A VLSI implementation of an intelligent sensor.

Robert Bart. Maclean

University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/872

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®
A VLSI Implementation Of An Intelligent Sensor

By

Robert Bart Maclean

A Thesis
Submitted to the Faculty of Graduate Studies and Research
Through the Department of Electrical Engineering
In Partial Fulfillment of the Requirements for
Degree of Master of Applied Science
At the University of Windsor

Windsor, Ontario, Canada

1998
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-52602-X
© 1998, Robert Bart MacLean

All Rights Reserved. No part of this document may be reproduced, stored or otherwise retained in a retrieval system or transmitted in any form, on any medium or by any means without the prior written permission of the author.
Abstract

The purpose of this thesis is to introduce and demonstrate an implementation of the intelligent sensor in a 0.8μ BiCMOS process. The intelligent sensor is a multilayer programmable optically coupled neural network which can be used for pattern recognition for many applications. The intelligent sensor contains an 8x8 array of photosensitive elements as the inputs to this multilayer neural network. The design presented here marks an improvement over an existing design done in a 1.2μ CMOS process. Performance and area efficiency are both improved as a result of the design practices used and the benefits of the BiCMOS process used for this design. The photosensitive element offers a great performance increase over the previous design by using a true device as opposed to a parasitic device. An optical cell test chip containing several optical cell designs was fabricated and tested. The results of these tests were used to choose the optical cell used in the intelligent sensor. As well, a new training method which was tailored for this intelligent sensor design was written. This new training algorithm reduces the error of normal training methods by trying to more closely match the actual network during training. As well, a complete graphical interface was designed for this training tool. Both a 5.0V design and a low power 3.3V design of the intelligent sensor were completed and fabricated. The low power design offers a great power savings over the 5.0V design. The building blocks for the designs were designed and characterized using Cadence 4.3 design tools and were simulated using Hspice. The two chips which were fabricated, were tested using several VLSI test tools and verified the simulation results achieved during the design process.
# Table Of Contents

**ABSTRACT** \hspace{1cm} IV  

**TABLE OF CONTENTS** \hspace{1cm} V  

**TABLE OF FIGURES** \hspace{1cm} VIII  

**TABLE OF EQUATIONS** \hspace{1cm} IX  

**TABLE OF TABLES** \hspace{1cm} X  

1. INTRODUCTION \hspace{1cm} 1-1  
   1.1 Introduction \hspace{1cm} 1-1  
   1.2 Objective Of The Thesis \hspace{1cm} 1-2  
   1.3 Organization Of The Thesis \hspace{1cm} 1-3  

2. THE BASICS OF ARTIFICIAL NEURAL NETWORKS \hspace{1cm} 2-5  
   2.1 Introduction \hspace{1cm} 2-5  
   2.2 The Neuron \hspace{1cm} 2-6  
   2.3 The Interconnection Topology \hspace{1cm} 2-7  
   2.4 The Synapse \hspace{1cm} 2-8  
   2.5 The Learning Algorithm \hspace{1cm} 2-9  
   2.5.1 The Back-Propagation Learning Algorithm \hspace{1cm} 2-10  
   2.6 Different Implementations of Neural Networks \hspace{1cm} 2-13  
   2.7 Conclusion \hspace{1cm} 2-14  

3. A BICMOS VLSI IMPLEMENTATION OF AN INTELLIGENT SENSOR \hspace{1cm} 3-15  
   3.1 Introduction \hspace{1cm} 3-15  
   3.2 The Synapse \hspace{1cm} 3-16  
   3.2.1 The Multiplying DAC \hspace{1cm} 3-16  
   3.2.2 The Neuron \hspace{1cm} 3-19  
   3.2.3 The Memory Cell \hspace{1cm} 3-24  
   3.2.4 The Bias Cell \hspace{1cm} 3-25  
   3.2.5 The Complete Synapse \hspace{1cm} 3-26  

\v
3.3 The Optical Sensor
3.3.1 The Photo-Sensitive Device
3.3.2 The Optical Cell

3.4 The Intelligent Sensor

3.5 Conclusions

4. TRAINING A NEURAL NETWORK

4.1 Introduction

4.2 A Comparison of the Sigmoidal and Actual Activation Functions

4.3 The LUT_BP Training Program
4.3.1 Difference Between The Activation Functions
4.3.2 The Scaling Property Of The Neuron
4.3.3 The Quantization Of Weights
4.3.4 The LUT_BP Training/Recall Simulator

4.4 A Comparison Of The Training Methods
4.4.1 How The Comparison Was Done
4.4.2 The Results Of The Comparison

4.5 Conclusions

5. VERIFICATION OF THE INTELLIGENT SENSOR

5.1 Introduction

5.2 The Optical Cell Test Chip

5.3 The Intelligent Sensor Chips
5.3.1 Characterizing The Neuron
5.3.2 Characterizing The Synapse

5.4 Conclusions

6. CONCLUSIONS

6.1 Conclusions

6.2 Future Directions

REFERENCES

APPENDIX A: RELVSTHY.M

APPENDIX B: LUT_BP.M
# Table Of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIGURE 2-1</td>
<td>THE MCCULLOCH-PITTS NEURON</td>
<td>2-6</td>
</tr>
<tr>
<td>FIGURE 2-2</td>
<td>A FEED-FORWARD NEURAL NETWORK</td>
<td>2-8</td>
</tr>
<tr>
<td>FIGURE 3-1</td>
<td>THE MULTIPLYING DAC SYNAPSE</td>
<td>3-17</td>
</tr>
<tr>
<td>FIGURE 3-2</td>
<td>MDAC OUTPUT CURRENT VS. SUCCESSIVE WEIGHTS</td>
<td>3-18</td>
</tr>
<tr>
<td>FIGURE 3-3</td>
<td>LAYOUT OF THE MDAC</td>
<td>3-19</td>
</tr>
<tr>
<td>FIGURE 3-4</td>
<td>THE NEURON</td>
<td>3-20</td>
</tr>
<tr>
<td>FIGURE 3-5</td>
<td>THE NONLINEAR CHARACTERISTIC OF THE NEURON</td>
<td>3-20</td>
</tr>
<tr>
<td>FIGURE 3-6</td>
<td>SATURATION CURRENT OF THE 4 MOS DEVICES</td>
<td>3-21</td>
</tr>
<tr>
<td>FIGURE 3-7</td>
<td>THE LAYOUT OF THE NEURON</td>
<td>3-22</td>
</tr>
<tr>
<td>FIGURE 3-8</td>
<td>THE AUTO-SCALING PROPERTY OF THE NEURON</td>
<td>3-23</td>
</tr>
<tr>
<td>FIGURE 3-9</td>
<td>THE NONLINEAR CHARACTERISTIC OF THE 3.3V NEURON</td>
<td>3-24</td>
</tr>
<tr>
<td>FIGURE 3-10</td>
<td>THE MEMORY CELL</td>
<td>3-24</td>
</tr>
<tr>
<td>FIGURE 3-11</td>
<td>LAYOUT OF THE MEMORY CELL</td>
<td>3-25</td>
</tr>
<tr>
<td>FIGURE 3-12</td>
<td>THE BIAS CIRCUIT</td>
<td>3-26</td>
</tr>
<tr>
<td>FIGURE 3-13</td>
<td>THE COMPLETE SYNAPSE</td>
<td>3-26</td>
</tr>
<tr>
<td>FIGURE 3-14</td>
<td>A DARLINGTON CONFIGURED NPN TRANSISTOR</td>
<td>3-28</td>
</tr>
<tr>
<td>FIGURE 3-15</td>
<td>THE OPTICAL CELL</td>
<td>3-28</td>
</tr>
<tr>
<td>FIGURE 3-16</td>
<td>LAYOUT OF THE OPTICAL CELL</td>
<td>3-29</td>
</tr>
<tr>
<td>FIGURE 3-17</td>
<td>LAYOUT OF LARGE BUILDING BLOCK OF INTELLIGENT SENSOR</td>
<td>3-30</td>
</tr>
<tr>
<td>FIGURE 3-18</td>
<td>LAYOUT OF THE INTELLIGENT SENSOR</td>
<td>3-31</td>
</tr>
<tr>
<td>FIGURE 4-1</td>
<td>A COMPARISON OF THE TWO ACTIVATION FUNCTIONS</td>
<td>4-35</td>
</tr>
<tr>
<td>FIGURE 4-2</td>
<td>THE DIFFERENCE BETWEEN THE TWO ACTIVATION FUNCTIONS</td>
<td>4-35</td>
</tr>
<tr>
<td>FIGURE 4-3</td>
<td>A COMPARISON OF THE TWO QUANTIZED ACTIVATION FUNCTIONS</td>
<td>4-36</td>
</tr>
<tr>
<td>FIGURE 4-4</td>
<td>THE DIFFERENCE BETWEEN THE TWO QUANTIZED ACTIVATION FUNCTIONS</td>
<td>4-36</td>
</tr>
<tr>
<td>FIGURE 4-5</td>
<td>SCREEN SHOT OF LUT_BP</td>
<td>4-38</td>
</tr>
<tr>
<td>FIGURE 4-6</td>
<td>TRAINING PATTERNS</td>
<td>4-39</td>
</tr>
<tr>
<td>FIGURE 4-7</td>
<td>TRAINING PROGRESS</td>
<td>4-40</td>
</tr>
<tr>
<td>FIGURE 5-1</td>
<td>AVERAGE ACTIVATION FUNCTION ACROSS THREE CORNERS</td>
<td>5-48</td>
</tr>
<tr>
<td>FIGURE 5-2</td>
<td>ACTUAL VS. SIMULATION RESULTS</td>
<td>5-48</td>
</tr>
<tr>
<td>FIGURE 5-3</td>
<td>MAXIMUM VARIATION FROM AVERAGE CURVE</td>
<td>5-49</td>
</tr>
<tr>
<td>FIGURE 5-4</td>
<td>AVERAGE ACTIVATION FUNCTION OF 3.3V DESIGN</td>
<td>5-50</td>
</tr>
<tr>
<td>FIGURE 5-5</td>
<td>THE SYNAPSE</td>
<td>5-51</td>
</tr>
<tr>
<td>Equation</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>EQUATION 1</td>
<td>2-6</td>
<td></td>
</tr>
<tr>
<td>EQUATION 2</td>
<td>2-7</td>
<td></td>
</tr>
<tr>
<td>EQUATION 3</td>
<td>2-7</td>
<td></td>
</tr>
<tr>
<td>EQUATION 4</td>
<td>2-10</td>
<td></td>
</tr>
<tr>
<td>EQUATION 5</td>
<td>2-11</td>
<td></td>
</tr>
<tr>
<td>EQUATION 6</td>
<td>2-11</td>
<td></td>
</tr>
<tr>
<td>EQUATION 7</td>
<td>2-11</td>
<td></td>
</tr>
<tr>
<td>EQUATION 8</td>
<td>2-12</td>
<td></td>
</tr>
<tr>
<td>EQUATION 9</td>
<td>2-12</td>
<td></td>
</tr>
<tr>
<td>EQUATION 10</td>
<td>2-12</td>
<td></td>
</tr>
<tr>
<td>EQUATION 11</td>
<td>2-12</td>
<td></td>
</tr>
<tr>
<td>EQUATION 12</td>
<td>2-13</td>
<td></td>
</tr>
<tr>
<td>EQUATION 13</td>
<td>2-13</td>
<td></td>
</tr>
<tr>
<td>EQUATION 14</td>
<td>3-21</td>
<td></td>
</tr>
<tr>
<td>EQUATION 15</td>
<td>3-27</td>
<td></td>
</tr>
<tr>
<td>EQUATION 16</td>
<td>3-27</td>
<td></td>
</tr>
<tr>
<td>EQUATION 17</td>
<td>3-28</td>
<td></td>
</tr>
<tr>
<td>EQUATION 18</td>
<td>4-34</td>
<td></td>
</tr>
</tbody>
</table>
Table Of Tables

TABLE 1: COMPARISON OF TRAINING RESULTS
1. Introduction

1.1 Introduction

Multi-layer artificial neural networks offer many advantages in the field of pattern recognition. They offer a simple model of the computational elements like those that a human brain uses. A human brain is much better at recognizing complex patterns, and an artificial neural network can offer these same advantages. A VLSI implementation of an artificial neural network described in this thesis is designed for the use in process control applications requiring image capture or non-contact measurement. This can be used to determine the position or surface geometry of an object using either reflected laser beam-steering methods or structured illumination. Existing designs of an optically-coupled artificial neural network have been successfully implemented in the past [1][2].

A VLSI implementation of a programmable multi-layer neural network with optical sensor fusion, or an intelligent sensor, offers many challenges. The massive interconnection found in the network, the memory required for programmability of the network, the realization of a photo-sensitive device for the optical sensor, and the training method, can all compromise the design and performance of the network. The VLSI implementation discussed in this thesis offers increased photo-sensor performance together with robustness, modularity, high interconnection density, an area efficient layout, and a better training method.
1.2 Objective Of The Thesis

The objective of this thesis is to investigate and demonstrate an implementation of an intelligent sensor. The work presented here is an improvement to an existing design done in a 1.2\mu, double metal CMOS process and translated to a 0.8\mu, 3-metal, twin-well BiCMOS process. This previous design was improved upon through the benefits of the BiCMOS process and through the design practices followed. Both performance, and area efficiency was improved.

As well, a new training method is introduced for this new design. Traditional training methods resulted in an unacceptable amount of error and this new training method reduces the error of normal training methods by almost half.

In all, two designs of the intelligent sensor were completed, a high speed 5.0V design, and a low power 3.3V design. Simulation results and actual testing results of these two chips and a third optical sensor chip, will be discussed.

Throughout this project, expertise in many VLSI design tools has been demonstrated. The tools used for the design, layout, and extraction of the VLSI implementation of the intelligent sensor were the Cadence Analog Artist 4.3 design tools. Comprehensive simulations of all the building blocks and the complete designs were performed using Hspice. The new training method was written in the MatLab environment and uses a fully graphical interface. The testing of the chips used a HP workstation running the VecTest programming environment, CMC's TH1000 test head, a HP 8180A data generator, and a Tektronix 11402A oscilloscope.
1.3 Organization Of The Thesis

The thesis consists of six chapters. This first chapter is an introduction to the thesis explaining the thesis' motivation, objective, and organization.

The second chapter is an introduction to the theory of artificial neural networks. It explains the building blocks found in any neural network, what a training algorithm is used for, and also explains the back-propagation training algorithm. The chapter also discusses the different types of implementations of artificial neural networks that exist.

The third chapter introduces the VLSI implementation of the intelligent sensor. This begins by describing the design of the building blocks, followed by the VLSI layout of the blocks and simulation results, for both the 5.0V and 3.3V designs. It also describes the advantages of the building blocks over previous designs. Also, a new design of the optical cell is presented which offers higher performance over previous designs. The chapter ends with a description of how the building blocks were put together to form the full-size intelligent sensor.

Chapter Four introduces a new method of training for the intelligent sensor and the graphical front-end to the program. This new method of training uses the a modified version of the back-propagation algorithm presented in Chapter Two which minimizes the inaccuracies normally associated with the training of the intelligent sensor. This new algorithm tries to more closely match the actual network rather than a theoretical model of the network.

Chapter Five discusses the test results of the three chips that were fabricated. The chips included both a 5.0V and a 3.3V intelligent sensor and as well, an optical cell test chip.
The building blocks of the neural network are characterized and the performance of the optical cells was discussed.

Chapter Six is the concluding chapter. This chapter presents a summary of the work done and as well, presents possible directions for future research and work in this area.
Chapter 2.

The Basics Of Artificial Neural Networks

2.1 Introduction

The artificial neural network is a man-made system motivated by the neural structure found in the biological nervous system. It is an electronic network which not only models the structure of the biological nervous system, but many have the ability to learn or update the knowledge they have acquired from past experiences. From a neurophysiological standpoint, these models are extremely simplified [3], but have been used for several applications including pattern classification, prediction and financial analysis, computer science, control and optimization, and telecommunications [4].

The motivation for research on artificial neural networks is due to the advantages of the neural network when compared to a digital computer [5]. Higher rates of computation can be achieved in an artificial neural network due to its massively parallel and asynchronous nature, whereas a digital computer works sequentially. A higher degree of robustness or fault tolerance is achieved because of the high number of processing nodes. A problem in a few of these nodes may not affect the network’s performance significantly. Along the same principle, the neural network can handle fuzzy or noisy inputs. Also, due to the networks nonlinear characteristics, nonlinear mapping and approximation can be easily achieved.
The rest of this chapter is a discussion of the four building blocks which can describe any artificial neural network: the neuron, the interconnection topology, the synapse and it’s learning algorithm. The final section of this chapter will discuss several different ways of implementing neural networks.

2.2 The Neuron

The neuron is the basic processing element of an artificial neural network. The classic neuron model was developed by McCulloch and Pitts as a binary threshold unit [6]. The model computes a weighted sum of its inputs and output either a one or a zero depending whether the sum was greater then or less than the neuron’s threshold.

![Diagram of McCulloch-Pitts Neuron]

*Figure 2-1: The McCulloch-Pitts Neuron*

Figure 2-1 shows the McCulloch-Pitts neuron model. Equation 1 is the mathematical expression for this model.

\[
n_i(t+1) = f \left( \sum_j w_{ij} n_j(t) - \Theta_i \right)
\]

*Equation 1*

The output \( n_i \) represents the output of the \( i^{th} \) neuron and is either one or zero. \( j \) is the number of inputs and \( \Theta \) is the threshold value of the neuron. The weights are given by \( w_{ij} \) and can be positive or negative or zero which effectively means there is no connection between the \( i \) and \( j \) neurons. The \( f \) function represents the unit step function given in Equation 2 and is referred to as the transfer function or the activation function.
\[ f(x) = \begin{cases} 1; & \text{if } x \geq 0 \\ 0; & \text{otherwise} \end{cases} \]

*Equation 2*

There are several differences of this model to a neuron found in nature. In nature, real neurons often are not even approximate threshold devices. In fact, the transfer function varies from neuron to neuron. Also, real neurons do not act synchronously, but asynchronously. To make Equation 1 more realistic, it can be modified to Equation 3.

\[ n_i = g_i \left( \sum_j w_{ij} n_j - \Theta_i \right) \]

*Equation 3*

The output \( n_i \) is now continuously valued and the function \( g(x) \) is a more general nonlinear activation function. This function can be any one of an infinite number of nonlinear activation functions.

### 2.3 The Interconnection Topology

The interconnection topology describes the way the neurons are interconnected. It can be feed-forward, recursive, or a mixture of both feed-forward and recursive. A feed-forward network propagates signals in one direction through the network where a recursive network feeds outputs of the network back as inputs to the network. A network can also be distinguished by the number of inputs, output nodes, layers, and the number of neurons in each layer. The input layer is not counted when referring to the number of layers in a network. Also, neurons occurring anywhere but the output layer are hidden layer neurons.
Figure 2-2 shows a typical feed-forward neural network. It has four inputs, two outputs, two layers, and three neurons on the only hidden layer. Also notice the connections between the layers. Every output of a layer is connected to every input of the next layer. This is referred to as full connectivity and is a feature of most feed-forward neural networks. Each connection is also referred to as the synapse.

2.4 The Synapse

The synapse is a connection between an output of one layer to the input of another layer. Each synapse has a weight associated with it that can be positive, negative, or zero which effectively means there is no connection between the layers. These weights are the multiplication factor $w_{ij}$ from Equation 1 and Equation 3. The weights for all of the synapses between a layer in a neural network form a weight matrix. This weight matrix is usually a result of a learning process. It is possible for a neural network to perform the same function with different weight matrices. In other words, there can be more than one solution to every problem.
2.5 The Learning Algorithm

In order to realize a desired result with an artificial neural network, a training or learning process must be used to find the correct weight matrix. There are two main classes of training algorithms, supervised and unsupervised. With unsupervised learning, the network does not receive any feedback from the environment, and no information about the correct results are given. The training relies on redundancies in order for the network to self-organize. Supervised learning is a process where feedback about the errors is given and the weights are adjusted to reduce the error. One such supervised training algorithm used for multi-layer neural networks is the back-propagation training algorithm [7].

The development of the back-propagation learning algorithm was an important discovery because of the algorithm’s ability to train multi-layer networks with multiple output nodes and a smooth nonlinearity for an activation function [7]. These type of neural networks offer solutions to problems much more complex than with single-layer neural networks. Also, the capabilities of these multi-layer networks stem from the nonlinear activation function. If a linear activation function was used, any single layer neural network could be trained to form the exact function. A two-layer neural network with a smooth nonlinear activation function can solve a smooth convex region in a space spanned by the inputs [5]. This means that a two-layer neural network can recognize patterns that can be classified with a single closed region but not multiple closed regions. If a hard-limiting activation function was used, the patterns would be bounded by straight line segments. The complexity of the region can be increased by increasing the number of the neurons in the network. For even more complex problems, it has been shown that
a three-layer neural network with a nonlinear activation function and a sufficient number of neurons can solve any complex problem with multiple closed decision regions [8].

The following section explains one variation of the back-propagation training algorithm.

2.5.1 The Back-Propagation Learning Algorithm
Back-Propagation (BP) is one of the most popular supervised learning algorithms used for multilayer feed-forward networks. This algorithm works on the minimum disturbance principle: it adapts to reduce the output error for the current training pattern with a minimal disturbance to patterns already learned. Its objective is to reduce the mean-square error averaged over all training patterns by altering the weights of a network by gradient descent. The gradient descent rule used in the BP algorithm is referred to as the method of steepest-descent. The minimum mean-square error is found at a local minimum of the mean-square error function.

Back-Propagation begins by initializing the weights to small, random real numbers. Each input pattern is propagated through the network and the mean-square errors from each neuron from every layer are used to minimize the error by method of steepest-descent. This method can be described by the following equation:

$$W_{k+1} = W_k + \mu(-\nabla_k)$$

*Equation 4*

The initial weight matrix $W_0$ is used and the mean-square error function is found. The gradient of this function $\nabla_0$ is found and the weight matrix is altered in the negative direction of this gradient. The parameter $\mu$ controls stability and rate of convergence.
This cycle is repeated over and over until the mean-square error is minimized and the weight matrix reaches an optimum value.

Since the BP training algorithm involves a derivative, a differentiable activation function is required for the neuron. A sigmoidal activation is used with the following equation:

\[ g(x) = \frac{1}{1 + e^{-2\beta x}} \]

*Equation 5*

where \( \beta \) is a steepness parameter (which is usually set to 0.5 or 1) and \( x \) is the input to the neuron. The derivative of this function is given as:

\[ g'(x) = 2 \cdot \beta \cdot g(x) \cdot (1 - g(x)) \]

*Equation 6*

For a neural network with \( M \) layers and \( m=1, 2, ..., M \) where \( m=0 \) is the input layer, let \( w_{ij} \) is the weight to the \( i^{th} \) neuron with \( j \) inputs, and \( n_i \) be the output from the \( i^{th} \) neuron.

The algorithm continues as follows:

- The weight matrix is initialized with small real numbers.

- Choose an input pattern and propagate the pattern through the network with the equation:

\[ n_i^m = g \left( \sum_j w_{ij}^m n_j^{m-1} \right) \]

*Equation 7*

for each \( i \) and \( m \) until the output layer \( M \) is reached.

- Compute the error gradient \( \delta \) at the output layer \( M \) with the equation
\[ \delta_i^M = g' \left( \sum_j w_{ij}^M n_j^{M-1} \right) \ast (t_i - n_i^M) \]

Equation 8

where \( t \) is the desired output.

- The errors for the preceding layers are then calculated by propagating the errors backward with the following equation:

\[ \delta_i^{m-1} = g' \left( \sum_j w_{ij}^{m-1} n_j^{m-2} \right) \ast \sum_j w_{ji}^m \delta_j^m \]

Equation 9

for \( m=M, M-1, ..., 2 \) until a \( \delta \) is computed for every neuron.

- The weights are then updated with

\[ w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij} \]

Equation 10

where \( \Delta w \) is given by

\[ \Delta w_{ij}^m = \eta \cdot \delta_i^m \cdot n_j^{m-1} \]

Equation 11

and \( \eta \) is a gain term called the learning rate.

- The next training pattern is then selected and propagated through the network and the whole cycle begins again.

- When the gain term \( \eta \) is too small, the gradient descent can be very slow. If it is too large, it can oscillate widely. To minimize this problem, in this project a momentum term has been added to \( \Delta w \) which forces a gradient descent change in the average downhill direction. The equation then becomes:
\[ \Delta w^w_q(t + 1) = \eta \cdot \delta^m_j \cdot n^w_j^{-1} + \gamma \cdot \Delta w^w_q(t) \]

Equation 12

The \( \gamma \) is the momentum term and usually has a value of around 0.9. The idea is to give each weight a kick in the direction of the average downhill force to avoid wild oscillations found with the little kicks when the momentum term is zero. The learning rate, \( \eta \), can also be changed from iteration to iteration depending on the change in error. This is shown in Equation 13.

\[
\eta(t + 1) = \begin{cases} 
  i \cdot \eta(t) & \text{if } \Delta E < 0 \\
  d \cdot \eta(t) & \text{if } \Delta E > 0 
\end{cases}
\]

Equation 13

where \( i \) is the learning rate increase term and \( d \) is the learning rate decrease term.

2.6 Different Implementations of Neural Networks

Artificial neural networks can be implemented many ways. They can be done in software or hardware. Software implementations are very flexible and can simulate huge neural networks. However, due to the nature of neural networks, software implementations are very slow. The computers which run these software implementations operate sequentially, and a neural network is a massively parallel device. This is the reason for a hardware implementation of an artificial neural network. While not as flexible as a software implementation, a hardware implementation is much faster which is required for many real-life applications.

Hardware implementations of neural networks come in many flavors. There are digital, analog, and hybrids which are a mix of both analog and digital components. Analog implementations are very speedy and compact and use lower power when compared to a
digital implementation. However, they are not very accurate due to the use of simple elements which are approximations to the theoretical elements. Digital implementations are much more accurate, but in order to achieve this accuracy, the elements become very complex. This causes the digital neural networks to use more power and area and operate at a slower speed than an analog implementation.

2.7 Conclusion

The basics of neural networks and their building blocks have been discussed. The building blocks consist of the neuron, the interconnection topology, the synapse, and the learning algorithm. One type of learning algorithm, the back-propagation learning algorithm was introduced and will be referred to later in this thesis. Also, the different types of implementations of neural networks were discussed and a hybrid hardware implementation will be introduced in the following chapter.
Chapter 3.

A BiCMOS VLSI Implementation Of An Intelligent Sensor

3.1 Introduction

A VLSI implementation of a programmable intelligent optical sensor based on a neural network offers many challenges. The massive interconnection requirements found in any implementation of a neural network, and especially a multi-layer one, is always a major design constraint. The layout of the memory elements required for the programmability of the network always represents a major allocation of silicon area. Also, the realization of a photo-sensitive device in a standard CMOS technology relies on enhancing a parasitic bipolar transistor where the substrate is used as a transistor terminal. The implementation described in this chapter offers an improved solution to these common problems and marks an improvement to an existing design done in a 1.2μ CMOS process [2]. These improvements are a result of better design practices, and a better technology choice.

The intelligent sensor is made up of several fundamental building blocks. Basically, it is made up of a two-layer neural network with programmable digital weights, and an array of photo-sensitive devices as the inputs to the network. The neural network uses a combined Multiplying DAC synapse and a distributed neuron to generate a unified synapse-neuron [2] building block which is also sometimes called a distributed neuron-synapse [9]. Each MDAC building block incorporates a realization of a nonlinear resistor. These distributed nonlinear resistors are combined in parallel to realize the
overall nonlinear activation function of the resulting neuron. The activation function has a well conditioned sigmoidal form. Another important building block in the sensor architecture is an array comprised of photo-sensitive elements. The final building block is the memory structure that stores the values associated with the programmable synaptic weights. All of these building blocks have been implemented in a 0.8μ 3-metal, twin-well BiCMOS process using Cadence 4.3 design tools. Comprehensive simulations have been performed using Hspice.

This chapter will discuss each of the building blocks of the intelligent sensor for both a 5.0V and a 3.3V design, and as well, will discuss the layout-related advantages of using an integrated synaptic weight-distributed neuron modules and the advantages of using BiCMOS as the technology chosen for the implementation. The design will also be compared to a previous realization based on the use of a 1.2μ CMOS technology [2] pointing out the improvements to the to the previous design.

3.2 The Synapse
The synapse in this design is actually made up of several smaller blocks: the Multiplying DAC (MDAC), a part of the distributed neuron, memory elements, and a bias voltage generator. Together, these blocks form an integrated synapse-neuron module.

3.2.1 The Multiplying DAC
The MDAC is basically designed with two common building blocks, the differential amplifier and the current mirror. Figure 3-1 shows the circuit diagram for the 5.0V design.
The MDAC receives six inputs, a 5-bit signed number, D0 to D4, and $V_{in}$. The circuit has one output, $I_{out}$. The circuit consists of four binary-weighted (8, 4, 2, 1) cascode current mirrors [10] and a sign circuit at the top which is a differential amplifier. This binary weighting scheme and the five digital inputs result in 31 possible input values, -15 to 15.

The binary-weighted current mirrors allow the inputs D3 to D0 to select a current between 0 and 15 times the original current. Also, the current mirrors were designed using a layout technique known as “ΔW correction” [11] which increases the accuracy of the amplification weight of the current mirror. For example, instead of using one transistor with a width eight times that of the width of the transistor in the first stage of the current mirror, eight transistors of equal length are connected in parallel. Statistically, this is more accurate. Using the cascode configuration of the current mirror, the output resistance was increased which minimizes the effect of channel-length modulation.

The top of the circuit consists of a differential amplifier which acts as the sign circuit. Normally the voltage difference of the differential pair is what is amplified. In the case, a
p-channel and a n-channel transistor make up the differential pair and the difference in voltage appears to be either \(+V_{dd}\) or \(-V_{dd}\) depending on D4. This causes the output to be either \(+I_{out}\) when D4 is 0, or \(-I_{out}\) when D4 is 1.

Overall, by changing D4 to D0, the output current \(I_{out}\) can be increased from -15 to 15 which allows 31 different weight values for each synapse. Figure 3-2 shows simulation results of the output current of the MDAC as the binary weight is stepped from -15 to 15 which corresponds to an output current of \(-77\mu A\) to \(77\mu A\).

![Figure 3-2: MDAC Output Current vs. Successive Weights](image)

The BiCMOS layout of this cell is shown in Figure 3-3. This block was laid out using minimum spacing and device sizing in order to minimize the area used. Through several Hspice simulations, the correct transistor sizing was determined in order to achieve the goal of a current source with 31 different output values, all an equal step size apart.
When comparing this design to a previous design done in a 1.2μ CMOS process, this block is 59% smaller and 33% more power efficient.

A 3.3V design of this block was also done. The only difference was that cascode current mirrors could no longer be used - a standard configuration current mirror was used. This is required due to the smaller supply voltage. The voltage drop across each transistor in the cascode configuration was affecting the performance of the current mirror - the transistors were not saturated which is required for correct operation of the current mirror. This 3.3V design of the MDAC has an output current of -20.8\(\mu\)A to 20.8\(\mu\)A when the weights are stepped from -15 to 15. Also, an additional 60 % power savings over the 5.0V design is realized.

3.2.2 The Neuron
The neuron used in this design is based on a nonlinear load which has an I-to-V characteristic that approximates a sigmoidal activation function. It consists of four MOS devices and the circuit is shown in Figure 3-4.
Figure 3.4: The Neuron

By sweeping the input current $I_{in}$ from -77μA to 77μA, exactly which is the output values of the MDAC, the desired output $V_{out}$ is a sigmoidal shape between 0 and 5V. This desired output is achieved with properly chosen MOS device geometries and bias voltages. These device geometries were arrived at through many Hspice simulations. loops were used in Hspice to quicken this process of finding perfect device geometries. This resulted in a neuron which more closely matched the MDAC than in the previous design. The simulation results appear in Figure 3.5.

Figure 3.5: The Nonlinear Characteristic Of The Neuron
This approximate sigmoidal shape is achieved by the combination of the four quadratic I-to-V characteristics of the four MOS devices. The quadratic characteristic is the I-to-V characteristic curve when a MOS device is in saturation which occurs when $V_{GS} > V_T$ and $V_{DS} = V_{GS} - V_T$. This quadratic equation for a NMOS device is given in Equation 14.

$$I_{ds} = K \cdot (V_{gs} - V_T)^2$$

Equation 14

Figure 3-6 shows the $I_{ds}$ current for the four MOS devices as the input current is swept from $-77\mu A$ to $77\mu A$.

![Figure 3-6: Saturation Current Of The 4 MOS Devices](image)

As shown, M2 is saturated when the output voltage is between 0 and 1V, M4 is saturated when the output voltage is between 0 and 2.5V, M3 is saturated when the output voltage is between 2.5 and 5V, and M1 is saturated when the output voltage is between 4 and 5V.

The layout of the neuron is shown in Figure 3-7. This block was designed so that it could be easily connected to the MDAC element to create a larger building block.
A major advantage to this type of nonlinear resistive neuron is its distributed property. Each MDAC is mated to a single neuron to form a synapse. If there are several inputs to a neuron in a neural network, and since each input is mated to a neuron, each of these neurons contribute to only part of the complete neuron's activation function. Normally, for each different number of input synapses, a new neuron has to be designed in order to scale the sigmoidal shape as the input current increases with each input synapse. In this case, each input synapse has neuron element, and connecting the synapses together effectively lowers the combined resistance of the neuron elements. It is basically connecting nonlinear resistances in parallel. Figure 3-8 shows this property by comparing the activation function when just one distributed neuron is used, with the activation function when five distributed neurons are connected together.
This property simplifies the design of a network with different numbers of neurons and synapses as it makes the design much more modular. The synapses just have to be connected together and the neuron is taken care of. Also, this increases the robustness of the design. For example, if there were five of the nonlinearities connected in parallel, a defect would only affect one fifth of a neuron instead of an entire neuron.

A 3.3V design of the neuron was also done. This required changing the neuron's device geometries and bias voltages to match the MDAC 3.3V output currents and still result in a good approximation to the sigmoidal shape. This involved again many simulations and Hspice loops to determine the appropriate device geometries. Figure 3-9 shows the simulation results of the 3.3V neuron.
3.2.3 The Memory Cell

The storage elements of this network are realized by a dynamic shift register (DSR) circuit. Each MDAC requires five memory cells (D0 to D4) to store the digital synaptic weights for the network. A vector of the synaptic weight values is serially loaded into the DSR and then shifted to the appropriate MDAC. An area-efficient memory cell that employs two nonoverlapping clocks to control the shifting has been designed. The memory cell circuit is shown in Figure 3-10.

Figure 3-10: The Memory Cell
The memory cell is made up of three inverters with two of them connected back-to-back. The double-phase clocks, Phi-1 and Phi-2, drive the internal switches for storing data in the cell. The BiCMOS layout of this cell is shown in Figure 3-11.

![Figure 3-11: Layout Of The Memory Cell](image)

Area efficiency drove the VLSI layout of this block. Minimum transistor sizing and spacing was used. As well, the block was designed so that several of them could be easily connected together. The area used by this cell is 61% less than the area of the same cell designed in a 1.2μ CMOS process.

### 3.2.4 The Bias Cell
The bias cell is just used to create the two bias voltages required by the neuron. It is just a simple voltage divider circuit and is shown in Figure 3-12.
Both a 5.0V and a 3.3V design was completed. This involved changing the MOS device geometries to achieve the correct bias voltages for both the 5.0V and the 3.3V neurons.

3.2.5 The Complete Synapse
A complete synapse is made up of a MDAC, a neuron element, five memory cells, and a bias circuit. This modular block can be easily connected with other synapses to form a complete node. Figure 3-13 shows the BiCMOS layout of the complete synapse.

3.3 The Optical Sensor
The optical sensor is a device that forms the inputs to the neural network. The optical sensor consists of a photo-sensitive device and some circuitry to convert the output of the photo-sensitive device to a usable voltage level.

3.3.1 The Photo-Sensitive Device
A photo-sensitive device is a device which converts light into an electric signal. Several choices are available including a photo-diode, a photo-MOS, and a photo-BJT. The deciding factor for the photo-sensitive device was determined by the gain displayed by
the element. Since the diode and the MOS have little or no optical gain, the BJT was chosen. The high optical gain of the BJT results in a very large S/N ratio and a high sensitivity [12]. In this case, a NPN transistor is used as the photo-sensitive device.

This photo-transistor basically acts as a photo-current generator. If light strikes the base of the transistor, and the photon has an energy level greater than the band gap of silicon, an electron-hole pair is created. The electrons created are collected by the collector and results in a positive charge at the base. This forward bias of the emitter-base junction causes the emitter to inject electrons into the base and holes injected from the base into the emitter. While the transistor is in this active mode of operation, the emitter current, $i_E$, is the desired photo-current.

This small current generated at the base, $i_B$, by the light striking it, multiplied by the common-emitter current gain, $\beta$, is equal to the collector current, $i_C$. This is stated in Equation 15.

$$i_C = \beta \cdot i_B$$

*Equation 15*

The base current, $i_B$, is proportional to the intensity of light. The photo-current, $i_E$, is given in Equation 16.

$$i_E = i_C + i_B$$

*Equation 16*

The gain of this device, $\beta$, is an important value. The base current is a very small value and must be amplified to a usable level. In the BiCMOS technology, a true NPN transistor is used which has a $\beta$ of 100. A configuration to increase this gain known as
the Darlington configuration was also considered. Figure 3-14 shows a Darlington configured NPN transistor.

![Darlington Configured NPN Transistor](image)

*Figure 3-14: A Darlington Configured NPN Transistor*

The gain of this Darlington configured transistor is given by Equation 17.

\[ \beta = \beta_1 \cdot \beta_2 \]

*Equation 17*

\( \beta_1 \) is the gain of the first NPN transistor and \( \beta_2 \) is the gain of the second NPN transistor. Basically, the gain of the device has been squared.

### 3.3.2 The Optical Cell

Figure 3-15 shows the circuit of the entire optical cell.

![Optical Cell Circuit](image)

*Figure 3-15: The Optical Cell*

It includes the NPN transistor which is the photo-sensitive device. Since the photocurrent generated by this device is proportional to a wide range of light intensities, a load
circuit with a logarithmic response is desirable to convert the wide range of photo-current to a small range of voltage. The two diode connected NMOS transistors form this logarithmic load [3]. The dc bias voltage is adjusted to compensate for different ambient light levels. There is a buffer made up of two inverters at the output of this circuit, and therefore, the output of the optical sensor is a voltage at a digital level. Figure 3-16 shows the optical cell's layout.

![Diagram of Optical Cell](image)

Figure 3-16: Layout Of The Optical Cell

Notice the geometry of the large base and the placement of the emitter right in the center of the base. The base occupies an area 16.2μ x 13.2μ. This cell was designed so that the support circuitry of photo-sensitive device was as small as possible and that the optical cell could easily be connected with other optical cells to form an array of sensors.

In a previous design in a 1.2μ CMOS process, a true bipolar transistor could not be used for the photo-sensitive device. A parasitic PNP transistor was used which used the substrate of the process as the collector of the transistor. Even by using a Field-Effect Modified (FEM) photo-transistor, which is a technique to increase the gain of the device, the β of the transistor was only 35. This small gain requires a very large base to generate
a much larger base current. Due to the differences of gain, the size of this optical cell in BiCMOS is only 22% the size of the optical cell designed in a 1.2μ CMOS process.

3.4 The Intelligent Sensor

Putting all of the building blocks together has resulted in a full size implementation of an intelligent sensor. The sensor consists of a 2-layer neural network with 64 inputs, 8 hidden layer neurons, and 4 output layer neurons. The 64 inputs consist of an 8x8 array of optical cells. All together, the sensor consists of 2780 memory cells, and 556 synapse modules.

The intelligent sensor was assembled by piecing together 64 large blocks consisting of 8 synapse modules and an optical cell. This block is shown in Figure 3-17.

![Figure 3-17: Layout Of Large Building Block Of Intelligent Sensor](image)

This large building block was designed so that many of them could be connected in an array to form the complete intelligent sensor. This modular design of this block helps reduce the common problem of the massive interconnection found in many neural
network designs. Putting 64 of these large building blocks together results in a complete intelligent sensor shown in Figure 3-18.

![Figure 3-18: Layout Of The Intelligent Sensor](image)

Both a 5.0V design and a 3.3V design were completed. Simulations of the 3.3V design have proven to be 77% more power efficient than the 5.0V design. For a small neural network simulation, the 5.0V design drew on average 13mW and the 3.3V design drew 3mW. Also, the 5.0V design was around four times as fast as the 3.3V design. Comparing these designs to an existing design done in a 1.2μ CMOS process, the design proved to be 68% more area efficient. This is due to the smaller minimum feature size and the additional metal layer found in the BiCMOS process that allowed an increased interconnection density. Also, using a true bipolar device in the optical cell allowed a large area savings.
3.5 Conclusions

A BiCMOS design of a programmable intelligent optical sensor has been presented. This results in an improvement and a translation of a previous design. The VLSI layout contains distributed synapse-neuron elements, a DSR realization of the programmable memory cells, and an array of optical cells. All of the building blocks of a hybrid, multi-layer neural network implementation of the intelligent sensor have been built and simulated. The design practices used resulted in a more closely matched neuron and MDAC than in the previous design. As well, the BiCMOS technology allowed for further improvements in performance and area through the use of a true NPN transistor as the optical cell and the third metal layer. Using these building blocks, two full size intelligent sensor designs, a 5.0V design and a 3.3V design, have been built and fabricated. With the BiCMOS technology chosen, the modular design of the building blocks, a better design practice, and a better technology choice, an existing design has been improved and problems associated with other neural network implementations have been reduced.
Chapter 4.

Training A Neural Network

4.1 Introduction

The neuron used in this neural network was designed to closely approximate the sigmoidal function. Traditionally, all analysis and training for the neural network used the sigmoidal function as the activation function. The sigmoidal function was chosen for its differentiable property which is a requirement of the Back Propagation training algorithm. This training algorithm is explained in detail in Section 2.5.1. To achieve proper operation of the trained neural network, the activation function of the actual neuron used in this project must be as close as possible to the sigmoidal function.

There are three possible sources of error using the present method of training. The first is the assumption that the neuron is equivalent to the theoretical model. The second is the introduction of quantization of the weights, which is a result of using a digital weighting scheme. The third is a result of the distributed neuron hardware structure used in this neural network architecture. This scaling property of the neuron is never considered in the traditional method of training the network. The activation function for the distributed neuron is different for a different number of inputs, but in the traditional training algorithm, the activation function is the same for any number of inputs.

This chapter will begin with a comparison of the two activation functions. This will be followed by a section describing the graphical LUT_BP training and recall simulator
which uses a look-up-table as the activation function for the Back Propagation training algorithm. The chapter will end with a section that compares results of a program which trains a neural network using the sigmoidal function, and then using the actual neuron. The program then compares simulation results of the neural network for the two training methods.

4.2 A Comparison of the Sigmoidal and Actual Activation Functions
The first activation function is the sigmoidal activation function. This is sometimes given as the hyperbolic tan function (tanh) which has an output range of -1 to 1. Since our real neuron's activation function has a range between 0 and 5V, a sigmoidal activation function with a range of 0 to 1 was chosen so that a simple scaling could be used. The sigmoidal function is given by Equation 18.

\[
g(x) = \frac{1}{1 + e^{-2x}}
\]

*Equation 18*

where \( x \) is the input to the neuron. The input range of the real neuron is -78\( \mu \)A to 78\( \mu \)A. For the sigmoidal function, this corresponds to a range from -5 to 5. Again, scaling was used. Figure 4-1 graphs the two activation functions.
Figure 4-1: A Comparison Of The Two Activation Functions

For the real neuron, actual simulation results were used. While very similar, the two functions do have definite distinctions. The real neuron's activation function is not as smooth as the sigmoidal function. At the middle of the graph, notice how steep the slope of the real neuron's function is when compared to the sigmoidal function. Figure 4-2 plots the difference between the two functions.

Figure 4-2: The Difference Between The Two Activation Functions

The biggest differences between the two functions occur around the middle of the graph. The error is as great as 15%.
One other factor in our neural network is the quantization of the weights. There are 31 separate weight values, -15 to 15. Figure 4-3 shows graphs of both the sigmoidal and the actual activation functions with the effects of quantization.

![Graph showing actual vs theoretical quantized activation function]

*Figure 4-3: A Comparison Of The Two Quantized Activation Functions*

Figure 4-4 shows the difference between the two quantized activation functions.

![Graph showing difference between quantized actual vs theoretical]

*Figure 4-4: The Difference Between The Two Quantized Activation Functions*

The quantization actually had the effect of reducing the difference between the two functions. To find a quantitative measure of the reduction, the area under the absolute value of the difference between the two functions was calculated. The result for the nonquantized error was determined to be 0.3354, and for the quantized error, 0.3114. This is a 7% decrease in error.
To see these results, the program RELVSTHY.M is included and it’s listing can be found in Appendix A.

4.3 The LUT_BP Training Program
A program called the LUT_BP training/recall simulator was written to reduce the errors associated with the traditional training methods used for neural networks due to the implementation of the activation function. The program was written in MATLAB using components of the Neural Network Toolbox. Components of the toolbox were modified and new programs were written to integrate with the toolbox.

4.3.1 Difference Between The Activation Functions
The first source of error arises from the difference in activation functions of the theoretical versus the actual which was discussed above. To overcome this error, a model of the neuron was written which uses actual Hspice simulation results of the neuron. This neuron model acts as a look-up table using linear interpolation between the points to make the curve smooth and differentiable. This model was written to be used with the Back Propagation training module included in MATLAB.

4.3.2 The Scaling Property Of The Neuron
The second source of error for training this neural network is due to the scaling property of this distributed neuron architecture. Traditional training algorithms cannot take this property into account and the same neuron activation function is used for the entire training process. The Back Propagation training module was modified to scale the neuron depending on the number of inputs. This avoids the problem of an activation function looking like a hard-limiting function when there are several inputs.
4.3.3 The Quantization Of Weights
The third source of error is a result of the digital quantization of the output of the training process, the weights. In this case, the weights are quantized into 5 bits, -15 to 15, or 31 separate values. This quantization cannot be done during the training process. The Back Propagation algorithm requires a smooth, differentiable activation function and with quantization, the algorithm would never converge. Therefore, quantization cannot be done until after the training process.

4.3.4 The LUT_BP Training/Recall Simulator
The LUT_BP Training/Recall Simulator is a graphical front end to the Back Propagation training algorithm using the enhancements to reduce error. A screen shot is shown in Figure 4-5.

![Figure 4-5: Screen Shot Of LUT_BP](image)

This screen gives a list of all the parameters of the Back Propagation training algorithm. There is also a list of network parameters where the size of the 2-layer neural network can
be specified and as well, the number of patterns to be trained. Selecting the Setup Patterns button opens the screen shown in Figure 4-6.

![Figure 4-6: Training Patterns](image)

By selecting the boxes in the Input Patterns section and the outputs in the Output Patterns section, the neural network can then be trained. Selecting the train button will complete the training using the training parameters and the training patterns entered. A window which shows the training progress will be opened and an example of this is shown in Figure 4-7.
Figure 4-7 shows two graphs. The first graph shows the error involved throughout the training process and the second graph is a graph of the learning rate. The learning rate is a measure of how quickly the network is being trained.

When the training process has finished and the error tolerance has been reached, the program begins its simulation mode which quantizes the weights from -15 to 15, and then simulates the network. An example output for a network with two outputs, being trained for four patterns, to an error level of 0.4 is as follows.

TRAIN: 0/1000 epochs, lr = 0.01, SSE = 2.23147.
TRAIN: 25/1000 epochs, lr = 1.48231, SSE = 1.84934.
TRAIN: 50/1000 epochs, lr = 10.1153, SSE = 1.7191.
TRAIN: 61/1000 epochs, lr = 117.758, SSE = 0.395548.

All done!

Simulation Results After Weights Rounded
0.6252 0.2932 0.6342 0.2789
0.3006 0.6151 0.5351 0.3180

SSEsim = 0.9938
The results of this training process is stored in two files - out.dat and results.dat. The first file out.dat, stores all of the weights associated with each training run and all the errors associated with the training and simulation. The second file results.dat, just stores the error associated with the training and the simulation so that multiple runs can be easily compared to find the best results.

The program LUT_BP.M is included and can be run and the program listing for this file can be found in Appendix B.

4.4 A Comparison Of The Training Methods
This section will examine the affect of training a neural network using a theoretical activation function by first training the network, and then simulating the network using the actual neural network. This will be compared with the new method of training the network using the actual activation function. This will determine whether the difference between the two neurons shown previously, will have a marked affect on the performance of the neural network.

4.4.1 How The Comparison Was Done
A program was written in MATLAB to do the comparison. The program begins with a neural network with 8 inputs, 6 hidden layer neurons, and 4 outputs. The program was run in a loop, 100 times. For each iteration of the program, the neural network was trained for 6 different, random input patterns. First, the neural network was trained using the same algorithm used in the LUT_BP program to an error of 0.05. When the training finished, the weights were quantized -15 to 15 and the network was then simulated. The error calculated from this simulation was stored in a file. Next, the same 6 random input
patterns were trained in the traditional way - using a sigmoidal function, also to an error of 0.05. When this training finished, the weights were quantized, as before, and then simulated using the actual neuron. The error calculated from this simulation was stored in the same file. After running this program through it's 100 iterations, the results were written to a file and then tabulated.

4.4.2 The Results Of The Comparison
The file written by the program stored several key pieces of information. It stored the number of training epochs required to reach the desired error level, and the error associated with the simulation of the network after the weights were quantized. The weights were quantized between -15 to 15 using two different methods. The first method was to round the weights to the nearest whole number and if any weight was greater than 15, it was made 15. Even though this is the only correct way to quantize the weights, another method was used. This second method took the largest absolute weight and scaled it to 15. The other weights were scaled the same amount and then rounded to the nearest integer. This is an incorrect assumption but will help to demonstrate a concept.

Table 1 lists the results.

<table>
<thead>
<tr>
<th></th>
<th>Using LUT_BP Method</th>
<th>Using Traditional Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Std. Dev.</td>
</tr>
<tr>
<td>Number of Training Epochs</td>
<td>107.5</td>
<td>48.2</td>
</tr>
<tr>
<td>Error With Rounded Weights</td>
<td>2.56</td>
<td>0.9</td>
</tr>
<tr>
<td>Error With Scaled Weights</td>
<td>3.32</td>
<td>0.4</td>
</tr>
<tr>
<td>Error Choosing Best Method</td>
<td>2.52</td>
<td>0.7</td>
</tr>
</tbody>
</table>

*Table 1: Comparison of Training Results*

The results in Table 1 are the averages taken from the 100 trainings of the neural network. The complete table of results can be found in Appendix C.
The first line of Table 1 shows the average number of training epochs to reach an error of 0.05 in training. The training using the sigmoidal function produced results much quicker than using the actual neuron. Using the LUT_BP training method took about 30% longer to converge than using the sigmoidal function. This is due to the fact that the actual activation function is not a completely smooth curve and is not truly differentiable. It differentiates to only an approximate value. Also notice the standard deviation. It is much higher when using the LUT_BP training method. This again is due to the fact that the activation function is not truly differentiable.

The second line is perhaps the most important. This shows the error associated with the network after the weights are rounded to the nearest integer between -15 and 15. The LUT_BP training algorithm produced results with almost half the error associated with using the sigmoidal function and using a traditional training algorithm. This discrepancy is due to the difference of the activation functions, and as well the scaling property of the neuron. Again, the standard deviation is higher for the LUT_BP training method and is due to the undifferentiability of the activation function.

The third line illustrates some interesting results. This line shows the error associated with the simulation of the neural network using weights which were scaled between -15 and 15. For the LUT_BP training method, the error increases which is expected due to the fact the weights are being changed. For the traditional training method, the error actually decreased. When the weights of the two training methods were compared prior to scaling, the weights associated with the traditional method were always smaller than the weights associated with the LUT_BP training. This is due to the fact that the scaling
property of the neuron is not accounted for. When those weights are combined during the simulation of the neural network, the sigmoidal function does not change. Thus, the weights appear very large and the neuron looks much more like a hard-limiting function. When these weights are simulated on a neural network which scales it's neuron depending on the number of inputs, these weights seem small and a great deal of error results. Therefore, this scaling of the weights actually increases the weights and when simulated, the weights operate closer to the range in which the training intended them to be used. Therefore, the error decreases.

The fourth line shows the minimum error associated with both quantization methods. The LUT_BP training method almost always had better results just rounding the weights where the traditional method always had better results with the scaling method.

The file CMPR.M is included and it’s source code can be found in Appendix D.

4.5 Conclusions
The three problems associated with training this type of neural network were presented and discussed. These problems are the difference between the actual and theoretical activation function, the quantization of weights, and the scaling property of the neuron used in this architecture. Two of these problems were handled by modifying the training method and the results of these modifications were discussed. The error was reduced to almost half when compared to a traditional training method. Also, a graphical interface to this new training algorithm was introduced and discussed.
Chapter 5.

Verification Of The Intelligent Sensor

5.1 Introduction

Overall, three chips were created and fabricated. The first chip was an optical cell test chip which had several optical cell designs. The last two chips were 5.0V and 3.3V designs of the full size intelligent sensor. These designs also included some test circuits. This chapter will discuss the testing results of the three chips and the implications of the results to the functionality of the intelligent sensor.

5.2 The Optical Cell Test Chip

This was a simple test chip which included 6 different optical cell designs. The designs varied in the size and geometry of the base of the photo-transistor, and as well, some included a Darlington configured photo-transistor. The tests were done using a Wentworth microscope, a variable power supply, the HP 34401A multimeter, and several light sources.

The tests performed on the optical cells were very simple. The variable power supply was used for the dc bias voltage for each optical cell as different light sources were used. The optical cells were tested for sensitivity to different ambient light levels, and light intensity required to turn the optical cell on. Some of the optical cells with smaller photo-transistors and little gain were completely non-functional. The designs that were
functional was the optical cell with the Darlington configured photo-transistor and a small base, and an optical cell with a photo-transistor with a very large base.

The optical cell is a threshold device as it has a digital output. The desirable characteristics of the optical cell is to be able to adjust the bias voltage for several different levels of ambient light, and to be sensitive enough to turn on when a light source with an intensity greater than the ambient light is used. With the two optical cells that met these requirements, the optical cell with the Darlington configured photo-transistor was chosen due to its higher sensitivity after being adjusted for the ambient light level. After the optical cell with the photo-transistor with the very large base was adjusted for the ambient light intensity, a light source with an intensity much larger than the ambient light was required to turn the optical cell on. The Darlington configured optical cell required a light source with an intensity much less than the other optical cell to turn on. This is due to the much higher gain that the Darlington configured optical cell exhibits. For this reason, the Darlington configured optical cell was chosen for the optical cell of the intelligent sensor design.

5.3 *The Intelligent Sensor Chips*

Due to the optical coupling and the density of the optical sensors, and the lack of an adequate functional data generator, the intelligent sensor is a very difficult chip to test. In order to characterize any of the building blocks of the neural network, test cells had to be added. These test cells were placed around the intelligent sensor and included neuron elements, complete synapse modules, and optical cells. A couple of sets of tests were run on the two intelligent sensor chips. The first set of tests were used to characterize the
neuron element. The second set of tests were used to characterize the complete neuron-
synapse.

5.3.1 Characterizing The Neuron
The first tests of the intelligent sensor characterized the neuron element and required a HP workstation with VeeTest software, and CMC's TH1000 test head. There are five blocks of neurons which each consisted of five neuron elements connected in parallel. Three of the blocks were placed on three corners of the chip. The remaining two blocks were placed side by side and had the neuron elements spread across the chip the exact distance the neuron elements were spread in the actual intelligent sensor.

To perform the tests, a program was written using HP's VeeTest software to control the test head. This program controlled the test head's current source and when measurements were taken. The measurements were automatically graphed and saved on the HP workstation. The tests of each of these blocks of neurons consisted of sweeping the input current from -390μA to 390μA for the 5.0V design and from -104μA to 104μA for the 3.3V design. The output voltage was then measured and plotted. Figure 5-1 shows the average waveform measured from the three corners of the chip.
This activation function is very close to the simulation results discussed in Chapter 3.

Figure 5-2 shows a comparison between the actual results and the simulation results.

Comparing the differences of the waveforms from each of the three corners gives some insight to the variation of the process parameters across the chip. For each point on the average curve shown in Figure 5-1, the maximum variation from this average from the three corners of the chip were taken. The absolute value of this maximum variation is shown in Figure 5-3.
The maximum variation occurs when the input current is at its minimum and maximum. The average variation is around 0.03V which is very good. The standard deviation of this curve is 0.0145.

The tests on the neuron blocks which had the neurons spread out had almost identical results. There was little error between the simulation results and the actual results. However, the variation of the curves from the average curve was much smaller than the three corners. The average variation was around 0.005V. The standard deviation of this maximum variation is 0.0025 which is much smaller than the results from the corners of the chip. Therefore, the neurons across the intelligent sensor will actually be more closely matched than the test blocks found on the three corners.

The tests on the 3.3V intelligent sensor had very similar results. The activation function again nearly matched the simulation results, and the variation measured from corner to corner was about the same as the 5.0V design. Again, the variation for the neurons which
were spread out was almost zero. The average activation function for the 3.3V design is shown in Figure 5-4.

![Graph showing voltage vs. current](image)

*Figure 5-4: Average Activation Function Of 3.3V Design*

5.3.2 Characterizing The Synapse

Characterizing the synapse was done using a separate MDAC connected to a neuron through a pin on the chip. While this setup is very flexible, it may be the main source of error. For these tests, a HP 8180A data generator, a HP 34401A multimeter, and a Tektronix 11402A oscilloscope was used. The waveform data was generated by the oscilloscope and stored on a computer.

For this test, the data generator was connected to the digital weight inputs D0 to D4. These five inputs were cycled through -15 to 15 and the output voltage was measured. Figure 5-5 shows the results of this measurement.
Figure 5-5: The Synapse

As the digital weights are stepped from -15 to 15, the maximum voltage of 5V and the minimum voltage of 0V is never reached. The maximum voltage measured in this case is 4.25V and the minimum is 0.34V. This is a significant difference between the simulation results and the test results. Since the neuron test results from the previous section agreed so closely to the simulation results, the Spice models of the two blocks cannot be blamed, but it is possible that Hspice modeled the MDAC as a better current source than it actually is.

5.4 Conclusions

The main building blocks of the intelligent sensor have been tested and characterized. The test results closely matched the simulation results and a possible source of error in the synapse test has been identified. As well, the variation of process parameters has also been discussed. However, there is still much further testing which can be completed on the intelligent sensor chips. There are several optical cells which have pins in which the photo-current can be measured with respect to different light intensities. Also, testing of the complete intelligent sensor has not yet been done. This would require a method of
illumination of the 8x8 array of optical cells and as well, a highly programmable data generator in which the weights of the neural network could be set. All of these tests would be required to complete the testing of the intelligent sensor.
Chapter 6.

Conclusions

6.1 Conclusions
A BiCMOS implementation of a programmable optical sensor has been presented. This design marks an improvement from a design in a 1.2\(\mu\) CMOS process in area, and performance. A new training method for the design has been introduced and reduces much of the error associated with previous training methods. In total, three chips have been fabricated and tested. Throughout this project, a level of expertise in the VLSI design tools used and the VLSI testing tools has been displayed.

The implementation of the intelligent sensor presented in this thesis improves on a previous design. The improvements are a result of design practices used and the benefits of the BiCMOS technology chosen. The design practices used resulted in a neuron which more closely matched the MDAC building block. As well, minimum spacing and transistor sizing was used where applicable to minimize the area used. The extra metal layer offered in the BiCMOS process from the process used in the previous design increased interconnection density. As well, in the BiCMOS technology, a true NPN transistor could be used for the photo-sensitive device which marks a performance improvement to the previous design which uses a parasitic device as the photo-sensitive element. Both a lower power 3.3V and a 5.0V intelligent sensor have been completed. The low power design marks a huge power savings over the 5.0V design.
A new training method based on the back-propagation training algorithm for the intelligent sensor has also been presented. The three problems associated with training this type of neural network were presented and discussed. Two of these problems were minimized by modifying the training method to more closely match the actual network, and the results of these improvements were discussed. The error was reduced by almost half when compared to a traditional training method. As well, a graphical interface to this training method was introduced.

In total, three chips were designed and fabricated. Both a lower power 3.3V and a 5.0V intelligent sensor have been completed. Both of these chips were tested and verified the building blocks of the intelligent sensor with the simulation results. The third chip, the optical cell test chip was also fabricated. This was also tested and was used to pick a design of the optical cell to be used in the intelligent sensor chips.

The building blocks of the intelligent sensor were characterized and laid out using Cadence 4.3 design tools. These building blocks were simulated and verified in Hspice. The three chips fabricated were tested with many tools in the lab. These included a Wentworth probe station, HP's VeeTest software running on a HP workstation. The VeeTest software was used to control CMC's TH1000 testhead. As well, an HP 8180A data generator and a Tektronix 11402A oscilloscope was used. This complete project has demonstrated a level of expertise in these VLSI design tools, and VLSI test tools.

6.2 Future Directions
Future work should aim towards defining a definite application for the design and building an intelligent sensor to match the design requirements. This will include
defining the type of patterns to be recognized which would affect the size and complexity of the neural network. This could involve changing the number of layers and number of neurons on each layer. Also, the resolution of the array of optical cells would also have to be defined. This would include the magnification required for the illumination and what method of illumination would be used. As well, if the patterns to be trained are going to be part of a standard set, the area taken up by the memory cells could be eliminated as programming is not a necessary feature.

Also, there is much more testing which could be completed on the intelligent sensor chips. There are several optical cells which have hooks to allow for a measurement of the photo-current generated. As well, the intelligent sensor is not completely tested. A method of illumination for the 8x8 array of optical cells is required, and as well, a highly programmable data generator to program the weights of the network. This type of testing could also be used to show the performance of the new training method. Presently, the new training method has only been verified through simulation.
References


Appendix A

RELVSTHY.M

This file shows comparisons between the theoretical activation function tanh, and the actual activation function from simulation.

echo on;
clear;

% Load the file containing the actual activation function and make global for the neuron functions
load neuron.dat;
global neuron;

% We will scale -78uA to 78uA of the actual simulation to -5 to 5
%mxx=5;
mnx=-5;
x=mnx:0.1:mxx;

% Get the actual
ya=neur6(x);

% Get the theoretical
yt=logsig(x);

% Plot for a comparison
plot(x,ya,'b-',x,yt,'r-')
title('Actual vs Theoretical Activation Function');
xlabel('Current (Scaled)');
ylabel('Voltage (Scaled)');
pause;

% Find the difference between the 2
diff=ya-yt;

% Plot the difference
plot(x,diff,'b-')
title('Difference Between Actual vs Theoretical');
xlabel('Current (Scaled)');
ylabel('Voltage (Scaled)');
pause;

% Plot the Absolute difference
absdiff=abs(diff);
plot(x,absdiff,'b-')
title('Absolute Difference Between Actual vs Theoretical');
xlabel('Current (Scaled)');
ylabel('Voltage (Scaled)');
pause;

% Use the Newton Cote's Rule for a Estimation of the error (integral)
% Approx_Err=quad8('difffunc',-5,5,0.1);
% Approx_Err=mean(absdiff)*(mxx-mnx);
Approx_Err
pause;

% Now lets look at the effect of quantization
% We will quantize it with 31 steps (-15 to 15)

step_size=(mxx-mnx)/30;
half_step=step_size/2;
q=0;
for n=mnx:0.1:mxx
    q=q+1;
    for i=0:30
        if n>(mnx-half_step+i*step_size) & n<=(mnx+half_step+i*step_size)
            quant_n=mnx+i*step_size;
        end
    end
    ya_q(q)=neurb(quant_n);
yt_q(q)=logsig(quant_n);
end

% Now lets plot the comparison of the 2 quantized activation functions
plot(x,ya_q,'b-';x,yt_q,'r-')
title('Actual vs Theoretical Quantized Activation Function');
xlabel('Current (Scaled)');
ylabel('Voltage (Scaled)');
pause;

% Find the difference between the 2
diff=ya_q-yt_q;
% Plot the difference
plot(x,diff,'b-')
title('Difference Between Quantized Actual vs Theoretical');
xlabel('Current (Scaled)');
ylabel('Voltage (Scaled)');
pause;

% Plot the Absolute difference
absdiff=abs(diff);
plot(x,absdiff,'b-')
title('Absolute Difference Between Quantized Actual vs Theoretical');
xlabel('Current (Scaled)');
ylabel('Voltage (Scaled)');
pause;

% Use the Newton Cote's Rule for a Estimation of the error (integral)
% Approx_Err=quad8('difffunc',-5,5,0.1);
% Approx_Err=mean(absdiff)*(mxx-mnx);
Approx_Err
Appendix B

LUT_BP.M

function lut_bp(action);

% This is a graphical interface to the LUT_BP training algorithm
% All results are stored in out.dat and a short form is stored in results.dat
% Results.dat has the following three unlabeled values
% Training Number   Training Error   Simulation Error
% where Training Number will be between 1 and Number of Runs

% Possible actions:
% initialize
% numepochs
% errtol
% lr
% lri
% lrd
% mom
% runs
% numinputs
% numhid
% numoutputs
% numpatterns
% setuppatterns
% train
% close

% Defaults
% TP(1) - Epochs between updating display, default = 25.
% TP(2) - Maximum number of epochs to train, default = 1000.
% TP(3) - Sum-squared error goal, default = 0.05.
% TP(4) - Learning rate, 0.01.
% TP(5) - Learning rate increase, default = 1.25.
% TP(6) - Learning rate decrease, default = 0.7.
% TP(7) - Momentum constant, default = 0.85.
% TP(8) - Maximum error ratio, default = 1.04.

if nargin<1,
    action='initialize';
end;

global TPglob
load f:\matlab\neuron.dat;
global neuron

% Defaults
tp=[25,1000,0.05,0.01,1.25,0.7,.85,1.04];

if strcmp(action,'initialize'),
% Graphics initialization
oldFigNumber = watchon;
figNumber = figure;
set(figNumber, ... 'NumberTitle','off', ... 'Name','LUT_BP Training Algorithm', ... 'backingstore','off', ... 'Units','normalized');

% Information for all buttons
top=0.95;
bottom=0.05;
left=0.91;
yInitLabelPos=0.90;
btnWid = 0.08;
btnHt=0.05;
% Spacing between the label and the button for the same command
btnOffset=0.02;
% Spacing between the button and the next command's label
spacing=0.02;

% The Setup Patterns button
uicontrol( ... 'Style','push', ... 'Units','normalized', ... 'Position',[left-2*btnWid-3*spacing bottom+2*btnHt+3*spacing 2.5*btnWid 2*btnHt], ... 'String','Setup Patterns', ... 'Callback','lut_bp("setuppatterns")');

% The TRAIN button
uicontrol( ... 'Style','push', ... 'Units','normalized', ... 'Position',[left-3*btnWid-3*spacing bottom 2*btnWid 2*btnHt], ... 'String','Train', ... 'Callback','lut_bp("train")');

% The CLOSE button
done_button=uicontrol('Style','PushButton', ... 'Position',[left-btnWid-spacing bottom 2*btnWid 2*btnHt], ... 'Units','normalized', ... 'Callback','lut_bp("done")');,'String','Close');

text1=uicontrol('Style','text','Position',[.13 .95 .22 .04],... 'Units','normalized', 'BackgroundColor',[0.8 0.8 0.8],... 'ForegroundColor','red','String','Training Parameters');

text2=uicontrol('Style','text','Position',[.63 .95 .22 .04],... 'Units','normalized', 'BackgroundColor',[0.8 0.8 0.8],... 'ForegroundColor','red','String','Network Parameters');
M = uicontrol('Style', 'edit', 'Position', [.32 .21 .13 .03], ...
'Units', 'normalized', 'String', num2str(real(tp))), ...
'CallBack', lut_bp("mom"), 'BackgroundColor', 'white', ...
'ForegroundColor', 'black');

RUNSlabel = uicontrol('Style', 'text', 'Position', [.06 .08 .25 .03], ...
'Units', 'normalized', 'String', 'Number of Runs', ...
'BackgroundColor', [.5 0.5 0.5], ...
'ForegroundColor', 'white', 'Interruptible', 'no', 'Horiz', 'left');

tmp = 1;
RUNS = uicontrol('Style', 'edit', 'Position', [.32 .08 .13 .03], ...
'Units', 'normalized', 'String', num2str(real(tmp))), ...
'CallBack', lut_bp("runs"), 'BackgroundColor', 'white', ...
'ForegroundColor', 'black');

NIIlabel = uicontrol('Style', 'text', 'Position', [.56 .86 .25 .03], ...
'Units', 'normalized', 'String', 'Number of Inputs', ...
'BackgroundColor', [.5 0.5 0.5], ...
'ForegroundColor', 'white', 'Interruptible', 'no', 'Horiz', 'left');

tmp = 4;
NI = uicontrol('Style', 'edit', 'Position', [.82 .86 .13 .03], ...
'Units', 'normalized', 'String', num2str(real(tmp))), ...
'CallBack', lut_bp("numinputs"), 'BackgroundColor', 'white', ...
'ForegroundColor', 'black');

NHlabel = uicontrol('Style', 'text', 'Position', [.56 .73 .25 .03], ...
'Units', 'normalized', 'String', 'Hidden Layer Neurons', ...
'BackgroundColor', [.5 0.5 0.5], ...
'ForegroundColor', 'white', 'Interruptible', 'no', 'Horiz', 'left');

tmp = 3;
NH = uicontrol('Style', 'edit', 'Position', [.82 .73 .13 .03], ...
'Units', 'normalized', 'String', num2str(real(tmp))), ...
'CallBack', lut_bp("numhid"), 'BackgroundColor', 'white', ...
'ForegroundColor', 'black');

NOlabel = uicontrol('Style', 'text', 'Position', [.56 .60 .25 .03], ...
'Units', 'normalized', 'String', 'Number of Outputs', ...
'BackgroundColor', [.5 0.5 0.5], ...
'ForegroundColor', 'white', 'Interruptible', 'no', 'Horiz', 'left');

tmp = 2;
NO = uicontrol('Style', 'edit', 'Position', [.82 .60 .13 .03], ...
'Units', 'normalized', 'String', num2str(real(tmp))), ...
'CallBack', lut_bp("numoutputs"), 'BackgroundColor', 'white', ...
'ForegroundColor', 'black');

NPlabel = uicontrol('Style', 'text', 'Position', [.56 .47 .25 .03], ...
'Units', 'normalized', 'String', 'Number of Patterns', ...
'BackgroundColor', [.5 0.5 0.5], ...
'ForegroundColor', 'white', 'Interruptible', 'no', 'Horiz', 'left');

tmp = 4;
NP = uicontrol('Style', 'edit', 'Position', [.82 .47 .13 .03], ...
'Units', 'normalized', 'String', num2str(real(tmp)), ...
'Callback', 'lut_bp(\text{"numpatterns"})', 'Background', 'white', ...
'Foreground', 'black');

TPglob = [NE; ERRTOL; LR; LRI; LRD; M; RUNS; N; NH; NO; NP];

elseif strcmp(action, 'numepochs'),
  % set from edit text
  tmpval = str2num(get(TPglob(1), 'string'));
  %tp(2) = tmpval;

elseif strcmp(action, 'errtol'),
  % set from edit text
  tmpval = str2num(get(TPglob(2), 'string'));
  %tp(3) = tmpval;

elseif strcmp(action, 'lr'),
  % set from edit text
  tmpval = str2num(get(TPglob(3), 'string'));
  %tp(4) = tmpval;

elseif strcmp(action, 'lri'),
  % set from edit text
  tmpval = str2num(get(TPglob(4), 'string'));
  %tp(5) = tmpval;

elseif strcmp(action, 'lrD'),
  % set from edit text
  tmpval = str2num(get(TPglob(5), 'string'));
  %tp(6) = tmpval;

elseif strcmp(action, 'mom'),
  % set from edit text
  tmpval = str2num(get(TPglob(6), 'string'));
  %tp(7) = tmpval;

elseif strcmp(action, 'runs'),
  % set from edit text
  tmpval = str2num(get(TPglob(7), 'string'));
  %tp(7) = tmpval;

elseif strcmp(action, 'numinputs'),
  % set from edit text
  tmpval = str2num(get(TPglob(8), 'string'));
  %tp(7) = tmpval;

elseif strcmp(action, 'numhid'),
  % set from edit text
  tmpval = str2num(get(TPglob(9), 'string'));
  %tp(7) = tmpval;

elseif strcmp(action, 'numoutputs'),
  % set from edit text
  tmpval = str2num(get(TPglob(10), 'string'));
elseif strcmp(action,'numpatterns'),
\% set from edit text
\% tmpval=str2num(get(TPglob(11),'string'));
\%tmp(7)=tmpval;

elseif strcmp(action,'train'),
figNumber2 = figure;
set(figNumber2,...
'NumberTitle','off', ...
'Name','LUT_BP Training', ...
'backingstore','off',...
'Units','normalized');
\%set(gca,'Pointer','watch');
tmpval=str2num(get(TPglob(1),'string'));
tp(2)=tmpval;
tmpval=str2num(get(TPglob(2),'string'));
tp(3)=tmpval;
tmpval=str2num(get(TPglob(3),'string'));
tp(4)=tmpval;
tmpval=str2num(get(TPglob(4),'string'));
tp(5)=tmpval;
tmpval=str2num(get(TPglob(5),'string'));
tp(6)=tmpval;
tmpval=str2num(get(TPglob(6),'string'));
tp(7)=tmpval;
ntimes=str2num(get(TPglob(7),'string'));
nofinputs=str2num(get(TPglob(8),'string'));
nofhid=str2num(get(TPglob(9),'string'));
nofoutputs=str2num(get(TPglob(10),'string'));
nofpatterns=str2num(get(TPglob(11),'string'));

strt=11;
for pi=1:nofpatterns
  for pj=1:nofinputs
    ind=strt+pj+(pi-1)*nofinputs;
    val=get(TPglob(ind),'Value');
    p(pi,pj)=val;
  end
end
p=p';

strt=11+nofpatterns*nofinputs;
for ti=1:nofpatterns
  for tj=1:nofoutputs
    ind=strt+tj+(ti-1)*nofoutputs;
    val=get(TPglob(ind),'Value');
    t(ti,tj)=val;
  end
end
t=t';

pinit=rand(nofinputs,2);
pinit=pinit/2.5;
load f:\matlab\neuron.dat;
global neuron

fid=fopen('f:\matlab\out.dat','w+');

for num=1:nTimes

[w1,b1,w2,b2]=initff(pinit,nofhid,'neur6',nofoutputs,'neur6');

%scale weights between -0.5 and 0.5 (offers better results)
maxw1=max(w1);
minw1=min(w1);
maxw2=max(w2);
minw2=min(w2);
maxb1=max(b1);
maxb2=max(b2);
minb1=min(b1);
minb2=min(b2);
temp1=[maxw1,maxw2,maxb1,maxb2];
temp2=[minw1,minw2,minb1,minb2];
mxw=max(temp1);
minw=min(temp2);
mid=(mxw-minw)/2+minw;
w1=w1-mid;
w2=w2-mid;
b1=b1-mid;
b2=b2-mid;
sc_fact=10/(mxw-minw);
w1=w1*sc_fact/10;
w2=w2*sc_fact/10;
b1=b1*sc_fact/10;
b2=b2*sc_fact/10;

%Train
[w1,b1,w2,b2,i.tr] = bartbp(w1,b1,'neur6',w2,b2,'neur6',p,t,tp);
disp(' ');
disp('All done!')
maxw1=max(w1);
minw1=min(w1);
maxw2=max(w2);
minw2=min(w2);
maxb1=max(b1);
maxb2=max(b2);
minb1=min(b1);
minb2=min(b2);
temp1=[maxw1,maxw2,maxb1,maxb2];
temp2=[minw1,minw2,minb1,minb2];
mxw=max(temp1);
minw=min(temp2);
range=mxw-minw;

%Storing Results in a file
fprintf(fid,'Finished RUN: %10f Epochs = %10f\n',num,i);
fprintf(fid,'W1\n');
[prntl,prntc]=size(w1);
for pr=1:prnt
    for pc=1:prntc
        tmpprt=w1(pr,pc);
        fprintf(fid,'%10.5f',tmpprt);
    end
    fprintf(fid,'\n');
end
fprintf(fid,'B1\n');
fprintf(fid,'%10.5f \n',b1);
fprintf(fid,'W2\n');
[prntl,prntc]=size(w2);
for pr=1:prnt
    for pc=1:prntc
        tmpprt=w2(pr,pc);
        fprintf(fid,'%10.5f',tmpprt);
    end
    fprintf(fid,'\n');
end
fprintf(fid,'B2\n');
fprintf(fid,'%10.5f \n',b2);
fprintf(fid,'SSE= %10.5f \n',tr(1,i+1));
results(num,1)=num;
results(num,2)=tr(1,i+1);

%Round Weights To Integers
w1=round(w1);
w2=round(w2);
b1=round(b1);
b2=round(b2);

%Get rid of any values > 15
[mx_i,mx_j]=size(w1);
for in_i=1:mx_i
    for in_j=1:mx_j
        if w1(in_i, in_j) > 15
            w1(in_i, in_j) = 15;
        end
        if w1(in_i, in_j) < -15
            w1(in_i, in_j) = -15;
        end
    end
end

[mx_i,mx_j]=size(b1);
for in_i=1:mx_i
    for in_j=1:mx_j
        if b1(in_i, in_j) > 15
            b1(in_i, in_j) = 15;
        end
        if b1(in_i, in_j) < -15
            b1(in_i, in_j) = -15;
        end
    end
end

[mx_i,mx_j]=size(w2);
for in_i=1:mx_i
    for in_j=1:mx_j
        if w2(in_i, in_j) > 15
            w2(in_i, in_j) = 15;
        end
        if w2(in_i, in_j) < -15
            w2(in_i, in_j) = -15;
        end
    end
end
[mx_i,mx_j]=size(b2);
for in_i=1:mx_i
    for in_j=1:mx_j
        if b2(in_i, in_j) > 15
            b2(in_i, in_j) = 15;
        end
        if b2(in_i, in_j) < -15
            b2(in_i, in_j) = -15;
        end
    end
end

%SImulate Network With Rounded Weights
[in_num,temp]=size(p);
a1_init=w1*p*5.2e-6;
b1s=b1*5.2e-6;
a1=feval('neur5',a1_init,b1s,in_num);
[in_num,temp]=size(a1);
a2_init=w2*a1*5.2e-6;
b2s=b2*5.2e-6;
a2=feval('neur5',a2_init,b2s,in_num);
fprintf(fid,'Results Rounded\n');
[prnt,prntc]=size(a2);
for pr=1:prnt
    for pc=1:prntc
        tmpprnt=a2(pr,pc);
        fprintf(fid,'%10.5f',tmpprnt);
    end
fprintf(fid,'\n');
end
fprintf(fid,'%10.5f %10.5f %10.5f %10.5f \n',a2);
e=(t-a2);
SSEsim=sumsq(e);
fprintf(fid,'SSEsim= %10.5f\n\n',SSEsim);
results(num,3)=SSEsim;
disp('')
disp('Simulation Results After Weights Rounded')
f_res=a2;
disp(f_res)
disp('SSEsim'=)
disp(SSEsim)
end
fclose(fid);
 fid=fopen('f:\matlab\results.dat','w+');
 fprintf(fid,'%10.5f %10.5f %10.5f \n',results);
fclose(fid);

elseif strcmp(action,'setuppatterns'),
    figNumber3 = figure;
    set(figNumber3, ...'NumberTitle','off', ...'Name','LUT_BP Training Patterns', ...'backingstore','off', ...
        'Units','normalized');
    noinputsn=str2num(get(TPglob(8),'string'));
    nooutputsn=str2num(get(TPglob(10),'string'));
    nopatternsn=str2num(get(TPglob(11),'string'));
    ratioi2o=noinputsn/nooutputsn;
    inspace=ratioi2o/(ratioi2o+1)*0.95;
    outspace=1/(ratioi2o+1)*0.95;
    isperin=inspace/noinputsn;
    osperout=outspace/nooutputsn;
    ssperpatt=0.9/nopatternsn;

    indx=(inspace-0.22)/2;
    text1=uicontrol('Style','text','Position',[indx .95 .22 .04],...,}
        'Units','normalized','BackgroundColor',[0.8 0.8 0.8],...
        'ForegroundColor','red','String','Input Patterns');

    indx=inspace/0.95+(outspace-0.22)/2;
    text2=uicontrol('Style','text','Position',[indx .95 .22 .04],...,}
        'Units','normalized','BackgroundColor',[0.8 0.8 0.8],...
        'ForegroundColor','red','String','Output Patterns');

    %strt=length(TPglob);
    strt=11;

    for q=1:nopatternsn
        for w=1:noinputsn
            index=strt+w+(q-1)*noinputsn;
            indx=(isperin-0.03)/2+(w-1)*isperin;
            indy=0.9-(ssperpatt-(ssperpatt-0.03)/2+ssperpatt*(q-1));
            TPglob(index)=uicontrol('Style','checkbox','BackgroundColor',...
                'white','Units','normalized','Position',[indx indy 0.03 0.03]);
        end
    end
    %strt=length(TPglob);
    strt=11+npatternsn*noinputsn;

    for q=1:nopatternsn
        for w=1:nooutputsn
            index=strt+w+(q-1)*nooutputsn;
            indx=inspace/0.95+(osperout-0.03)/2+(w-1)*osperout;
            indy=0.9-(ssperpatt-(ssperpatt-0.03)/2+ssperpatt*(q-1));
            TPglob(index)=uicontrol('Style','checkbox','BackgroundColor',...
                'white','Units','normalized','Position',[indx indy 0.03 0.03]);
        end
    end

elseif strcmp(action,'done'),
close(gc);
clear global tp
close all;
clear global TPglob;
end
## Appendix C

### Comparison Results Of The Two Training Methods

<table>
<thead>
<tr>
<th>Using LUT</th>
<th>BP Method</th>
<th>Using Traditional Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epochs</td>
<td>SSE</td>
<td>Rounded</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>------------------------</td>
</tr>
<tr>
<td>132</td>
<td>0.0428</td>
<td>1.620429</td>
</tr>
<tr>
<td>119</td>
<td>0.0378</td>
<td>1.434352</td>
</tr>
<tr>
<td>79</td>
<td>0.0377</td>
<td>1.6564</td>
</tr>
<tr>
<td>101</td>
<td>0.0371</td>
<td>1.299837</td>
</tr>
<tr>
<td>147</td>
<td>0.0470</td>
<td>3.507476</td>
</tr>
<tr>
<td>49</td>
<td>0.0456</td>
<td>2.179584</td>
</tr>
<tr>
<td>86</td>
<td>0.039</td>
<td>1.849144</td>
</tr>
<tr>
<td>86</td>
<td>0.0408</td>
<td>2.152708</td>
</tr>
<tr>
<td>51</td>
<td>0.0418</td>
<td>3.674352</td>
</tr>
<tr>
<td>76</td>
<td>0.0487</td>
<td>1.5610568</td>
</tr>
<tr>
<td>106</td>
<td>0.0496</td>
<td>2.1201148</td>
</tr>
<tr>
<td>246</td>
<td>0.0493</td>
<td>1.7903068</td>
</tr>
<tr>
<td>112</td>
<td>0.0382</td>
<td>1.7779452</td>
</tr>
<tr>
<td>88</td>
<td>0.0356</td>
<td>2.6121204</td>
</tr>
<tr>
<td>87</td>
<td>0.0351</td>
<td>1.5121204</td>
</tr>
<tr>
<td>101</td>
<td>0.0418</td>
<td>1.2713363</td>
</tr>
<tr>
<td>102</td>
<td>0.0486</td>
<td>2.4893</td>
</tr>
<tr>
<td>78</td>
<td>0.0410</td>
<td>2.0054216</td>
</tr>
<tr>
<td>92</td>
<td>0.0231</td>
<td>2.008228</td>
</tr>
<tr>
<td>142</td>
<td>0.0497</td>
<td>1.4764228</td>
</tr>
<tr>
<td>87</td>
<td>0.0381</td>
<td>2.0346852</td>
</tr>
<tr>
<td>86</td>
<td>0.0438</td>
<td>2.507504</td>
</tr>
<tr>
<td>93</td>
<td>0.0326</td>
<td>2.3153562</td>
</tr>
<tr>
<td>80</td>
<td>0.0356</td>
<td>2.220062</td>
</tr>
<tr>
<td>97</td>
<td>0.0411</td>
<td>1.4607368</td>
</tr>
<tr>
<td>82</td>
<td>0.0467</td>
<td>2.56729</td>
</tr>
<tr>
<td>120</td>
<td>0.0453</td>
<td>2.7342412</td>
</tr>
<tr>
<td>286</td>
<td>0.0434</td>
<td>3.0004003</td>
</tr>
<tr>
<td>51</td>
<td>0.0352</td>
<td>1.8277296</td>
</tr>
<tr>
<td>66</td>
<td>0.0479</td>
<td>1.1245772</td>
</tr>
<tr>
<td>240</td>
<td>0.0349</td>
<td>1.3685872</td>
</tr>
<tr>
<td>116</td>
<td>0.0374</td>
<td>2.533444</td>
</tr>
<tr>
<td>75</td>
<td>0.0467</td>
<td>1.5936724</td>
</tr>
<tr>
<td>82</td>
<td>0.0475</td>
<td>1.9175956</td>
</tr>
<tr>
<td>85</td>
<td>0.0391</td>
<td>1.7763088</td>
</tr>
<tr>
<td>94</td>
<td>0.0468</td>
<td>2.5925189</td>
</tr>
<tr>
<td>161</td>
<td>0.0479</td>
<td>2.2515528</td>
</tr>
<tr>
<td>326</td>
<td>0.0498</td>
<td>2.996932</td>
</tr>
<tr>
<td>52</td>
<td>0.0465</td>
<td>1.374916</td>
</tr>
<tr>
<td>140</td>
<td>0.0498</td>
<td>1.8397908</td>
</tr>
<tr>
<td>104</td>
<td>0.0463</td>
<td>2.6490324</td>
</tr>
<tr>
<td>208</td>
<td>0.0283</td>
<td>2.3122216</td>
</tr>
<tr>
<td>108</td>
<td>0.0466</td>
<td>3.6902489</td>
</tr>
<tr>
<td>191</td>
<td>0.0372</td>
<td>3.3939385</td>
</tr>
<tr>
<td>75</td>
<td>0.0552</td>
<td>2.3591468</td>
</tr>
<tr>
<td>49</td>
<td>0.0457</td>
<td>1.8674292</td>
</tr>
<tr>
<td>53</td>
<td>0.0721</td>
<td>2.95623</td>
</tr>
<tr>
<td>94</td>
<td>0.049</td>
<td>2.220926</td>
</tr>
<tr>
<td>84</td>
<td>0.0425</td>
<td>2.9233928</td>
</tr>
<tr>
<td>261</td>
<td>0.0471</td>
<td>2.6218904</td>
</tr>
<tr>
<td>75</td>
<td>0.0489</td>
<td>2.3373992</td>
</tr>
<tr>
<td>194</td>
<td>0.0489</td>
<td>1.7373334</td>
</tr>
<tr>
<td>97</td>
<td>0.0475</td>
<td>2.654348</td>
</tr>
<tr>
<td>171</td>
<td>0.0497</td>
<td>3.1186488</td>
</tr>
<tr>
<td>293</td>
<td>0.0498</td>
<td>3.64473</td>
</tr>
</tbody>
</table>
Appendix D

CMPR.M

This program runs a comparison of 2 training techniques for a 2-layer neural network with a distributed neuron architecture. The first technique uses simulation results from an actual neuron and the second uses the traditional method - a theoretical neuron.

How many runs with different random patterns will be trained
Num_Runs=100;

Setting up training parameters

NE=1000;
ERRTOL=0.05;
LR=0.01;
LRI=1.25;
LRD=0.7;
M=0.85;
RUNS=1;
NI=8;
NH=6;
NO=4;
NP=6;

tp=[10;NE;ERRTOL;LR;LRI;LRD;M;RUNS;NI;NH;NO;NP];

Open up an output file
fid=fopen('F:\matlab\compare.dat','w+');

Load the neuron look-up table
load F:\matlab\neuron.dat;
global neuron

Begin
for i=1:Num_Runs

Help the rand function to be more random
rand('seed', sum(100*clock));
p=rand(NI,NP);

Get rid of identical input patterns
bad_inputs = 0;
while bad_inputs == 0
   for i=2:NPI
      if p(:,1) == p(:,i)
         bad_inputs = 1;
      end
   end
end
if bad_inputs == 1
    p = rand(NL, NP);
    bad_inputs = 0;
else
    break;
end

% Setup the output patterns
s = [0,1,0,0,0,1 ; 0,0,1,0,0,1 ; 0,0,0,1,0,1 ; 0,0,0,0,1,1];

p = round(p);
s = round(s);
disp('inputs')
disp(p)
disp('outputs')
disp(s)

% Initial p
pinit = rand(NL, 2);
pinit = pinit/2.5;

for num = 1:RUNS

w1, b1, w2, b2 = initff(pinit, NH, 'neur6', NO, 'neur6');

% Scale weights between -0.5 and 0.5 (offers better results)
maxw1 = max(w1);
minw1 = min(w1);
maxw2 = max(w2);
minw2 = min(w2);
maxb1 = max(b1);
maxb2 = max(b2);
minb1 = min(b1);
minb2 = min(b2);
temp1 = [maxw1, maxw2, maxb1, maxb2];
temp2 = [minw1, minw2, minb1, minb2];
mxw = max(temp1);
mnw = min(temp2);
mid = (mxw - mnw)/2 + mnw;
w1 = w1 - mid;
w2 = w2 - mid;
b1 = b1 - mid;
b2 = b2 - mid;
scale_fact = 10/(mxw - mnw);
w1_orig = w1 * scale_fact/10;
w2_orig = w2 * scale_fact/10;
b1_orig = b1 * scale_fact/10;
b2_orig = b2 * scale_fact/10;

% Beginning of actual

[w1, b1, w2, b2, i, tr] = bartbp(w1_orig, b1_orig, 'neur6', w2_orig, b2_orig, 'neur6', p, t, tp);
disp('')
disp('All done!')

D-73
maxw1=max(w1);
minw1=min(w1);
maxw2=max(w2);
minw2=min(w2);
maxb1=max(b1);
maxb2=max(b2);
minb1=min(b1);
minb2=min(b2);
temp1=[maxw1,maxw2,maxb1,maxb2];
temp2=[minw1,minw2,minb1,minb2];
mxw=max(temp1);
mnw=min(temp2);
range=mxw-mnw;

%Print results to a file
fprintf(fid,'Actual RUN: Epochs = %10.5f \n',i);
fprintf(fid,'W1\n');
[prntr,prntc]=size(w1);
for pr=1:prntr
    for pc=1:prntc
        tmprnt=w1(pr,pc);
        fprintf(fid,'%10.5f',tmprnt);
    end
    fprintf(fid,'\n');
end
fprintf(fid,'B1\n');
fprintf(fid,'%10.5f \n',b1);
fprintf(fid,'W2\n');
[prntr,prntc]=size(w2);
for pr=1:prntr
    for pc=1:prntc
        tmprnt=w2(pr,pc);
        fprintf(fid,'%10.5f',tmprnt);
    end
    fprintf(fid,'\n');
end
fprintf(fid,'B2\n');
fprintf(fid,'%10.5f \n',b2);
fprintf(fid,'SSE= %10.5f \n',tr(1,i+1));
results(num,1)=num;
results(num,2)=tr(1,i+1);
results(num,3)=range;

%Quantize the weights - scale and round
a_mx=max(abs(mxw),abs(mnw));
w1_1=w1*15/a_mx;
b1_1=b1*15/a_mx;
w2_1=w2*15/a_mx;
b2_1=b2*15/a_mx;
w1_1=round(w1_1);
w2_1=round(w2_1);
b1_1=round(b1_1);
b2_1=round(b2_1);
w1=round(w1);
w2=round(w2);
b1=round(b1);
b2=round(b2);

%Get rid of any values > 15
[mx_i,mx_j]=size(w1);
for in_i=1:mx_i
    for in_j=1:mx_j
        if w1(in_i, in_j) > 15
            w1(in_i, in_j) = 15;
        end
        if w1(in_i, in_j) < -15
            w1(in_i, in_j) = -15;
        end
    end
end
[mx_i,mx_j]=size(b1);
for in_i=1:mx_i
    for in_j=1:mx_j
        if b1(in_i, in_j) > 15
            b1(in_i, in_j) = 15;
        end
        if b1(in_i, in_j) < -15
            b1(in_i, in_j) = -15;
        end
    end
end
[mx_i,mx_j]=size(w2);
for in_i=1:mx_i
    for in_j=1:mx_j
        if w2(in_i, in_j) > 15
            w2(in_i, in_j) = 15;
        end
        if w2(in_i, in_j) < -15
            w2(in_i, in_j) = -15;
        end
    end
end
[mx_i,mx_j]=size(b2);
for in_i=1:mx_i
    for in_j=1:mx_j
        if b2(in_i, in_j) > 15
            b2(in_i, in_j) = 15;
        end
        if b2(in_i, in_j) < -15
            b2(in_i, in_j) = -15;
        end
    end
end

%simulate NN first with rounded weights
[in_num,temp]=size(p);
a1_init=w1*p*5.2e-6;
b1s=b1*5.2e-6;
a1=feval('neur5',a1_init,b1s,in_num);
[in_num.temp]=size(a1);
a2_init=w2*a1*5.2e-6;
b2s=b2*5.2e-6;
a2=feval('neur5',a2_init,b2s,in_num);
fprintf(fid,'Results Rounded\n');
[prtr,prntc]=size(a2);
for pr=1:prtr
    for pc=1:prntc
        tmpprnt=a2(pr,pc)*5;
        fprintf(fid,'%10.5f ',tmpprnt);
    end
    fprintf(fid,\n');
end
e=5*(t-a2);
SSEsim=sumsqr(e);
fprintf(fid,'SSEsim= %10.5f \n',SSEsim);
results(num,4)=SSEsim;
disp('')
disp('Results Rounded')
f_res=a2*5;
disp(f_res)
disp('SSEsim=')
disp(SSEsim)

%simulate NN with scaled weights
[in_num.temp]=size(p);
a1_init=w1*a1*5.2e-6;
b1s=b1*5.2e-6;
a1=feval('neur5',a1_init,b1s,in_num);
[in_num.temp]=size(a1);
a2_init=w2*a1*5.2e-6;
b2s=b2*5.2e-6;
a2=feval('neur5',a2_init,b2s,in_num);
fprintf(fid,'Results -15 to 15\n');
[prtr,prntc]=size(a2);
for pr=1:prtr
    for pc=1:prntc
        tmpprnt=a2(pr,pc)*5;
        fprintf(fid,'%10.5f ',tmpprnt);
    end
    fprintf(fid,\n');
end
e=5*(t-a2);
SSEsim=sumsqr(e);
fprintf(fid,'SSEsim= %10.5f \n',SSEsim);
results(num,4)=SSEsim;
disp('')
disp('Results -15 to 15')
f_res=a2*5;
disp(f_res)
disp('SSEsim=')
disp(SSEsim)

%Beginning of theoretical training
functions = tbpz2(w1_orig,b1_orig,'logsig',w2_orig,b2_orig,'logsig',p,t,tp);
disp('')
disp('All done!')
maxw1=max(w1);
minw1=min(w1);
maxw2=max(w2);
minw2=min(w2);
maxb1=max(b1);
maxb2=max(b2);
minb1=min(b1);
minb2=min(b2);
temp1=[maxw1,maxw2,maxb1,maxb2];
temp2=[minw1,minw2,minb1,minb2];
mxw=max(temp1);
mnw=min(temp2);
rangex=mxw-mnw;

% Output to results file
fprintf(fid,'Theoretical RUN: Epochs = %10.5f
',i);
fprintf(fid,'W1
');
[prntr,prntc]=size(w1);
for pr=1:prntr
  for pc=1:prntc
    tmpprnt=w1(pr,pc);
    fprintf(fid,'%10.5f ',tmpprnt);
  end
  fprintf(fid,'
');
end
fprintf(fid,'B1
');
fprintf(fid,'%10.5f
',b1);
fprintf(fid,'W2
');
[prntr,prntc]=size(w2);
for pr=1:prntr
  for pc=1:prntc
    tmpprnt=w2(pr,pc);
    fprintf(fid,'%10.5f ',tmpprnt);
  end
  fprintf(fid,'
');
end
fprintf(fid,'B2
');
fprintf(fid,'%10.5f
',b2);
fprintf(fid,'SSE = %10.5f
',tr(1,i+1));
results(num,1)=num;
results(num,2)=tr(1,i+1);
results(num,3)=rangex;

% Quantize the weights - both scale and round
a_mx=max(abs(mxw),abs(mnw));
w1_2=w1*15/a_mx;
b1_2=b1*15/a_mx;
w2_2=w2*15/a_mx;
b2_2=b2*15/a_mx;
w1_2=round(w1_2);
w2_2=round(w2_2);
% Get rid of any values > 15
[mx_i,mx_j]=size(w1);
for in_i=1:mx_i
    for in_j=1:mx_j
        if w1(in_i, in_j) > 15
            w1(in_i, in_j) = 15;
        end
        if w1(in_i, in_j) < -15
            w1(in_i, in_j) = -15;
        end
    end
end
[mx_i,mx_j]=size(b1);
for in_i=1:mx_i
    for in_j=1:mx_j
        if b1(in_i, in_j) > 15
            b1(in_i, in_j) = 15;
        end
        if b1(in_i, in_j) < -15
            b1(in_i, in_j) = -15;
        end
    end
end
[mx_i,mx_j]=size(w2);
for in_i=1:mx_i
    for in_j=1:mx_j
        if w2(in_i, in_j) > 15
            w2(in_i, in_j) = 15;
        end
        if w2(in_i, in_j) < -15
            w2(in_i, in_j) = -15;
        end
    end
end
[mx_i,mx_j]=size(b2);
for in_i=1:mx_i
    for in_j=1:mx_j
        if b2(in_i, in_j) > 15
            b2(in_i, in_j) = 15;
        end
        if b2(in_i, in_j) < -15
            b2(in_i, in_j) = -15;
        end
    end
end

% simulate NN first with rounded weights
[in_num,temp]=size(p);
a1_init=wl*p*5.2e-6;
b1s=b1*5.2e-6;
a1=feval('neur5',a1_init,b1s,in_num);
[in_num, temp]=size(a1);
a2_init=w2*a1*5.2e-6;
b2s=b2*5.2e-6;
a2=feval('neur5',a2_init,b2s,in_num);
fprintf(fid,'Results Rounded\n');
[prnt, prntc]=size(a2);
for pr=1:prnt
    for pc=1:prntc
        tmpprnt=a2(pr, pc)*5;
        fprintf(fid,'%10.5f ', tmpprnt);
    end
    fprintf(fid, '\n');
end
e=5*(t-a2);
SSEsim=sumsqr(e);
fprintf(fid, 'SSEsim= %10.5f \n', SSEsim);
results(num, 4)=SSEsim;
disp('')
disp('Results Rounded')
f_res=a2*5;
disp(f_res)
disp('SSEsim=')
disp(SSEsim)

% simulate NN f with scaled weights
[in_num, temp]=size(p);
a1_init=w1_2*p*5.2e-6;
b1s=b1_2*5.2e-6;
a1=feval('neur5',a1_init,b1s,in_num);
[in_num, temp]=size(a1);
a2_init=w2_2*a1*5.2e-6;
b2s=b2_2*5.2e-6;
a2=feval('neur5',a2_init,b2s,in_num);
fprintf(fid,'Results -15 to 15\n');
[prnt, prntc]=size(a2);
for pr=1:prnt
    for pc=1:prntc
        tmpprnt=a2(pr, pc)*5;
        fprintf(fid,'%10.5f ', tmpprnt);
    end
    fprintf(fid, '\n');
end
e=5*(t-a2);
SSEsim=sumsqr(e);
fprintf(fid, 'SSEsim= %10.5f \n', SSEsim);
results(num, 4)=SSEsim;
disp('')
disp('Results -15 to 15')
f_res=a2*5;
disp(f_res)
disp('SSEsim=')
disp(SSEsim)

end

end

fclose (fid)
Vita Auctoris

Robert Bart MacLean was born in 1972 in Windsor, Ontario. He graduated from F.J. Brennan High School in 1991. From there he went on to the University of Windsor where he obtained a B.A.Sc in Electrical Engineering in 1995. He is currently a candidate for the Master’s degree in Electrical Engineering and hopes to graduate in Fall 1998.