ARTIFICIAL NEURAL NETWORK CIRCUIT FOR SPECTRAL PATTERN RECOGNITION

A Thesis

by

FARAH RASHEED

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Jiang Hu
Committee Members,	Yufeng Ge
	Peng Li
	Samuel Palermo
Head of Department,	Chanan Singh

December 2013

Major Subject: Electrical Engineering

ABSTRACT

Artificial Neural Networks (ANNs) are a massively parallel network of a large number of interconnected neurons similar to the structure of biological neurons in the human brain. ANNs find applications in a large number of fields, from pattern classification problems in Computer Science like handwriting recognition to cancer classification problems in Biomedical Engineering.

The parallelism inherent in neural networks makes hardware a good choice to implement ANNs compared to software implementations. The ANNs implemented in this thesis have feedforward architecture and are trained using backpropagation learning algorithm. Different neural network models are trained offline using software and the prediction algorithms are implemented using Verilog and compared with the software models.

The circuit implementation of feedforward neural networks is found to be much faster than its software counterpart because of the parallel and pipelined structure as well as the presence of a large number of computations that makes the software simulations slower in comparison. The time taken from input to output by the circuit implementing the feedforward prediction algorithm is measured from the waveform diagram, and it is seen that the circuit implementation of the ANNs provides an increase of over 90% in processing speeds obtained via post-synthesis simulation compared to the software implementation.

The ANN models developed in this thesis are plant disease classification, soil clay

content classification and handwriting recognition for digits. The accuracy of the ANN model is found to be 75% to 97% for the three different problems. The results obtained from the circuit implementation show a < 1% decrease in accuracy compared with the software simulations because of the use of fixed-point representation for the real numbers. Fixed-point representation of numbers is used instead of floating-point representation for faster computational speed and better resource utilization.

To my family

ACKNOWLEDGEMENTS

First of all, I would like to thank my thesis advisor, Dr. Jiang Hu, for all his help, guidance and encouragement throughout my thesis. I would also like to thank all my committee members, Dr. Palermo, Dr. Li, and Dr. Ge, for agreeing to be on my thesis committee and for their valuable suggestions. A special thanks to Dr. Ge and his team for sharing their laboratory data with me which helped me immensely in my research.

I wish to thank my parents, my grandparents, my sister, my brother-in-law and my niece for their unconditional love and support, especially my father for being who he was that made me the person I am today. He will always be my inspiration. Thanks to my sister, Asma, for always believing in me and teaching me to reach for the stars. Thanks to my husband, Shadab, for his love and endless patience throughout the course of my graduate education.

Thank you, God, for everything you have given me throughout my life and for giving me my family that makes every living moment a precious one.

Finally, I would like to thank everyone in the Department of Electrical and Computer Engineering at Texas A&M for their support and encouragement while I pursued my Master in Science degree and for providing such a good learning environment to us students.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. INTRODUCTION

Artificial Neural Network (ANN) is a state of the art technique for different machine learning problems such as classification, image processing, etc. Unlike linear or logistic regression, ANNs can learn complex non-linear hypothesis for a large number of input features more efficiently [1]. The motivation behind neural networks was to have machines that could mimic the working of the human brain [2]. Just like the brain can learn to perform unlimited tasks from seeing things around us to solving complicated mathematical problems, ANNs can be trained using a learning algorithm to solve several problems. In this thesis, MATLAB, C and Verilog have been used to develop an ANN, train it, and apply it to new inputs to test their accuracy and generalization ability.

1.1 Motivation

ANNs are inherently parallel since neurons in each layer of the network are independent of one another. As such, each neuron performing many multiplications simultaneously requires massive amounts of parallel computation being performed. Also, in real-time applications, high speed operation can only be achieved if the neural network implementation uses a parallel architecture. ANNs can be implemented using either software or hardware.

Traditionally, ANNs were implemented in software because of the mathematically expensive computations involved. Computations such as multiplications and divisions are easier to implement in software since they are very expensive in

hardware. Software implementations also allow flexibility for structural modification as it is easier to modify the parameters in the ANN such as the number of layers, number of neurons in the hidden layer and the activation function used. But, even the fastest sequential processor cannot provide real-time response. As the ANN topology increases with more neurons and layers the speed of the software ANN starts falling rapidly. And from the applicability point of view, using one processor board to implement a single neural network is too expensive.

ANNs implemented in hardware are more efficient because of the parallel processing capabilities of hardware which increases the processing speeds. A spatial parallelism approach, for example, allows each neuron in the same layer to run simultaneously at really fast speeds. This will be shown in this thesis by comparing the speeds of prediction using both C and Verilog (See Table 3). Hardware's parallel processing capabilities can lead to a dramatic increase in the processing speeds of an ANN. For example, ANN can be used for voice recognition in a consumer application such as a smartphone. It is easier to directly implement the neural network in a circuit which provides a dedicated hardware solution compared to the expensive method of using software processors to implement just one function.

By training ANN to solve problems the way human brain learns several functions over time, we can use ANN to solve all kinds of artificial intelligence problems such as image processing, classification problems such as pattern recognition, robotics, etc. And these applications are not just limited to science and engineering, but also in fields such as financial applications such as the stock market and forecasting and prediction.

Different applications often have different requirements, especially when it comes to speed. One of the circuits implemented in this thesis is plant disease classification using reflectance spectra. The ANN is trained to look at reflectance spectra of the leaves and decide if the leaves are healthy or diseased. This circuit, for example, has a good application in the real-world. Consider a tractor fixed with a sprayer which goes around in a field and needs to find diseased plants so that they can be sprayed with pesticides to help control the disease. In such an application, first the diseased plant needs to be identified and then sprayed. An ANN has been developed in this thesis to classify plants as healthy or diseased based on reflectance spectra of leaves. Using the hardware approach to ANN can help save time as they can perform real-time processing after collecting the data to classifying the plant as healthy or diseased so that the sprayer can be controlled automatically to turn on or stay off. The ANN circuit can complete the classification in much less than a second so that the sprayer can be controlled automatically. Using software to solve this problem on the other hand requires the presence of a processor (or a computer) on board which performs the data collection and classification to control the sprayer and may not be able to complete this fast according to the speed of the tractor and the speed of operation required.

The ANNs are trained offline using MATLAB so that the training is better since there is no restriction on the mathematical computations and resolution of data, and the training is more flexible with regards to number of layers and number of neurons in the ANN architecture. After training, the feedforward algorithm is implemented in hardware using Verilog and is synthesized. The algorithm in Verilog uses a pipelined architecture

so that all the computations are performed efficiently and faster prediction speeds are obtained. The feedforward module in Verilog is implemented by dividing it into sub-modules. The base modules are the two multipliers, the two accumulators, and the activation function which will be explained in detail in Section 3.

1.2 Biological Neurons

The idea behind ANNs is to simulate neurons in the brain (See Figure 1) or rather, simulate a network of several interconnected neurons. A neuron is a biological cell that processes information, and consists of a cell body called soma, several dendrites that serve as input wires to the neuron and receive inputs/signals from other locations, and an axon that serves as an output wire through which it sends signals/messages to other neurons [3].

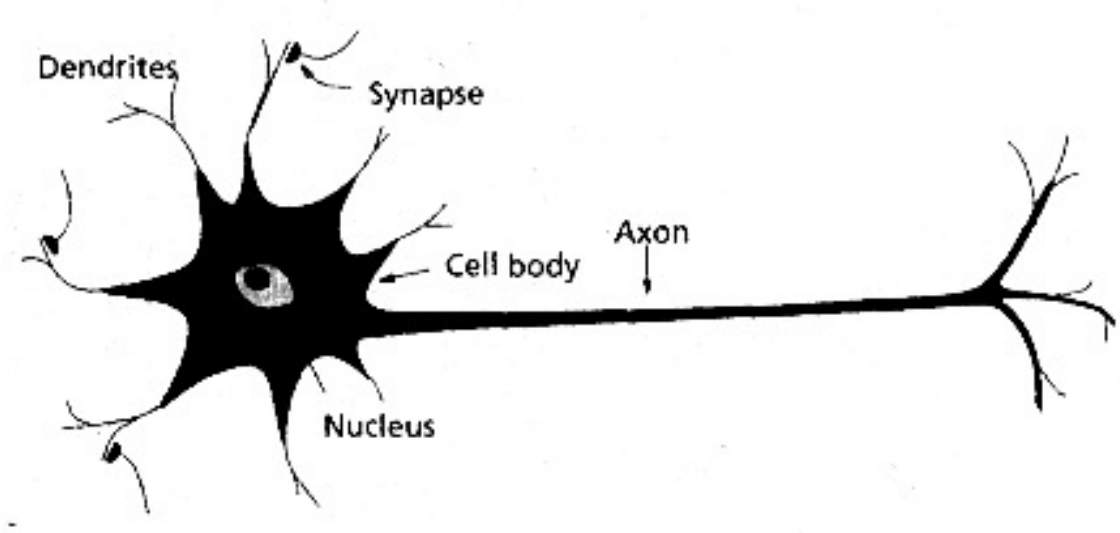

Figure 1: Neurons in the Brain

A neuron basically serves as a computational unit that receives inputs from other neurons via input wires, does computations and sends the results to other neurons or locations via axons. The data transmission to other neurons occurs when the electrical potential of the neuron exceeds a certain threshold [1]. When the threshold has not been reached, the neuron 'suppresses' the information by not activating the neuron. Similar to biological neurons, artificial neurons in an ANN consist of input wires and an output wire. The electric potential and the subsequent "firing" of the neuron when this threshold is exceeded is modeled as weighted inputs to the neurons and an activation function which is given as some non-linear function of the summation of all the weighted inputs to the neuron. This means that when the summation of the products of weights and inputs to the neuron exceeds a set threshold after passing it through the activation function, the neuron passes the output, otherwise inhibits it.

1.3 Artificial Neural Networks

ANNs can be modeled using neurons that are interconnected as shown in Figure 2 [2]. An ANN consists of an input layer where the number of neurons is equal to the number of input features, any number of hidden layers in between with an arbitrary number of neurons, and an output layer in which the number of neurons depends on the classification problem being solved. This architecture is called a feedforward ANN architecture, as is evident from the model that signals are fed forward through the network from the input layer to the output layer via the hidden layers. Figure 2 shows a feedforward ANN with three inputs shown as the three input neurons, a hidden layer

with four neurons, and two outputs of the network as shown in the two output neurons. Additionally, there exists a bias neuron at the input layer and the hidden layer which servers as an offset in the processing elements [2].

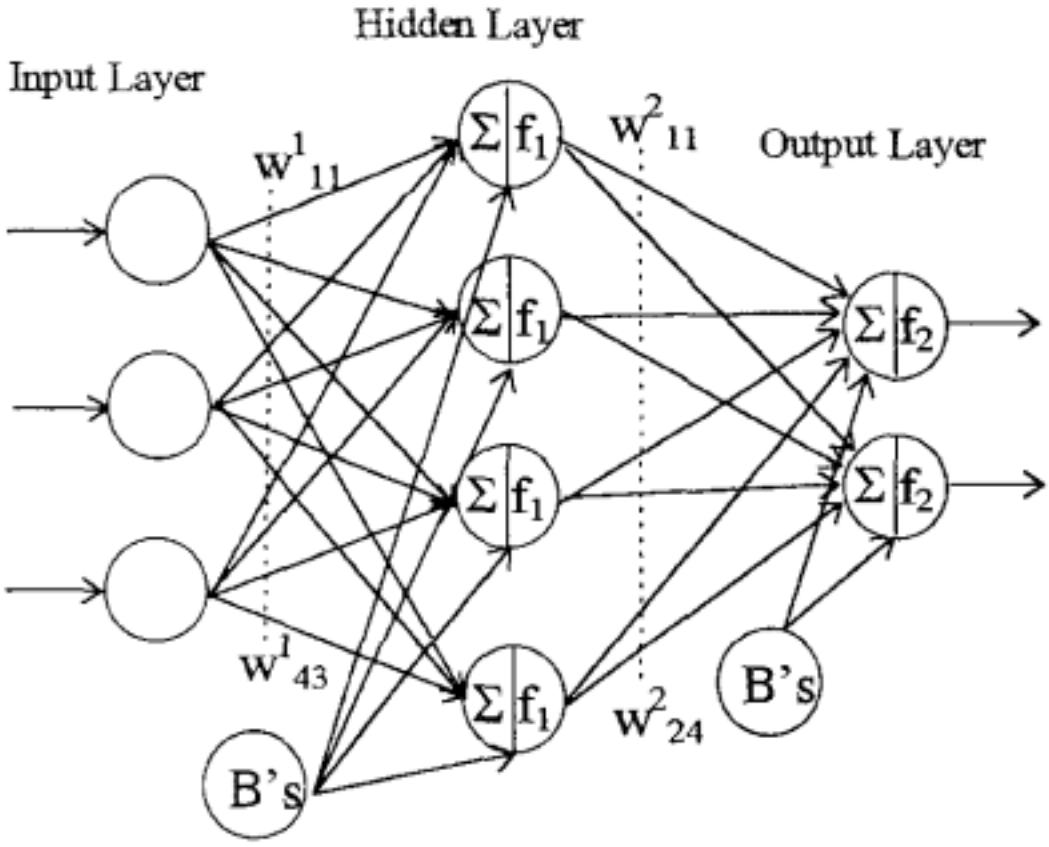

Figure 2: Feedforward Artificial Neural Network Representation (Multi Layer Perceptron (MLP) Model[1]

The connections between the different nodes in an ANN are weighted, and it is by updating these values of the weights while the ANN is being trained that the network

---

[1] Soufian, M.; Soufian, M.; Thomson, M., "Practical comparison of neural networks and conventional identification methodologies," Artificial Neural Networks, Fifth International Conference on (Conf. Publ. No. 440) , vol., no., pp.262,267, 7-9 Jul 1997.

is able to learn to perform its functions. This phase of the ANN where the network is trained is called the training phase of the network which can be both supervised and unsupervised. During this phase, training data sets that have been collected in the real world are presented to the ANN. In the case of handwriting recognition, the training data could be scanned pages with different handwritten numbers/letters on them. In supervised learning, which is what this thesis focuses on, the training set consists of input features as well as the desired output classification. The network is then trained using a technique called backpropagation explained in Section 2.4, the basic idea of which is to present the input features to the network, propagate them forward to the output via the hidden layers, take the error between this output and the desired output, and use that to update the weights for each neuron in each layer of the ANN. The above process is repeated several times until the error value reaches some desired stage at which the neural network is deemed trained.

Figure 3 shows the model of a single neuron in the ANN. The network in this case consists of 'm' inputs x_1 to x_m with x_0 being the bias value. The bias is responsible for increasing or decreasing the input of the activation function which leads to the neuron being "fired" or "inhibited". The weights associated with an input x_i is given by w^j_{ki} (where *j* represents the layer in which the neuron is located and the subscript *ki* represents the connection from neuron *i* to neuron *k*). The neuron accepts these inputs and weights, processes them by multiplying each input by its weight and performing a summation of the products. The last step is performing the activation function on this result which is the output of the neuron. The output of the neuron is given by

$$y_k = f(\sum_{i=1}^{m} w_{ki} x_i + b_k) \quad (1.1)$$

where *m* is the number of inputs to the neuron and b_k is the bias value in that layer. The bias is considered as another weight w_0 associated with a constant input $x_0 = 1$ [3]. The summation of the weighted inputs is then passed to the activation function f(.) explained in the next section.

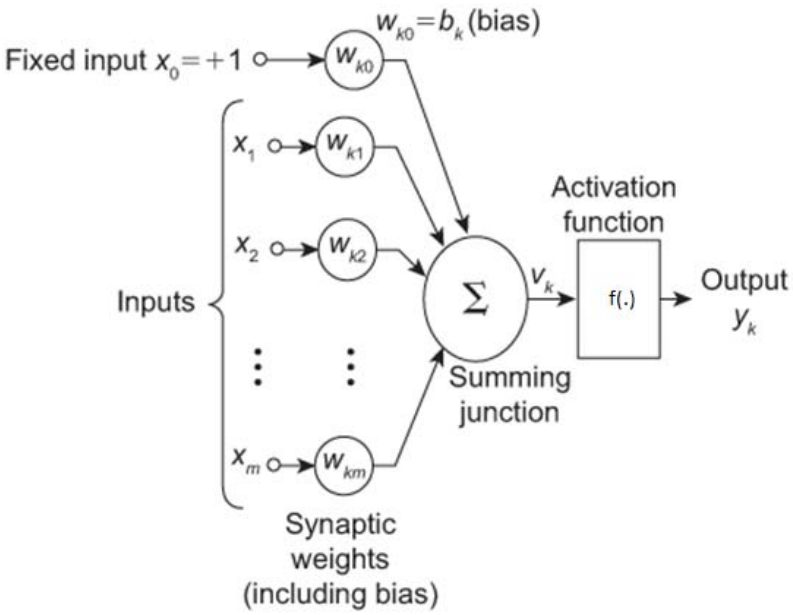

Figure 3: A Single Neuron

For the ANN shown in Figure 2, the outputs of the hidden layer are fed forward as inputs to the next layer in the network. The next layer has its own set of corresponding weights for each input and uses Equation 1.1 to find outputs for all the neurons in this hidden layer. The final output of this ANN is a set of two outputs for each of the neuron in the output layer. The number of neurons in the output layer corresponds to the number of classifications for the machine-learning problem. The

activation function is responsible to either “fire” the neuron or “inhibit” it. This function is responsible for making sure that the output of the neuron is within a certain range so that it is not unbounded. The feedforward architecture in Figure 2 is a Multi Layer Perceptron (MLP) model where the neurons have unidirectional connections from each layer to the next forward layer [3].

The final output of the ANN shown in Figure 2 is

$$y_1 = f(\sum_{j=1}^{n} w_{j1}^2 f(\sum_{i=1}^{m} w_{ij}^1 x_i + b_{1i}) + b_{2j}) \quad (1.2)$$

$$y_2 = f(\sum_{j=1}^{n} w_{j2}^2 f(\sum_{i=1}^{m} w_{ij}^1 x_i + b_{1i}) + b_{2j}) \quad (1.3)$$

In the above equation, *m* represents the number of input features and *n* represents the number of neurons in the hidden layer.

1.4 Plant Disease Classification

The ANN model developed here is to classify whether a plant is diseased or healthy. The training set is obtained from Prof. Yufeng Ge’s lab in the Biological & Agricultural Engineering department at Texas A&M. The data set consists of 43 input features that are the reflectance spectra of cotton plant leaves from wavelengths 375 nm to 2475 nm (in steps of 50 nm). The target output of the data sets consists of two labels that stand for the leaf sample being either healthy or diseased. Figure 4 shows the spectral energy of two random samples from each label taken from the training data set for the cotton plant leaves. The plot for the reflectance drawn in blue indicates a healthy leaf while the red plot indicates a diseased leaf.

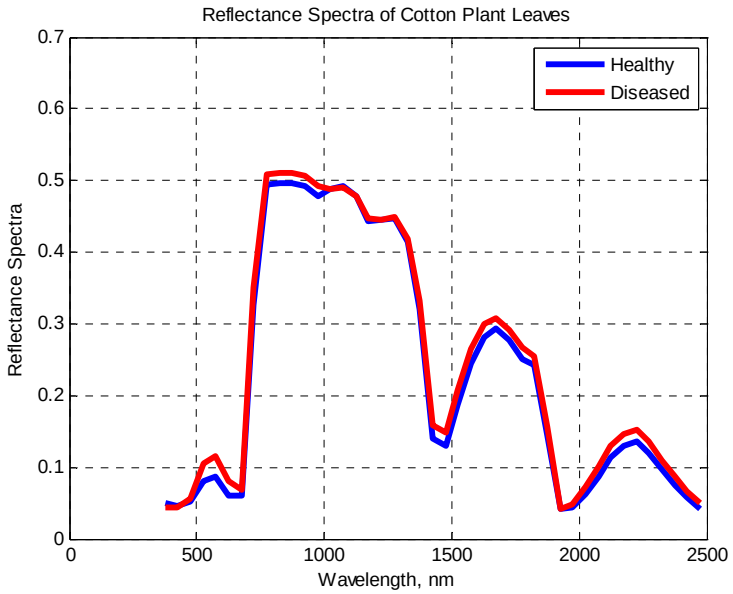

Figure 4: Spectral Energy of Cotton Plant Leaves

1.5 Soil Spectral Classification

The ANN model developed here is to classify whether the soil consists of low, medium or high clay content. The training set is also from Prof. Yufeng Ge's lab. The data set has of 270 training samples each of which consists of 216 input features that are the reflectance spectra of soil from wavelengths 350 nm to 2500 nm (in steps of 10 nm). The target output of the data sets consists of three labels that stand for the clay content of the soil. Figure 5 shows the spectral energy of three random samples from each label taken from the training data set for the soil. The plot for the reflectance drawn in blue indicates a low clay content, while the red and green plots indicate medium and high clay content respectively.

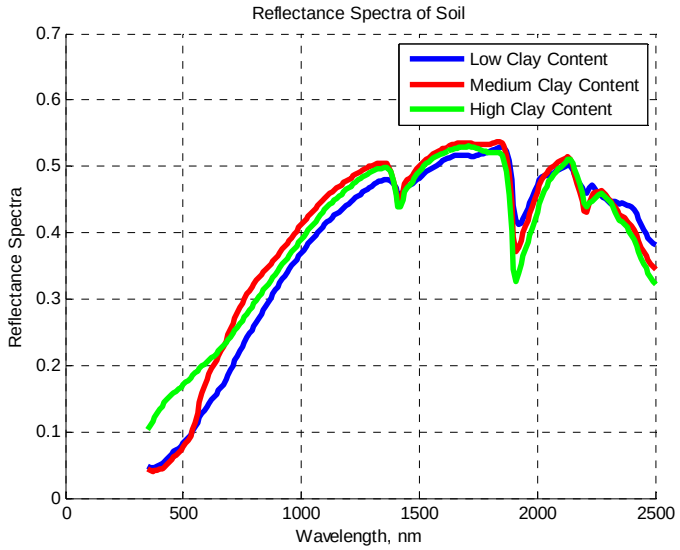

Figure 5: Spectral Energy of the Soil Samples

1.6 Handwriting Recognition for Digits

In handwriting recognition for digits, the inputs present to the neural network consist of 400 features. These features are a 20x20 pixel scanned in image of several handwritten numbers obtained from the MNIST database [4] [5]. The ANN is trained to recognize digits from 0 to 9 i.e., this is a classification problem with 10 labels. The training data consists of over 5000 samples which is a subset of over 60,000 training samples present in the MNIST database. Figure 6 shows few of the handwritten digits in the MNIST data set used for training the ANN.

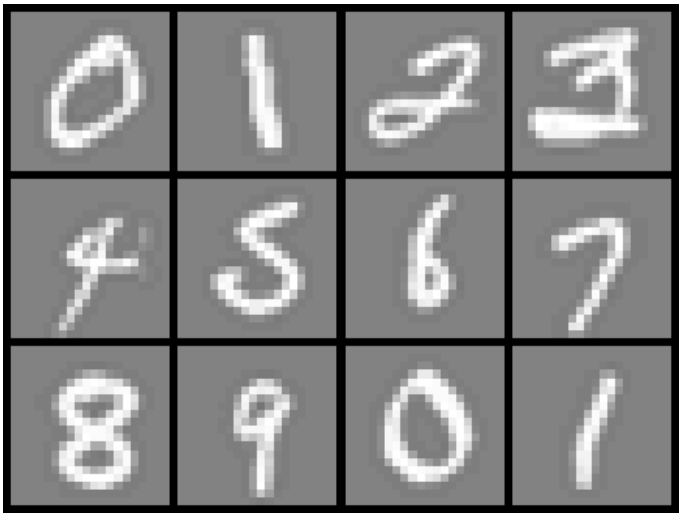

Figure 6: Handwritten Digits from the MNIST Database

In this thesis, ANN models are developed for the three models described above. The results obtained as well as the accuracy is compared with each other and with the software model. Section 2 explains the training algorithms used to train the ANN to solve the above problems. Section 3 explains the hardware implementation of the ANN and Section 4 shows the results obtained for the ANNs.

2. SOFTWARE MODEL FOR TRAINING NEURAL NETWORKS

In this thesis, supervised learning is used to train the ANN using backpropagation algorithm. Learning here implies using training data sets with known target outputs to learn the weights of the network. This means updating the weights of the ANN between the neurons and the layers over several iterations [3]. The ANN architecture used in the thesis is the Multi-layered Perceptron (MLP) model and consists of two layers. The ANN model is tested on three different sets of training data for different problems, each with varying number of input features and output classification labels, as well as different amounts of training data samples. As such, the number of neurons in the hidden layer varies for all three models. In this section, the learning algorithm as well as the prediction algorithm are described and are implemented in software using MATLAB, and the accuracy of the obtained weights and ANN is tested using statistical techniques.

Software is chosen to train the ANN because of its flexibility in changing the parameters of the ANN like the number of hidden layers, the number of neurons in the hidden layers, as well as the ease in computing mathematical functions such as the exponentials, derivatives, multiplications and divisions used in the learning algorithm [6]. Scaling the size of the ANN for the backpropagation algorithm leads to an exponential increase in the computational complexity of the model implemented in hardware. Also, implementing the learning algorithm in hardware leads to loss of efficiency if fixed-point representation of numbers is used, and is hardware expensive if

floating-point representation is used. Therefore, offline learning in MATLAB is chosen instead to train the ANN.

2.1 Pre-processing Inputs

The first step in this process consists of pre-processing the input data or transforming the raw data by normalizing it for better results. During normalization, the inputs are scaled to a range [-1,+1] which is called the pre-processing stage before propagating them through the network for learning. The technique used is the min-max normalization which distributes the input values on a whole, normalized interval chosen [7]. The equation to perform normalization is:

$$y = [y_{max} - y_{min}] \frac{x - x_{min}}{x_{max} - x_{min}} + y_{min} \quad (2.1)$$

where y is the normalized output, y_{max} and y_{min} are the maximum and minimum ranges the inputs are to be normalized to, and x_{max} and x_{min} are the maximum and minimum values of the inputs (for each training sample). This process is performed on each input belonging to the training samples data set.

2.2 Activation Function

The activation function in an ANN is responsible for "firing" or "inhibiting" the neuron. It is a non-linear function and is used to bound the output of a neuron to a specific range. The activation function used in this thesis is the continuous logistic function, also called as a sigmoid function (See Equation 2.2). The sigmoid function limits the range of the output signal to [0,1] and its derivative can be easily calculated.

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.2}$$

The derivative of the sigmoid function is:

$$f'(x) = \frac{d}{dx}\frac{e^x}{e^x+1} = \frac{(e^x+1)e^x - e^{2x}}{e^x+1} = \frac{e^x}{(e^x+1)^2} = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$f'(x) = \frac{1}{(1+e^{-x})}\frac{e^{-x}}{(1+e^{-x})} = \frac{1}{(1+e^{-x})}\frac{1-1+e^{-x}}{(1+e^{-x})}$$

$$= \frac{1}{(1+e^{-x})}\left(\frac{1+e^{-x}}{(1+e^{-x})} - \frac{1}{(1+e^{-x})}\right)$$

$$f'(x) = f(x)(1-f(x)) \tag{2.3}$$

Since the derivative of the sigmoid function is easily calculated given the input term *x* (See Figure 7), it makes it computationally efficient as an activation function since the gradient descent algorithm in the learning phase of the ANN needs to calculate the gradient of the sigmoid function as will be explain in the backpropagation algorithm in Section 2.4.

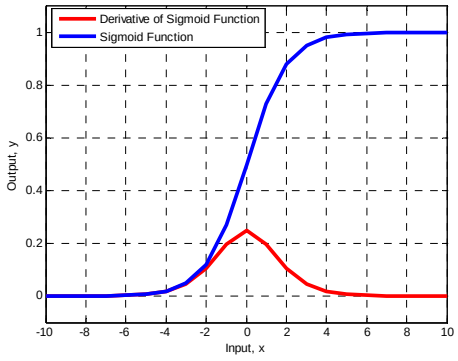

Figure 7: Sigmoid Function and its Derivative

2.3 Feedforward Prediction Algorithm

After the ANN has been trained using backpropagation, the model can be used for prediction of new and unknown inputs to solve machine learning problems. The feedforward algorithm to propagate the inputs forward to the output layer via the hidden layers is explained in this section. The prediction part is explained before the learning part since the backpropagation learning algorithm uses the feedforward algorithm to measure the output of the ANN. After training has been completed, the parameters of the ANN i.e., the connection weights are loaded into the feedforward program. The weights are represented by W^1 for the connection weights from neurons in the input layer to the neurons in the hidden layer, and by W^2 for the weight connections from neurons in the hidden layer to the neurons in the output layer. The dimensions of the connection weights are:

W^1 is a (number of neurons in hidden layer) x (number of input features + 1) dimension matrix, with the first row containing the weights from each neuron in the input layer to the first neuron in the hidden layer, and so on.

W^2 is a (number of neurons in output layer) x (number of neurons in hidden layer + 1) dimension matrix, with the first row containing the weights from each neuron in the hidden layer to the first neuron in the output layer, and so on.

In this algorithm, the dimensions and notations are given as follows:

The input consists of *m* features and is given by

$$[x_1, x_2, \ldots, x_m]$$

The output of the hidden layer consisting of *k* neurons is given by

$$[z_1, z_2, \ldots, z_k]$$

The output of the output layer consisting of *p* neurons is given by

$$[y_1, y_2, \ldots, y_p]$$

The steps in the feedforward prediction algorithm are as follows:

1. For each neuron in the hidden layer, find the summation of the weighted inputs

$$z_k = \sum_{i=1}^{m} w_{ki}^1 \; x_i + b_{k1} \tag{2.4}$$

where *k* represents the number of neurons in the hidden layer and b_{k1} represents the bias value which can be seen as the weight for the constant value input $x_0 = 1$.

2. Apply the activation function to the results obtained in step 1 to obtain the outputs of the neurons in the hidden layer.

$$z_k = \frac{1}{1+e^{-\left(\sum_{i=1}^{m} w_{ki}^1 \; x_i + b_{k1}\right)}} \tag{2.5}$$

3. For each neuron in the output layer, i.e., the number of classifications of the machine learning problem, find the summation of their weighted inputs.

$$y_p = \sum_{i=1}^{k} w_{pi}^2 \; z_i + b_{p2} \tag{2.6}$$

where p represents the number of output neurons.

4. Apply the activation function to the results obtained in step 2 to obtain the outputs of the neurons in the hidden layer.

$$y_p = \frac{1}{1+e^{-\left(\sum_{i=1}^{k} w_{pi}^2 \; z_i + b_{p2}\right)}} \tag{2.7}$$

5. The final prediction of the ANN is the neuron that contains the maximum output y_p which indicates the class that the input vector belongs to.

2.4 Backpropagation Learning Algorithm

The backpropagation learning algorithm was used to train the feedforward MLP ANN. To perform this supervised learning, the training data set is fed to the network which consists of inputs and their known associated output. The inputs from the training set are given to the system and the output of the neuron is measured by feedforward propagation using randomly initialized weights. The error between this output and the desired output is measured and the weights are updated to minimize this error at each node in each layer. This process is repeated over several iterations while the error reaches some specified threshold. The total error E is defined as follows [8]

$$E = \frac{1}{2}\sum_c \sum_p (y_p - Y) \tag{2.8}$$

where Y is the desired target output, p is the number of neurons in the output layer and c is the number of samples in the training data set. Gradient descent is used to minimize this error for which the partial derivative of E with respect to the weights in the network are needed [8]. For this reason, it is important that the activation function selected be continuously differentiable as mentioned in Section 2.2.

The steps of the algorithm are:

1. Initialize the weights of the network randomly using uniform distribution between the range given below where d_{in} represents the fan-in i.e., number of input nodes [9].

$$\left[-\frac{2.38}{\sqrt{d_{in}}}, +\frac{2.38}{\sqrt{d_{in}}}\right] \tag{2.9}$$

This is done in MATLAB using the rand function in the following way

```
A + (B-A) * rand()
```

where A is the lower range and B is the upper range. The proper initialization is important as it affects the learning speed of the ANN. If the inputs are not normalized, the algorithm might have a slow convergence rate [7].

2. For each input in the training set *x*, perform the feedforward algorithm on it (as explained in Section 2.2) to calculate the output of the ANN y_p to the initialized weights (where *p* denotes the number of classifications i.e., neurons in the output layer).

3. Calculate the error in the output layer between the obtained value and the outputs of the data set.

$$\frac{\delta E}{\delta y_p} = y_p - Y \tag{2.10}$$

where *Y* represents the target output as per the training data set for the associated input.

4. The next step is to calculate the error over the first layer of the ANN i.e., for the internal neurons. This is done by propagating the output error signal obtained back to all neurons in the internal layers [10]. The internal errors for the internal neurons in the hidden layer are given by

$$\frac{\delta E}{\delta z_k} = \sum_p \frac{\delta E}{\delta y_p} w_{pk} \tag{2.11}$$

and for the internal neurons in the input layer are given by

$$\frac{\delta E}{\delta x_m} = \sum_k \frac{\delta E}{\delta z_k} w_{km} \tag{2.12}$$

5. After all the internal deltas have been calculated, the connection weights for each neuron is updated. The connection weights from the input layer to the hidden layer are updated by:

$$w_{km} = w_{km} + \eta \frac{\delta E}{\delta y_m} f'(\sum_m x_m w_{km})\, x_m \tag{2.13}$$

The connection weights from the hidden layer to the output layer are updated by:

$$w_{pk} = w_{pk} + \eta \frac{\delta E}{\delta y_k} f'\left(\sum_k z_k w_{pk}\right) z_k \tag{2.14}$$

η is the learning rate of the algorithm. The derivative of the sigmoid function f'(.) is used to adjust the weight in the direction of the gradient to arrive at the minima of a function. This method is called the gradient descent method which is a first-order optimization algorithm.

6. The steps 3 to 5 are repeated for each data sample in the training set for a large number of iterations so that the weights converge quickly.

The number of iterations used in the program for this thesis for testing was 6000 iterations.

2.5 Performance of the ANN

Two statistical techniques are used to estimate the error rate of the ANN models trained using the backpropagation learning algorithm. Leave One Out Cross Validation (LOOCV) test consists of using one data set from the training samples to test the ANN while training the network with the rest of the samples. This method is repeated so that each data set in the training sample is used once as the test data. [11]. The other method used is Monte Carlo Cross Validation (MCCV) which randomly chooses 80% of the training data set for training and uses the remaining 20% for testing, and averages the errors obtained by repeating over 100 iterations.

The performance of the ANN for the three models trained using the backpropagation method is tabulated in Table 1. An explanation of the results is given in Section 4.

	MCCV	**LOOCV**
Handwriting Recognition for Digits	98.26 %	98.53 %
Plant Disease Classification	83.8 %	84.2 %
Soil Clay Content Classification	76.16 %	74.25 %

Table 1: Accuracy of the ANN Models using C

For the purpose of this thesis, 80% of the training data sets were randomly chosen to train the ANN. The remaining 20% of the data sets were saved to test the ANN after the model was trained. This means that the MCCV method was chosen over the LOOCV method because of simplicity of implementation while testing the hardware circuit.

3. HARDWARE IMPLEMENTATION OF NEURAL NETWORKS FOR PREDICTION

The most important part of this thesis is to implement the prediction of ANNs in hardware after the ANN has been trained offline. After training the ANN using the backpropagation algorithm and measuring the accuracy of the ANN on several test sets, the feedforward propagation algorithm is implemented in hardware using Verilog and the performance is analyzed and compared to the software results. The weights obtained from the training part are stored in a text file and loaded into a testbench for the Verilog module to read the data, and the inputs to be classified are also stored in a text file and passed to the module as well. It will be shown at the end of the section that the speed of operation of the algorithm in hardware is much faster than the speed obtained when running the program in MATLAB or C. Also, since all the values are real numbers instead of integers, they are converted to a fixed-point representation before loading them into Verilog. Fixed point is preferred over floating point because of faster computational speed and better resource utilization, though it does lead to a loss in accuracy of <1% [12].

3.1 Methods of Implementation of ANN in Hardware

In software, the feedforward algorithm basically consists of a number of multiplications and additions for each neuron, and the operation of an activation function on the accumulated products of the inputs. Consider the ANN model for the plant

disease classification problem posed in Section 1.3. The ANN model for this problem is shown in Table 2. The model consists of 43 input neurons, 8 neurons in the hidden layer and 2 output neurons.

	Handwriting Recognition for Digits	**Plant Disease Classification**	**Soil Clay Content Classification**
Number of Classifications	10	3	2
Number of Training Samples	5000 (500 for each digit)	702 (176 for healthy label, 526 for diseased label)	270 (63 for low, 101 for medium, 106 for high clay content)
Number of Input Neurons	400	43	216
Number of Hidden Neurons	25	8	8
Number of Output Neurons	10	3	2
Number of Clock Cycles from Input to Output	729	64	252

Table 2: Comparison of ANN Models for Handwriting Recognition, Plant Disease and Soil Clay Content Classification

Directly translating the software code to hardware code would therefore require 44 multiplications and 43 additions for each neuron in the hidden layer and 9 multiplications and 8 additions for each neuron in the output layer which is a total of 379 multiplications and 368 additions at the very least. It comes down to needing one multiplier for each neuron input in the model and the cost increases significantly when the number of input features and the number of hidden neurons increase. In this type of

implementation, all the neurons in the first layer function simultaneously and therefore, this is the fastest implementation of an ANN.

To avoid the huge number of multipliers and therefore, the resources needed, an alternative architecture is developed for the hardware implementation of the feedforward algorithm. The hardware architecture for ANN feedforward consists of developing a neuron model for executing each layer sequentially. Each neuron in the hidden layer computes the product of the inputs at the positive edge of every clock cycle. This method reduces the number of multipliers but increases the number of clock cycles required as the time taken for each layer to finish processing is equal to the number of inputs. Therefore, for the plant disease classification problem, since the hidden layer processes 43 input features and 1 bias input value, it takes 44 clock cycles to complete the multiplications and two more clock cycles to compute the final sum of the products and the activation function output. The outputs of these then are processed as inputs to the next output layer. This method of hardware implementation requires 8 multipliers for the hidden layer and 2 multipliers for the output layer for the plant disease classification problem which is a huge decrease of 97% from the previous method. The trade-off in this method is the decreased speed from input to output. Section 3.3 describes the circuit implementation of this selected method.

3.2 Fixed-point Representation of the Numbers

As mentioned earlier, fixed-point representation of numbers is chosen over floating-point so that area is more efficiently utilized and speed is also faster at the

expense of a slight loss in accuracy [12]. 15-bit to 17-bit numbers are used for the inputs and the weights after looking at the range of numbers in the training set for the inputs and the range of the weights for the two layers obtained from training the ANN in MATLAB. A uniform scaling factor for the inputs and the two sets of connection weights is used to provide an accuracy of up to 10^{-4} decimal points. To undo the scaling, a power of 2 is chosen so that right shift operation can be performed instead of division which is a hardware intensive operation.

3.3 Neuron Implementation

The circuit implementation of the ANN feedforward algorithm consists of making use of the parallel processing capabilities of hardware by performing the computations of all the neurons in each layer in parallel. The ANN hierarchy is shown in Figure 8. The testbench is responsible for loading the weights and the inputs into the module main.v and thus start the execution of the program.

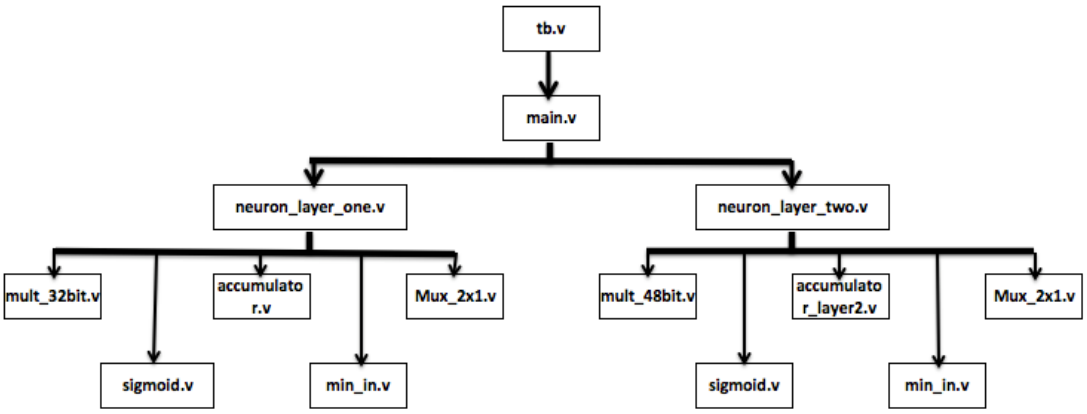

Figure 8: ANN Module Hierarchy

The main.v module is the starting point of the algorithm. The module has input ports which reads the weights for each neuron in the hidden layer and the output layer as well as the values of the input data which is to be classified. The main program passes the predicted value back to the testbench after the processing has been performed. The main.v program instantiates modules for each neuron in the hidden layer and for each neuron in the output layer. Since the number of neurons in the hidden and output layer are different, two different modules called neuron_layer_one.v and neuron_layer_two.v have been written.

3.3.1 main.v

The main.v program is the control center of the ANN module and is responsible for activating the two layers in the model. Initially, it sets the trigger to enable the first layer while keeping the second layer inactive. After the processing in the first layer has been completed, it inactivates the first layer and triggers the operation of the second layer. Since the inputs to the second layer neurons are the outputs of the first layer neurons, the main.v program transfers the neuron_layer_1 results to the input ports of the second layer neurons using a multiplexer one at a time. After the processing at the hidden layer has been completed and the outputs of the neurons in the hidden layer have been calculated, it stores these outputs in registers (one for each neuron) using a multiplexer to send the results to the next layer, starts the trigger so that the next layer can start processing, and sends the first layer outputs and the corresponding weights to the neuron_layer_two.v module.

After the second layer has finished processing, the module then finds out the final output of the ANN i.e., the classification to which the inputs belong to and sends the predicted output back to the testbench to be displayed to the user.

3.3.2 neuron_layer_one.v and neuron_layer_two.v

The neuron_layer_one.v and neuron_layer_two.v modules are written for neurons in the hidden layer and output layer respectively. The modules are instantiated in the main.v program and are passed an input and its corresponding weight every clock cycle. During the processing of the hidden layers, all the neurons in the hidden layer receive inputs and corresponding weights one at a time and perform multiplication operation on them, storing the result in a register so that they can be accumulated. Figure 9 shows the architectural structure of a single neuron. This is a fully parallel implementation of the ANN model. The number of clock cycles it takes to perform the operations of the hidden layer depends on the number of input features in the ANN model. The main.v module sends inputs and weights to the hidden layer neurons at each positive edge of the clock cycle.

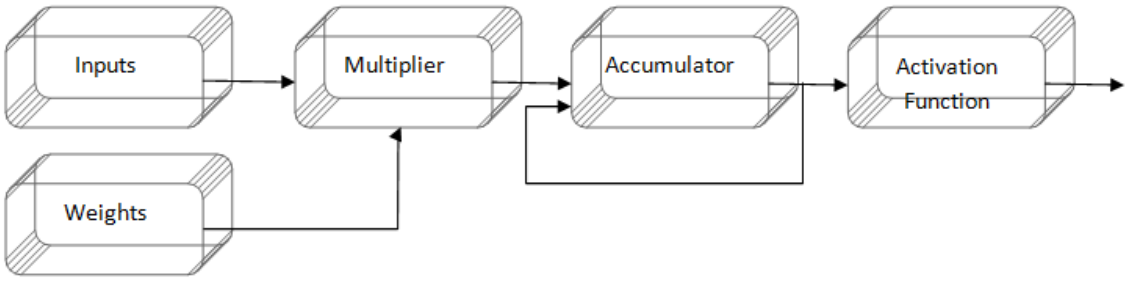

Figure 9: Structure of a Neuron

This module uses the multiplier module mult_32bit.v (for layer one) and mult_48bit.v (for layer two) to find the product of the input and the weight. This result is then passed to the accumulator module which keeps a running sum of this result. This resulting cumulative sum is right shifted to take care of the fixed-point scaling and is passed to the sigmoid module. The sigmoid.v module passes the sigmoid output back to the neuron_layer_one.v and neuron_layer_two.v modules which is the final output of the neuron. Figure 10 shows the flowchart for the operations performed by the neurons for both the layers.

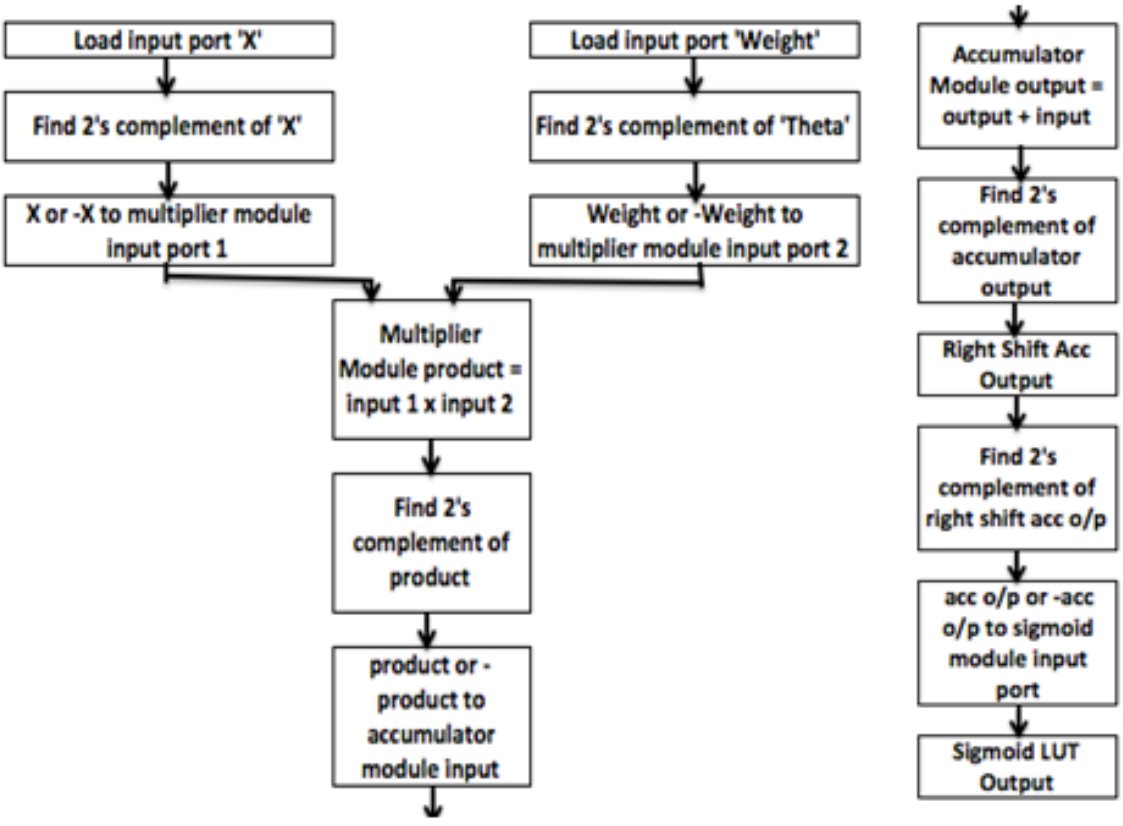

Figure 10: Flowchart of the Neuron Operations

3.3.3 mult_32bit.v and mult_48bit.v

These multiplier modules perform simple multiplication of the two inputs and store the result in the output reg. This operation is carried out when the trigger to start the multiplication is received from the neuron_layer_one.v and neuron_layer_two.v modules.

3.3.4 accumulator.v and accumulator_layer2.v

These accumulator modules keep an ongoing cumulative sum of the product of the inputs and the weights for the neurons. This operation is carried out when the trigger to start the addition is received from the neuron_layer_one.v and neuron_layer_two.v modules.

3.3.5 sigmoid.v

The activation function used in the thesis is the sigmoid function given by

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.1}$$

A lookup table implementation is used to implement the sigmoid function in Verilog. A LUT is written where the inputs are divided into 201 different points uniformly. To obtain more precision, a larger LUT can be implemented for more points of the input values, but this leads to an exponential increase in the area of the synthesized circuit being used for a LUT [13]. This operation is carried out when the trigger to start the sigmoid LUT is received from the neuron_layer_one.v and

neuron_layer_two.v modules. The modules send this trigger when the cumulative sum operation has been finished.

The total processing time of the algorithm is equal to the sum of number of neurons in the input layer (+1 for the bias value), the number of neurons in the hidden layer (+1 for the bias value) and four clock cycles, two for the accumulator for each layer and two for the LUT for the activation function for each layer.

Table 2 lists the comparison of the ANN architectures for the handwriting recognition for digits, plant disease classification and soil clay content classification. Of all the three ANN architectures, handwriting recognition had a good training set that consisted of 500 training samples for each digit that enabled it to be a well trained ANN. The ANN model used consisted of 25 neurons giving it an accuracy of 97%, but testing it with 8 neurons thereby reducing the size of the network by three times leads to a slight decrease in accuracy by 2% while increasing the number of hidden nodes to 50 gave it an accuracy of 99% as tested in C.

On the other hand, the training set for the soil clay content classification consisted of lesser samples. A smaller neural network was used in this architecture as using more than 8 hidden nodes led to overfitting. Trial and error was used when testing the backpropagation algorithm in MATLAB to determine the best architecture for this model. The accuracy obtained in this case was 75% as shown in Table 2.

The results for the plant disease classification ANN model are also shown in Table 2. The accuracy of this model was in between the accuracy of the other two models since the training data available for this model was more than the data set for the

previous problem, but was not evenly distributed between the healthy and diseased labels [14]. Balanced data sets are needed for ANNs to work well since the overall classification accuracy is being optimized [14]. If a training set is imbalanced, then the algorithm tens to be biased towards the class which contains the majority of the samples among the training set and accurate decision boundaries between the different classes is not well constructed [14]. The training data set consists of more samples for the diseased classification when compared to the healthy classification which leads to the ANN being trained in a bias way towards the majority classification data set. This is explained in more detail in Section 4.1.

4. VERIFICATION AND SYNTHESIS

4.1 Pre-synthesis Simulation

Before synthesizing the circuit, functional simulation was performed using a testbench using Synopsys VCS Compiler. The connection weights obtained from training the ANN models using MATLAB were saved in text files and loaded into the Verilog module using the testbench. 20% of the training data sets saved while training the ANN model were loaded into the testbench via text files. The results obtained were compared with the MATLAB and the C results. Using fixed-point representation of numbers and using the right-shift operators for the two layers instead of division operators caused a loss in accuracy for over < 1% of the testing data set. The results and comparison of the three models are tabulated in Table 3.

	C Accuracy	Verilog Accuracy
Plant Disease Classification	85.61%	85.04%
Handwriting Recognition for Digits	97.46%	97.10%
Soil Clay Content Classification	75.00%	73.00%

Table 3: Comparison of Accuracy of Software and Hardware ANN Models

4.2 Synthesis

The ANN models were synthesized using Synopsys Design Compiler. The symbol library used was generic.sdb and the target and link libraries used were osu018_stdcells.db. The testbench used for simulation was edited to add the .sdf delay file generated during the synthesis procedure and gate-level simulation of the

synthesized netlist was performed. The results obtained from synthesis procedure were verified with the results obtained from functional simulation and are shown in section 4.4. The results from the post-synthesis simulation were similar to the results from the pre-synthesis simulation, thus, verifying the correctness and completeness of the written Verilog code. The maximum clock frequency of the circuits was found to be 166.7 MHz for the ANN models for soil clay content classification, 166.7 MHz for plant disease classification, and 83.4 MHz for the ANN for the handwriting recognition for digits. Table 4 lists the comparison of the summarized synthesis report for the three ANN models.

	Handwriting Recognition for Digits	**Plant Disease Classification**	**Soil Clay Content Classification**
Maximum Clock Period	12 ns	6 ns	6 ns
Maximum Clock Frequency	83.4 MHz	166.7 MHz	166.7 MHz
Number of Clock Cycles from Input to Output	729	64	252
Time taken from Input to Output (Prediction Time)	8.7 us	0.384 us	1.512 us
Critical Path Length	11.19 ns	5.82 ns	5.82 ns
Total Cell Area	4300972	1307949	1458403
Combinational	3717852	1134733	1267091
Non-combinational	583120	173216	191312
Total Dynamic Power	185.4229 mW	2.4481 uW	96.5632 mW
Total Number of Gates	3924	1312	1319
Total Number of Transistors	202554	54498	60767
Total Number of Flip-flops	955	477	477
Total Number of Latches	80	80	80

Table 4: Synthesis Comparison of the ANN Models

It is seen from the summarized synthesis report in Table 4 that the ANN circuit for plant disease classification is smallest in terms of area, power consumption, number of gates and number of transistors because of the ANN model which consist of 43 input neurons, 8 hidden layer neurons and 2 output layer neurons (See Table 2). On the other hand, the ANN model for handwriting recognition of numbers consists of 400 input neurons, 25 hidden layer neurons and 10 output layer neurons and is the largest circuit. This also causes an increase in the critical path length since the number of hidden layer neuron instantiations and the number of multiplications needed for the inputs and the weights is more than thrice the number of instantiations for the plant disease and soil clay content classification models, leading to a slower circuit in terms of maximum clock frequency and time taken for prediction.

There are also 80 latches present in each circuit in addition to flip-flops as shown in Table 4. The latches are inferred in the sigmoid.v module as a result of how the module has been coded. They do not cause any undesirable effects in the synthesized circuit and this has been verified by comparing the post-synthesis and the pre-synthesis simulation results.

In order to compare the execution time of software and hardware, the clock() command is used in the C program to find out the execution time of the C program from the loading of inputs and weights into the ANN model to the display of the predicted output on the console. To find the similar time for the post-synthesized netlist of the ANN circuits, the number of clock cycles and the time taken are obtained from the waveform diagrams generated by the gate-level simulation. The waveform diagrams

obtained are shown in Section 4.4. Table 5 lists the comparison results for the three ANN models for the software and hardware execution times.

	C Program Run Time, seconds	Matlab Program Run Time, seconds	Verilog Post-synthesis Simulation Time, microseconds
Plant Disease Classification	0.001322	0.037678	0.384 us
Handwriting Recognition for Digits	0.005603	0.113470	8.700 us
Soil Clay Content Classification	0.001322	0.037678	1.512 us

Table 5: Comparison of Run Time of Software and Hardware ANN Models

In order to further study the relationship between the training data available and the accuracy of the ANN models, the accuracy of each label in the classification problems are studied as shown in Table 6.

	Soil Clay Content Classification	Plant Disease Classification	Handwriting Recognition for Digits
Number of Training Samples	270	702	5000
Number of Classifications	3	2	10
Overall Accuracy	73%	85%	97%
Number of Training Samples per Classification and Accuracy	1 63 => 68% 2 101 => 73% 3 106 81%	1 176 => 56% 2 526 => 94%	0 500 99.2% 1 500 98% 2 500 97.4% 3 500 96% 4 500 97% 5 500 95.2% 6 500 98.2% 7 500 95.6% 8 500 98.2% 9 500 96%

Table 6: Comparison of Accuracy of Each Label in the ANN Models

It is seen that the training database for the handwriting recognition problem is well distributed between the different labels and has a large sample with which to train the ANN. As such, the results show a very good accuracy for all the labels. On the other hand, an interesting relationship is seen between the different classifications for the plant disease classification problem. The ANN model shows an accuracy of 94% for the 'diseased' classification but a very low accuracy of 56% for the 'healthy' classification. As mentioned earlier, this is due to an imbalance in the training samples for the two classifications [14]. Since there are 526 training samples for the diseased classification and only 176 for the healthy classification, the ANN is biased towards the diseased class and is well trained for this class where as poorly trained to recognize the other class.

And lastly, the accuracy of the trained ANN is mostly uniform across the three labels for the soil clay content classification since the training data set is pretty unbiased over the three classifications.

Therefore, with a good training data set, the ANN models can be trained and implemented to provide a good deal of accuracy for various classification problems in different fields. The Verilog program is written in a way that makes it easy to change the parameters of the ANN such as the number of hidden layers, and the number of neurons in the input layer, the hidden layer, and the output layer.

4.3 Static Timing Analysis

After verifying the logical functionality of the circuit, the most important step is performing Static Timing Analysis (STA) on the design to measure the speed of the circuit [15]. A hardware circuit has various delays associated with it such as component delays for the gates and wires within the circuit [16]. While gates have inertial and propagation delays, presence of wires can lead to delays because of the parasitic capacitances and resistances of the wire. STA techniques are used to verify that the synthesized netlist will function correctly at the clock speed desired. STA is used to validate the timing of the netlist by checking all paths in the circuit for timing violations under the worst-care conditions [15]. The timing report is obtained from Synopsys Design Compiler to check the critical path length in the circuit and the report gives the data arrival time at this path, the required time as well as the slack present at that point.

Figure 11 shows the timing analysis performed by the Synopsys Design Compiler for the Soil Clay Content Classification ANN model and shows the critical path of the circuit. The critical path of the circuit starts at neuron_lauer_1_mux_out_reg [8] (rising-edge triggered flip-flop clocked by clk) and the endpoint of this path is at out_reg[44] in the mult_layer_2 module of the neuron_layer_two_01 module. It can be seen that the required time for the critical path is 5.82 ns which is the latest allowable arrival time for the signal at the path endpoint [15]. The data arrival time is seen to be 5.82 ns which is the amount of time it takes for the signal to travel from the source to the endpoint. The slack value (data required time – data arrival time) is zero which means

that the data arrives just in time to meet the requirements. The critical path length of all ANN models is shown in Table 4.

```
Point                                                                   Incr       Path
---------------------------------------------------------------------------------------
clock clk (rise edge)                                                   0.00       0.00
clock network delay (ideal)                                             0.00       0.00
neuron_layer_1_mux_out_reg[8]/CLK (DFFPOSX1)                            0.00 #     0.00 r
neuron_layer_1_mux_out_reg[8]/Q (DFFPOSX1)                              0.41       0.41 f
U12369/Y (INVX4)                                                        0.19       0.60 r
U13194/Y (INVX4)                                                        0.15       0.75 f
neuron_layer_two_01/mult2_layer_2/mult_30/U870/Y (XNOR2X1)              0.14       0.89 f
neuron_layer_two_01/mult2_layer_2/mult_30/U843/Y (OAI22X1)              0.18       1.06 r
neuron_layer_two_01/mult2_layer_2/mult_30/U546/YS (FAX1)                0.34       1.40 r
neuron_layer_two_01/mult2_layer_2/mult_30/U544/YC (FAX1)                0.23       1.63 r
neuron_layer_two_01/mult2_layer_2/mult_30/U536/YS (FAX1)                0.29       1.93 r
neuron_layer_two_01/mult2_layer_2/mult_30/U535/YS (FAX1)                0.25       2.18 r
neuron_layer_two_01/mult2_layer_2/mult_30/U273/Y (NOR2X1)               0.12       2.30 f
neuron_layer_two_01/mult2_layer_2/mult_30/U269/Y (NOR2X1)               0.12       2.41 r
U7347/Y (OAI21X1)                                                       0.06       2.48 f
U7977/Y (OAI21X1)                                                       0.10       2.58 r
U9591/Y (AOI21X1)                                                       0.11       2.69 f
neuron_layer_two_01/mult2_layer_2/mult_30/U198/Y (OAI21X1)              0.15       2.84 r
neuron_layer_two_01/mult2_layer_2/mult_30/U190/Y (AOI21X1)              0.13       2.97 f
neuron_layer_two_01/mult2_layer_2/mult_30/U184/Y (OAI21X1)              0.15       3.13 r
neuron_layer_two_01/mult2_layer_2/mult_30/U176/Y (AOI21X1)              0.13       3.25 f
neuron_layer_two_01/mult2_layer_2/mult_30/U170/Y (OAI21X1)              0.15       3.41 r
neuron_layer_two_01/mult2_layer_2/mult_30/U162/Y (AOI21X1)              0.13       3.54 f
neuron_layer_two_01/mult2_layer_2/mult_30/U156/Y (OAI21X1)              0.15       3.69 r
neuron_layer_two_01/mult2_layer_2/mult_30/U148/Y (AOI21X1)              0.13       3.82 f
neuron_layer_two_01/mult2_layer_2/mult_30/U142/Y (OAI21X1)              0.15       3.97 r
neuron_layer_two_01/mult2_layer_2/mult_30/U134/Y (AOI21X1)              0.13       4.10 f
neuron_layer_two_01/mult2_layer_2/mult_30/U128/Y (OAI21X1)              0.15       4.26 r
neuron_layer_two_01/mult2_layer_2/mult_30/U105/Y (AOI21X1)              0.13       4.39 f
neuron_layer_two_01/mult2_layer_2/mult_30/U99/Y (OAI21X1)               0.15       4.54 r
neuron_layer_two_01/mult2_layer_2/mult_30/U91/Y (AOI21X1)               0.09       4.63 f
U13969/Y (INVX2)                                                        0.07       4.70 r
neuron_layer_two_01/mult2_layer_2/mult_30/U88/YC (FAX1)                 0.18       4.89 r
neuron_layer_two_01/mult2_layer_2/mult_30/U87/YC (FAX1)                 0.19       5.08 r
neuron_layer_two_01/mult2_layer_2/mult_30/U86/YC (FAX1)                 0.19       5.28 r
neuron_layer_two_01/mult2_layer_2/mult_30/U85/YC (FAX1)                 0.19       5.47 r
neuron_layer_two_01/mult2_layer_2/mult_30/U84/YC (FAX1)                 0.17       5.64 r
U14260/Y (XOR2X1)                                                       0.10       5.75 f
U16972/Y (OAI21X1)                                                      0.07       5.82 r
neuron_layer_two_01/mult2_layer_2/out_reg[44]/D (DFFPOSX1)              0.00       5.82 r

data arrival time                                                                  5.82

clock clk (rise edge)                                                   6.00       6.00
clock network delay (ideal)                                             0.00       6.00
neuron_layer_two_01/mult2_layer_2/out_reg[44]/CLK (DFFPOSX1)            0.00       6.00 r
library setup time                                                     -0.18       5.82
data required time                                                                 5.82
---------------------------------------------------------------------------------------
data required time                                                                 5.82
data arrival time                                                                 -5.82
---------------------------------------------------------------------------------------
slack (MET)                                                                        0.00
```

Figure 11: Critical Path of the Soil Clay Content Classification ANN

4.4 Post-synthesis Simulation

The gate-level timing simulation of the three ANN models is performed using a clock period shown in Table 4. The circuit implementation of the three models is similar with the difference being in the number of instantiations of the hidden layer neurons and output layer neurons. Figure 12 shows the timing diagram for the gate-level timing simulation of the character recognition ANN running with a clock period 12 ns. The inputs and the weight parameters are read by the testbench and passed to the main module and the resulting output prediction is read back by the testbench from the main module as 7 which predicts the digit 7 for the input read by the program. The answer is accurate and the timing diagram shows the 3 of the 25 neurons in the hidden layer and 3 of the 10 neurons in the output layer performing their work in a parallel manner.

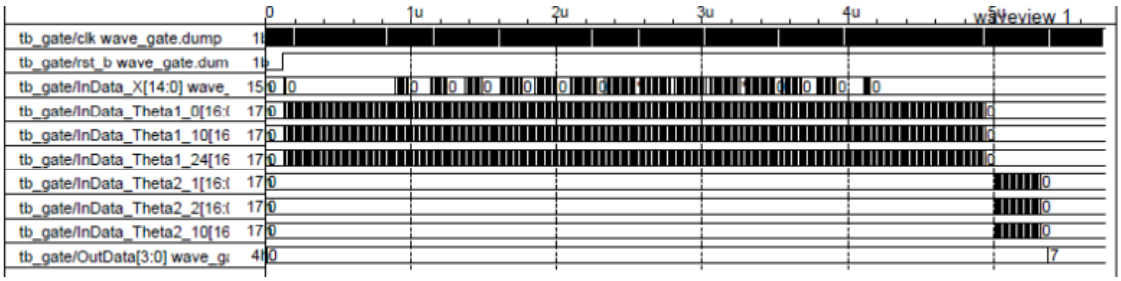

Figure 12: Timing Diagram: Inputs and Outputs in the Testbench

Figure 13 shows the work being performed in the main.v module where three of the 25 hidden layer neurons and 3 of the 10 output layer neurons are shown. All the hidden layer neurons read the inputs and their connection weights simultaneously when the trigger to start the first layer is provided. After the first layer has finished processing and compute their outputs, the main module provides the trigger to start the second layer and the output layer neurons start reading the inputs which are the outputs of the

previous layer as well as the connection weights and finish their processing. After all the layers finish their computations, the main program starts its prediction by firing the result_done trigger which predicts the output classification based on the outputs of the output layer neurons.

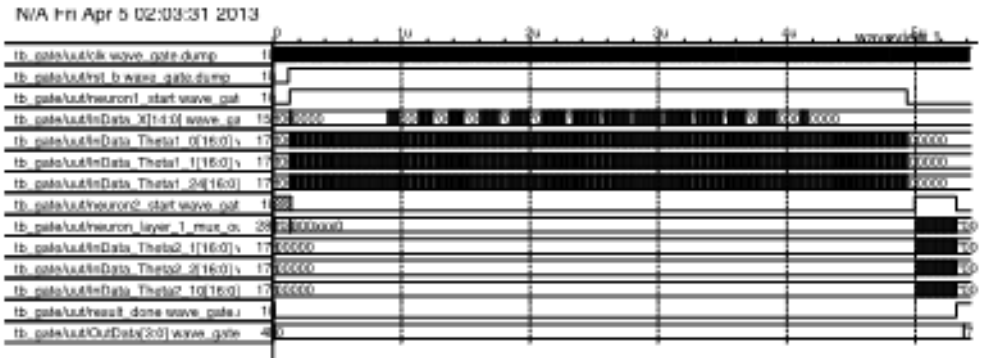

Figure 13: Timing Diagram: Neuron Module Instantiations

Figure 14 shows the timing diagram of the internal workings of the neurons in the hidden and the output layer. The timing diagram only shows one of the neurons in both the layers due to lack of space. The hidden layer starts computations when it receives the neuron1_start trigger at which point it starts reading its inputs for each neuron. The module then starts multiplying the inputs with their associated connection weight and keeps a running cumulative sum of the product. At the completion of this process where all 400 inputs have been read and processed, the answer is sent to the LUT for the sigmoid activation function. The neuron1_start trigger is now set to LOW and the results calculated now serve as inputs to the next layer. The neuron2_start signal is triggered and a multiplexer is used to send one of the inputs of the hidden layer to the output layer nodes, one at each positive edge of the clock cycle. The second layer

neurons perform the computations in the same way and finally all 10 outputs at the output layer are calculated over a period of 26 clock cycles (one for each of the 25 neuron inputs from the hidden layer and +1 for the final cumulative sum calculation).

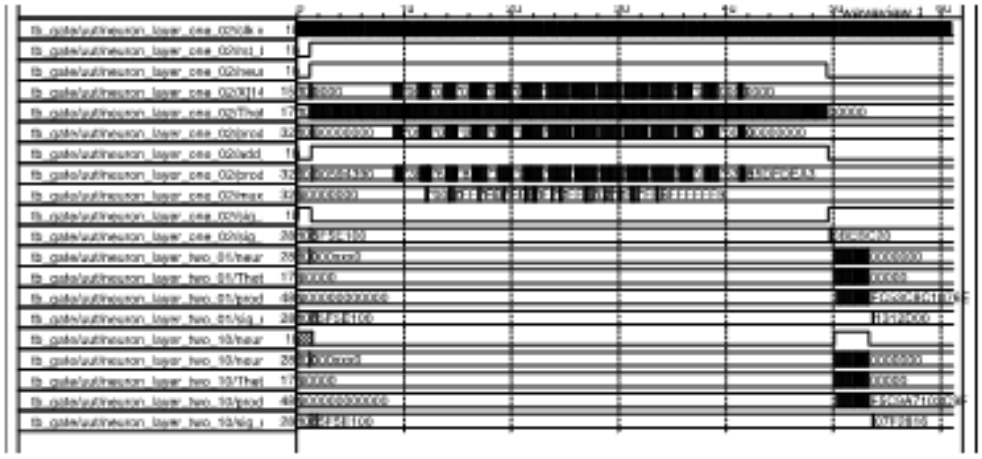

Figure 14: Timing Diagram: Neuron Module Processing

The result obtained on running vcs tb_gate.v and simv commands is shown below:

> SUCCESS : X file was read successfully.

> SUCCESS : Theta2 file was read successfully.

> SUCCESS : Theta1 file was read successfully.

OutData is 7

The accuracy of the hardware implementation of the Verilog model was found to be 1% less than the software implementation since fixed-point representation was used instead of floating point representation. This led to some loss in precision, but using

fixed-point representation saved computational efficiency as well as the design, and this serves as a trade-off between the two number representations.

4.5 Generalization Ability of the ANNs

The generalization ability of the ANN models is tested in this section. Generalization is the ability of the ANN to classify inputs that are not in the training data sets i.e., input data that has not been presented to the ANN yet and is unknown [17]. An ANN that is able to generalize well needs to be trained very well so it is able to recognize the new input features presented to it. The data sets presented to the ANN for training should be large and representative of all the classifications in the ANN architecture so that the ANN has the ability to be well generalized.

To test the generalization ability of the ANN, 80% of the data sets were used to train the ANN and the remaining 20% were saved to test the ANN. Table 1 had tabulated the results from C obtained on using statistical tests LOOCV and MCCV (for 100 iterations). Table 7 lists the accuracy obtained by the ANN models on the remaining 20% of the test data set for all three ANN models for 5 different iterations. Each iteration uses different randomly permuted data sets for training and testing.

	C Accuracy		Verilog Accuracy (Post-synthesis Simulation)	
Handwriting Recognition for Digits	1	94.7%	1	94.3%
	2	92.0%	2	89.7%
	3	92.4%	3	90.4%
	4	94.3%	4	94.2%
	5	93.5%	5	93.2%
Plant Disease Classification	1	87.0%	1	86.4%
	2	84.0%	2	82.4%
	3	85.3%	3	84.3%
	4	88.6%	4	85.6%
	5	82.9%	5	82.4%
Soil Clay Content Classification	1	74.4%	1	72.1%
	2	74.0%	2	67.8%
	3	73.1%	3	70.0%
	4	73.4%	4	69.8%
	5	70.2%	5	67.8%

Table 7: Comparison of the Generalization Ability of the ANN Models

For the very first iteration in the three ANN models in Table 7, the training data was sorted and 80% of data from each classification was chosen randomly. For example, in the plant disease classification problem, the data samples used consisted of 702 samples, 176 of which represented the healthy classification and 526 represented the diseased classification (See Table 6). To train the ANN, 141 samples from the healthy classification and 420 samples from the diseased classification were chosen. After the ANN was trained, the remaining 20% samples i.e., 141 samples were used to test the trained ANN. This was done since the training data set is imbalanced which leads to a very poorly trained and generalized ANN. In the case of soil clay content classification, 50 of the low clay content samples, 81 of the medium clay content samples and 85 of the high clay content samples were used for training the ANN (80%). The remaining 20%

samples i.e., 54 samples of the total 270 samples in the data set were used for testing the ANN.

Table 8 shows the accuracy obtained for different ratio of training and testing samples used in the three ANN models. It is seen that the accuracy starts dropping off when the number of samples used to train the ANNs starts decreasing. The accuracies for the plant disease classification ANN and the soil clay content classificaion ANN drop off sharply as the number of samples in the training data set is less compared to the ANN for handwriting recognition for digits which contains over 500 samples for each classification.

	Training Samples : Testing Samples Ratio	**C Accuracy**	**Verilog Accuracy (Post-synthesis Simulation)**
Handwriting Recognition for Digits	90% : 10%	93.8 %	93.6 %
	80% : 20%	90.6 %	89.9 %
	70% : 30%	89.2 %	86.0 %
	60% : 40%	85.7 %	82.3 %
	50% : 50%	82.8 %	80.9 %
Plant Disease Classification	90% : 10%	84.3 %	82.9 %
	80% : 20%	81.4 %	78.6 %
	70% : 30%	78.8 %	73.2 %
	60% : 40%	73.6 %	68.2 %
	50% : 50%	65.7 %	62.9 %
Soil Clay Content Classification	90% : 10%	75.9 %	73.2 %
	80% : 20%	71.7 %	66.6 %
	70% : 30%	65.9 %	63.2 %
	60% : 40%	52.7 %	49.1 %
	50% : 50%	55.5 %	52.8 %

Table 8: Accuracy Obtained by using Different Training:Testing Ratio

4.6 The ANN Circuit and Real World Applications

The ANN circuit synthesized can be used in the real world in a few different ways. One method which requires time and carries a high cost is to use the conventional VLSI chip. A VLSI implementation can achieve fast processing speeds in real-time applications, but a disadvantage is that they are not flexible when it comes to the ANN topology. The Verilog code for the ANN can be synthesized for a Field Programmable Gate Array (FPGA) or an Application Specific Integrated Circuit (ASIC). The synthesized netlist can be mapped onto an FPGA or an ASIC. Using FPGAs to implement the neural network circuit allows the circuit to be flexible when it comes to the number of neurons and number of layers in the system. Not just that, an FPGA implementation is faster than the traditional VLSI design and also has a smaller size [18]. The FPGA implementation of ANN circuits allows for the software advantage of flexibility and the hardware VLSI circuit advantage of speed and parallel algorithm acceleration at the same time making that the best hybrid approach to implementing the ANN circuit for different applications in the real-world [18].

5. CONCLUSION

This thesis implemented various models of an Artificial Neural Network using the feedforward architecture and trained using the backpropagation algorithm. The ANN was trained offline using software and hardware was used to implement the feedforward operation of the ANN.

MATLAB was used to train the neural network offline on the training data sets. The circuit implementation of the feedforward algorithm in Verilog requires the connection weights obtained by training the ANN in software to be read by the Verilog model. The neural network results were obtained using statistical testing methods LOOCV and MCCV. RTL code for the prediction part was implemented in Verilog and the synthesized netlist was verified by using a testbench to test the netlist on various input sets. The test samples used in post-synthesis simulation were the same ones used to test the Verilog model in pre-synthesis simulation. The results obtained were the same in both the cases thus verifying the functionality of the synthesized hardware by Synopsys DCS.

The time taken to execute the feedforward algorithm in Verilog was much faster than the time taken to implement the same algorithm in software. Since there are a large number of computations in an ANN [19], software simulation takes a longer time than the hardware realization. The use of fixed-point representation for the real numbers caused a drop in accuracy of $< 1\%$ that can be eliminated by using floating-point representation if desired.

An accuracy of 97% was obtained for handwriting recognition, 85% for plant disease classification and 75% for clay content in soil samples. The poor performance in the latter cases was due to the supervised training algorithm used on the limited/imbalanced training data sets available. The comparisons between the three problems provided in Table 6 show that a better trained ANN model for more uniform and large data sets can be implemented in Verilog for better accuracy.

REFERENCES

[1] Bansal, R., Goel, A., and Sharma, M., "MATLAB® and Its Applications in Engineering: [Based on MATLAB 7.5 (R2007b)]." Upper Saddle River, NJ: Prentice Hall, 2009, pp. 361-71.

[2] Montgomery, D., Peck, E., and Vining, G., "Introduction to Linear Regression Analysis." Hoboken, NJ: John Wiley & Sons, 2012, pp. 527-30.

[3] Jain, A., Mao, J., and Mohiuddin, K., "Artificial neural networks: a tutorial." *Computer*, vol. 29(3), 1996, pp. 31-44.

[4] LeCun, Y., and Cortes, C., "The MNIST Database of Handwritten Digits." [online] 1998, http://yann.lecun.com/exdb/mnist/.

[5] Frank, A. and Asuncion, A., "UCI Machine Learning Repository." [online] 2010, http://archive.ics.uci.edu/ml.

[6] Girau, B., "Neural Networks on FPGAs: a survey." *Second ICSC Symposium on Neural Computation,* Berlin, Germany, 2000.

[7] Leung, K., "Preparing the Data." [online] 2007, http://cis.poly.edu/~mleung/FRE7851/f07/preparingData.pdf.

[8] Rumelhart, D., Hinton, G., and Williams, R., "Learning representations by back-propagating errors." *Neurocomputing: foundations of research*. J. Anderson and E. Rosenfeld. Cambridge, MA: MIT Press, 1988, pp. 696-99.

[9] Thimm, G. and Fiesler, E., "Neural network initialization." *From Natural to Artificial Neural Computation*. J. Mira and F. Sandoval, eds., Springer Verlag: Berlin, vol. 930, 1995, pp. 535-542.

[10] LeCun, Y., Bottou, L., Orr, G., and Muller, K., "Efficient BackProp. Neural Networks: Tricks of the Trade." G. Orr and K.-R. Müller, eds., Springer Verlag: Berlin, vol. 1524, 1998, pp. 9-50.

[11] Priddy, K., and Keller, P., "Artificial Neural Networks: An Introduction." Bellingham, WA: SPIE Press, 2005, pp. 1-17.

[12] Nichols, K., Moussa, M., and Areibi, S., "Feasibility of Floating-Point Arithmetic in FPGA based Artificial Neural Networks." *15th International Conference on Computer Applications in Industry and Engineering*, 2002, pp. 8-13.

[13] Namin, A. H., Leboeuf, K., Wu, H., and Ahmadi, M., "Artificial neural networks activation function HDL coder." *IEEE Transactions on Electro/Information Technology*, 2009, pp. 389-92.

[14] Nguyen, G., Bouzerdoum, A., and Phung, S., "Learning Pattern Classification Tasks with Imbalanced Data Sets." *Pattern Recognition*, Peng-Yeng Yin (Ed.), [online] 2009, http://www.intechopen.com/books/pattern-recognition/learning-pattern-classification-tasks-with-imbalanced-data-sets

[15] Synopsys. "Synopsys Timing Constraints and Optimization User Guide." [Online] 2010, Available: http://acms.ucsd.edu/_files/tcoug.pdf

[16] Wang, L.., Chang, Y., and Cheng, K., "Electronic Design Automation: Synthesis, Verification, and Test (Systems on Silicon)." Burlington, MA: Morgan Kaufmann, 2009.

[17] Shekhar, S., and Amin, M., "Generalization by Neural Networks." *IEEE Transactions on Knowledge and Data Engineering*, vol. 4(2), 1992, pp. 177-85.

[18] Sahin, S., Becerikli, Y., and Yazici, S. "Neural Network Implementation in Hardware Using FPGAs." *Neural Information Processing*, vol. 4234, Springer Berlin Heidelberg, 2006, pp. 1105-12.

[19] Botros, N., and Abdul-Aziz, M., "Hardware implementation of an artificial neural network using field programmable gate arrays (FPGA's)." *IEEE Transactions on Industrial Electronics,* vol. 41(6), 1994, pp. 665-67.

Supplemental Sources Consulted

Soufian, M.; Soufian, M. and Thomson, M. "Practical comparison of neural networks and conventional identification methodologies." *Artificial Neural Networks, Fifth International Conference on* (Conf. Publ. No. 440), vol. 41(6), 1997, pp. 262-67.

Fausett, L., "Fundamentals of neural networks: architectures, algorithms, and applications." Upper Saddle River, NJ: Prentice-Hall, 1994.