for multilayer feedforward neural networks for simple hardware implementation, *Proc. 1993 IEEE International Symposium on Circuits and Systems*, Chicago, U.S.A., May 1993, pp.2363-2366.

Kwan, H.K., and Tang, C.Z., Multiplierless multilayer feedforward neural network design suitable for continuous input-output mapping, *IEE Electronics Letters*, vol. 29, no. 14, pp. 1259-1260, July 8, 1993.

Kwan, H.K., and Tang, C.Z., Multiplierless Multilayer Feedforward Neural Networks, *Proc. 36th Midwest Symposium on Circuits and Systems*,

Kwan, H.K., and Tang, C.Z., A multilayer feedforward neural network model for digital hardware implementation, *Proc. IEEE International Symposium on Circuits and Systems*, London, UK, vol.6, pp.343-345, May 30-June 2, 1994.

Kwan, H.K., Wang, Z., and Soltis, J.J., Method for generating partially correlated vectorsets for neural network simulations, *Int. J. Electron.*, vol. 74, no. 4, pp.523-528, 1993.

Lang, K.J., Waibel, A.H., and Hinton, G.E., A time-delay neural network architecture for isolated word recognition, in Sanchez-Sinencio, E. and Lau, C. (Eds.), *Artificial Neural Networks: Paradigms, Applications, and Hardware Implementations*, New York: IEEE Press, 1992.

Lau, C. (Eds.), *Neural Networks: Theoretical Foundations and Analysis*. New York: IEEE Press, 1992.

Lee, Y., Oh, S.H., and Kim, M.W., The effect of initial weights on premature saturation in back-propagation learning, *Proc. International Joint Conference on Neural Networks*, vol. 1, pp. 765-770, Seattle, USA, 1991.

Levin, E., Hidden control neural architecture modeling of nonlinear time varying systems and its applications, *IEEE Trans. Neural Networks*, vol.4, no.1, pp.109-116, Jan. 1993.

Lewis, F.L., Yesildirek, A., and Liu, K., Multilayer neural-net robot controller with guaranteed tracking performance, *IEEE Trans. Neural Networks*, vol.7, no.2, pp.388-399, Mar. 1996.

Lim, Y.C. and Constantinides, A.G., Linear phase FIR digital filter without multipliers, in *Proc. IEEE Symp. Circuits Syst.*, Tokyo, Japan, July 1979, pp. 185-188.

Lim, Y.C., Parker, S.R., and Constantinides, A.G., Finite wordlength FIR filter

Trans. Acoust., Speech, Signal Processing, vol. 30, pp. 661-664, Aug. 1982.

Lim, Y.C. and Parker, S.R., FIR filter design over a discrete powers-of-two coefficient space, *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 583-591, June 1983.

Lim, Y.C. and Parker, S.R., Discrete coefficient FIR digital filter design based upon an LMS criteria, *IEEE Trans. on Circuits and Systems*, vol. 30, no. 10, pp. 7233-7239, Oct. 1983.

Lippmann, R.P., An introduction to computing with neural nets, *IEEE ASSP Magazine*, vol. 4, pp. 4-22, Apr. 1987.

Marchesi, M., Orlandi, G., Piazza, F., and Uncini, A., Fast neural networks without multipliers, *IEEE Trans. Neural Networks*, vol. 4, pp. 53-62, Jan. 1993.

McCulloch, W.S. and Pitts, W., A logical calculus of the ideas imminent in nervous activity, *Bulletin of Mathematical Biophysics*, 5, pp.115-133, 1943.

Mead, C.A., *Analog VLSI and Neural Systems*, Reading MA: Addison-Wesley, 1989.

Minai, A.A. and Williams, R.D., Acceleration of back-propagation through learning rate and momentum adaptation, *Proc. International Joint Conference on Neural Networks*, vol. 1, pp. 676-679, Washington, DC, USA, 1990.

Minsky, M. and Papert, S., *Perceptrons: An Introduction to Computational Geometry*, Cambridge, MA: MIT Press, 1969.

Moore, K.L., Artificial neural networks: Weighing the different ways to systematize thinking, *IEEE Potentials*, pp. 23-28, Feb. 1992.

Verlag, 1990.

Nakayama, K., Inomata, S., and Takeuchi, Y., A digital multilayer neural network with limited binary expressions, *IJCNN '90*, San Diego, June 17-21, 1990, vol.2, pp. 587-592.

Narendra, K.S. and Parthasarathy, K., Identification and control of dynamical systems using neural networks, *IEEE Trans. Neural Networks*, vol.1, no. 1, pp. 4-27, Mar. 1990.

Nekovei, R. and Sun, Y., Back-propagation network and its configuration for blood vessel detection in angiograms, *IEEE Trans. Neural Networks*, vol.6, no.1, pp.64-72, Jan. 1995.

Nguyen, D.H. and Widrow, B., Neural networks for self-learning control systems, *IEEE Control Systems Magazine*, pp. 18-23, Apr. 1990.

Nikoonahad, M. and Liu, D.C., Medical ultrasound imaging using neural networks, *Electron. Lett.*, vol.26, no.8, pp.545-546, Apr. 1990.

Oh, H.J. and Salam, F.M.A., Analog CMOS implementation of neural network for adaptive signal processing, *Proc. IEEE Int. Symp. Circuits and Systems*, London, England, vol.6, pp.503-506, 1994.

Oh, H.J. and Salam, F.M.A., Modular analog chip for feedforward networks with on-chip learning, *Proc. 36th Midwest Symp. Circuits and Systems*, Detroit, vol.1, pp.766-769, 1993.

Piazza, F., Uncini, A., and Zenobi, M., Neural networks with digital LUT activation functions, *IJCNN '93*, Japan, vol. 2, pp. 1401-1404.

Phansalkar, V.V. and Sastry, P.S., Analysis of the back-propagation algorithm with momentum, *IEEE Trans. Neural Networks*, vol. 5, pp. 505-506, May 1994.

Przytula, K.W. and Prasanna, V.K., *Parallel Digital Implementations of Neural*

Rauch, H.E. and Winarske, T., Neural networks for routing communication traffic, *IEEE Control Systems Magazine*, pp.26-31, Apr. 1988.

Rosenblatt, F., The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, vol. 65, pp.386-408, 1958.

Rosenblatt, F., *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Washington D.C.: Spartan Books, 1962.

Rumelhart, D.E., Hinton, G.E., and William, R.J., Learning interrnal representations by error propagation, in Rumelhart, D.E. and McClelland, J.L. (Eds.), *Parallel Distributed Processing*, vol.1, ch.8, Cambridge, MA: MIT Press, 1986.

Rumelhart, D.E. and McClelland, J.L., *Parallel Distributed Processing*, vol. 1, *Foundations*. Cambridge, MA: MIT Press, 1986.

Sanchez-Sinencio, E. and Lau, C. (Eds.), *Artificial Neural Networks: Paradigms, Applications, and Hardware Implementations*, New York: IEEE Press, 1992.

Sanger, T.D., Optimal unsupervised motor learning for dimensionality reduction of nonlinear control systems, *IEEE Trans. Neural Networks*, vol.5, no.6, pp.965-973, Nov. 1994.

Schreibman, D.V. and Norris, E.M., Speeding up back propagation by gradient correlation, *Proc. Int. Joint Conf. on Neural Networks*, Washington, D.C., pp. I723-726, 1990.

Sebald, A.V. and Schlenzig, J., Minimax design of neural net controllers for highly uncertain plants, *IEEE Trans. Neural Networks*, vol.5, no.1, pp.73-82, Jan. 1994.

Sejnowski, T.J. and Rosenberg, C.R., Paraller networks that learn to pronounce

Sheu, B.J. and Chor, J., *Neural Information Processing and VLSI*, Kluwer Academic Publishers, Boston, 1995.

Stevenson, M., Winter, R., and Widrow, B., "Sensitivity of feedforward neural networks to weight errors," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 81-92, March 1990.

Szu, H., Fast simulated annealing, in Denker, J.S. (Ed.), *Neural Networks for Computing*, American Institute of Physics, New York, pp. 420-425, 1986.

Tang, C.Z. and Kwan, H.K., Convergence and generalization properties of multilayer feedforward neural networks, *Proc. 1992 IEEE International Symposium on Circuits and Systems*, San Diego, U.S.A., May 1992, pp. 65-68.

Tang, C.Z. and Kwan, H.K., Feedforward neural networks with powers-of-two weights, *Proc. 1992 International Joint Conference on Neural Networks*, Beijing, China, Nov. 1992, Vol.2, pp.93-98.

Tang, C.Z. and Kwan, H.K., Feedforward neural networks without multipliers, *Proc. 1992 International Conference on Signal Processing Applications and Technology*, Boston, U.S.A., Nov. 1992, pp.1194-1201.

Tang, C.Z. and Kwan, H.K., Parameter effects on convergence speed and generalization capability of backpropagation algorithm, *International Journal of Electronics*, vol. 74, no. 1, pp. 35-46, Jan. 1993.

Tang, C.Z. and Kwan, H.K., Multilayer feedforward neural networks with single powers-of-two weights, *IEEE Trans. Signal Processing*, vol. 41, pp. 2724-2727, Aug. 1993.

Tang, D.W. and Hopfield, J.J., Simple "neural" optimization networks: an A/D converter, signal decision circuit, and a linear programming circuit, *IEEE Trans. Circuits and Syst.*, vol. CAS-33, no. 5, pp. 533-541, May 1986.

Sinencio, E. and Lau, C. (Eds.), *Artificial Neural Networks: Paradigms, Applications, and Hardware Implementations*, New York: IEEE Press, 1992.

Thiran, P. and Hasler, M., Self-organization of a one-dimensional Kohonen network with quantized weights and inputs," *Neural Networks*, vol. 7, no. 9, pp. 1427-1439, 1994.

Thiran, P., Peiris, V., Heim, P., and Hochet, B., Quantization effects in digitally behaving circuit implementations of Kohonen networks, *IEEE Trans. Neural Networks*, vol. 5, pp.450-458, 1994.

Vogl, T.P., Mangis, J.K., Rigler, A.K., Zink, W.T., and Alkon, D.L., Accelerating the convergence of the back-propagation method, *Biol. Cybern.*, vol. 59, pp. 257-263, 1988.

Wakerly, J.F., *Digital Design Principles and Practices*, Englewood Cliffs, NJ: PTR Prentice Hall, 1990.

Wang, G.-J., and Chen, C.-C., A fast multilayer neural-network training algorithm based on the layer-by-layer optimizing procedures, *IEEE Trans. Neural Networks*, vol.7, no.3, pp.768-775, May 1996.

Watta, P.B. and Hassoun, M.H., A coupled gradient network approach for static and temporal mixed-integer optimization, *IEEE Trans. Neural Networks*, vol.7, no.3, pp.578-593, May 1996.

Werbos, P., *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. thesis, Harvard University, Cambridge, MA, Aug. 1974.

White, B.A. and Elmasry, M.I., The digi-neocognitron: A digital neocognitron neural network model for VLSI, *IEEE Trans. Neural Networks*, vol. 3, pp. 73-85, Jan. 1992.

Widrow, B. and Hoff, M.E., Adaptive switching circuits, *1960 IRE WESCON*

Widrow, B. and Lehr, M.A., 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation, *Proc. IEEE*, vol. 78, pp. 1415-1442, Sept. 1990.

Xue, P. and Liu, B., Adaptive equalizer using finite-bit power-of-two quantizer, *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 34, pp. 1603-1611, Dec. 1986.

Zaghloul, M.E., Mendor, J.L., and Newcomb, R.W., *Silicon Implementation of Pulse Coded Neural Networks*, Kluwer Academic Publishers, Boston, MA, 1994.

Zak, S.H., Upatising, V., and Hui, S., Solving linear programming problems with neural networks: A comparative study, *IEEE Trans. Neural Networks*, vol.6, no.1, pp.94-104, Jan. 1995.

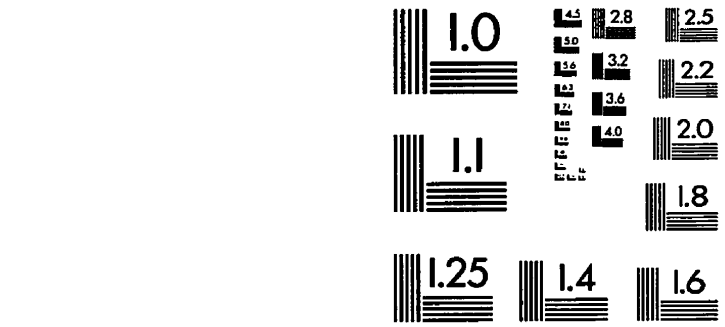
Zhang, M. and Fulcher, J., Face recognition using artificial neural network group-based adaptive tolerance (GAT) trees, *IEEE Trans. Neural Networks*, vol.7, no.3, pp.555-567, May 1996.

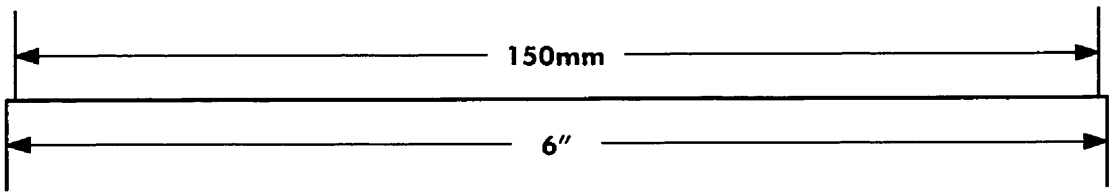
Zhao, Q. and Tadokoro, Y., A simple design of FIR filters with powers-of-two coefficients, *IEEE Tran. Circuits and Systems*, vol.35, no.5, pp. 566-570, May 1988.

Zurada, J.M., Analog implementation of neural networks, *IEEE Circuits and Devices Magazine*, vol. 8, pp. 36-41, Sept. 1992.

VITA AUCTORIS

NAME:	Chuan Zhang Tang
PLACE OF BIRTH:	Beijing, China
YEAR OF BIRTH:	1962
EDUCATION:	Jing Gong High School, Beijing, China 1976 - 1980
	Tsinghua University, Beijing, China 1980 - 1985 B.Eng. in Electrical Engineering
	Peking University, Beijing, China 1985 - 1988 M.Sc. in Electrical Engineering
	University of Windsor, Windsor, Ontario, Canada 1990 - 1996 Ph.D. in Electrical Engineering
WORK EXPERIENCE:	Chinese Academy of Sciences, Beijing, China 1988 - 1990 Research Engineer
	Cableshare Interactive Technology, Inc. London, Ontario, Canada 1995 - present Senior ASIC Design Engineer





APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989