1-1-2009

Investigation of integer neural networks for low cost embedded systems

Thomas Behan
Ryerson University

INVESTIGATION OF INTEGER NEURAL NETWORKS FOR LOW COST EMBEDDED SYSTEMS

by

Thomas Behan
Bachelor of Technology in Electronics Engineering Technology
RCC Institute of Technology, Concord, Ontario, Canada, 2007

A thesis
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of
Master of Applied Science
in the Program of
Electrical and Computer Engineering

Toronto, Ontario, Canada, 2009

I hereby declare that I am the sole author of this thesis.
I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Thomas Behan

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Thomas Behan

Ryerson University requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Investigation of Integer Neural Networks For Low Cost Embedded Systems. Master of Applied Science Thesis, Thomas Behan, Electrical and Computer Engineering, Ryerson University, Toronto, 2009.

Abstract

Neural networks have been a topic of study for almost half a century and have become one of the predominant methods used for intelligent systems. During this time much progress has been made on improving the accuracy and expanding the capabilities of neural networks. This thesis is an investigation in a different direction, that is to reduce the computational requirements of neural networks to make them more suitable for implementation on very low end microcontrollers and DSPs. The goal is to understand the trade offs in cost, accuracy, and execution time on these low cost processors. To do this, two tests are performed. The first compares execution speed of a simple neural network on low cost hardware. This test demonstrates the advantages to using integer neural networks, and DSP operations. The second test, is a contrast of the accuracy of an integer neural network and a floating-point neural network. This test uses a real world example and allows for testing multiple levels of quantization. The test results show the effects of quantization due to the use of integers, and the amount of decay to be expected when creating an integer neural network. The results show that there is a strong case for using integer neural networks on low cost microcontrolers, and that significant cost savings can be achieved in exchange for a very small reduction accuracy.

Acknowledgment

I would like to acknowledge and express my gratitude to those who have helped me in writing this thesis.

First and foremost I would like to thank my thesis supervisors, Prof. Lian Zhao and Prof. Zaiyi Liao, for their support and positive attitude throughout my time at Ryerson. This thesis, and my research, would not be possible without their constant encouragement and sage advice. Through their own example they instilled a strong work ethic in all of their students, creating supportive and dedicated working environment. Their commitment to high standards and professionalism has greatly inspired me improve the quality of my own work, and has become the idée fixe of my writings.

I would also like to thank my family for their support and encouragement. Without my fathers weekly inquirings it is unlikely this thesis would have been completed in time, for his keen interest in my success I am forever indebted to him.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The purpose of this thesis is to investigate the capabilities of Artificial Neural Networks (ANNs) on low cost microcontrollers and Digital Signal Processors (DSPs). The field of artificial neural networks has been a topic of study for many years and much progress has been made on expanding the capabilities of neural networks. Since their first inception, artificial neural networks have improved greatly, both in the range of tasks that they can solve, as well as the accuracy with which artificial neural networks can solve these tasks. However, these advances often come with the cost of additional complexity. This complexity has been partially offset by increasingly powerful computers. On modern computer hardware, very large networks can be simulated in reasonably short amount of time, leading to a large and varied range of applications. However, for low cost systems, the processing power needed to simulate these networks can be prohibitive. While processor technology continues to advance, the fact remains that simpler is most often cheaper, especially with respect to hardware. For this reason, low end microcontrollers and DSPs will always rely on having reduced circuit complexity in order to keep down production costs. Reduced circuit complexity means that low cost microcontrollers and DSPs may not have the hardware to perform some math operations natively. These operations be must be performed in software, leading to greatly reduced processing power.

1.1 Purpose Breakdown

1.1.1 Integer Neural Networks

The lower cost of less complex circuity creates a case for simplifying artificial neural networks. If the computational complexity can be reduced, then cost of the hardware required to execute the network in a reasonable time frame can also be reduced. In this thesis, the focus will be on using Integer Neural Networks (INNs) to reduce computational complexity. The hardware for performing integer calculations is much simpler than the hardware for floating-point hardware. This reduction in hardware complexity will allow for lower cost implementations of neural networks.

1.1.2 DSP Operations

A secondary investigation is into low cost DSPs. Many floating-point neural networks are implemented on DSPs that have special instructions that can be used to accelerate the calculation of the network. If low cost implementations are to receive wide acceptance, it will be necessary for them to have comparable execution time. For this reason, the execution time of a neural network must be contrast both with, and without DSP operations. The relative performance gap is very important in selection of a processing unit for any application, and neural networks are no exception.

1.1.3 Comparison

Having established a floating-point neural network as the base of comparison, the different combinations of hardware, and integer neural network will be examined. The key points to this comparison are, the cost of the processor, the accuracy of the network, and the execution time of the network. In addition, power consumption is also reduced when using simpler hardware. However the total power consumption is highly subjective and dependent on the other factors, such as processor type, clcok speed, and time spent in idle mode.

1.2 Thesis Structure

1.2.1 Introduction and Background

The first section of the thesis contains the introduction and background of the topic. Chapter 1 provides a basic overview of the thesis, its purpose, goals, and methods. In Chapter 2, the background of subject is examined. An overview of neural networks, integer neural networks, and the hardware under consideration is given. This establishes the current state of the art which the thesis expands on.

1.2.2 Integer Neural Networks

Chapter 3 is a special section on integer neural networks. Because integer neural networks are so central to this thesis, it is important to describe it in great detail. This provides the background on how the network operates, and of the problems faced when training integer neural networks. The ideas of this chapter are used to explain the specifics of the implementation of the tests performed in later chapters.

1.2.3 Contributions

Chapter 4 and Chapter 5 contain the tests that confirm the hypotheses. Chapter 4 concerns the performance tests on low cost microcontrollers. This test shows that integer neural networks using DSP operations are the fastest, followed by integer neural networks, followed by floating-point networks on low cost microcontrollers that lack floating-point instructions. Chapter 5 investigates the effects of quantization. Using a small range of integer values in the network is important in reducing memory usage. However a small range of values degrades the accuracy of the network using a real world test case. Chapter 5 shows the effects of various rates of quantization, and provides a simple method for comparing networks with variable rates of quantization.

1.2.4 Conclusion

The end of the thesis is Chapter 6 which is the conclusion. Here the work of the thesis is reviewed. This chapter explains how the work done in this thesis demonstrates the effectiveness of using integer neural networks on low cost hardware. This chapter also reflects on the considerations for future work, and emphasizes the balance of trade offs between cost, speed, power consumption, and accuracy.

Chapter 2

Background

This chapter provides a background for the work done in this thesis. Artificial Neural Networks (ANNs) are discussed in detail. This provides a starting point for the work done in this thesis as it moves from artificial neural networks in general, to integer neural networks which will be the major focus of this thesis. Integer neural networks are the section 2.2 and provide a brief understanding of what integer neural networks are and where they are situated in the realm of artificial neural networks. Once the properties of integer neural networks are defined, it is possible to describe the type of hardware being used. This forms the third section of this chapter. Finally a review is done on previous work relating to integer neural networks, specifically DSPs, power consumption, and Field Programable Gate Arrays (FPGA)s.

2.1 Artificial Neural Networks

2.1.1 Purpose

Artificial neural networks are mathematical models that attempt to simulate the workings of a brain, which is a biological neural network. Artificial neural networks derive functionality from their ability to adapt the weighting of the connections between the neurons of the network. In this thesis supervised learning will be used to train the networks. This means that the network is trained with a series of inputs and target values, with the goal being for the network's output equal to the target value for a given set of inputs. By repeatedly feeding

inputs into the network and updating the weights of the inter-neuron connections, a network will emerge with the ability to extract the features and relationships from the input values to produce the output values. This ability to learn and extract the salient features of the inputs is the key to neural network operation. The ability to create machines that can intelligently predict and classify without having to be explicitly programmed based on mathematical models, is very useful. There are many applications for such machines across a wide range of disciplines. Some of these applications include, power systems [1], steel manufacturing [2], optical processing [3], finance [4], traffic flow [5], and control problems [6]. Artificial neural networks are an old and promising field of study within machine learning. The primary goal of this thesis is to demonstrate the feasibility of extending this body of knowledge to low cost embedded systems.

2.1.2 Structure Of Artificial Neural Networks

The structure of an artificial neural network is taken directly from its biological counterpart [7]. The base element is called a neuron. A neuron has many inputs and only one output as shown in Fig. 2.1. The inputs (I) to a neuron are weighted (W) independently, then are combined with a scalar bias (B) and passed through a non-linear transform(f(x)) to produce an output (O),

$$O = f(W \cdot I + B) \tag{2.1}$$

Individually a neuron is quite simple in operation. The key to the power of artificial neural networks lies in the connections between neurons, making neural networks a connectionist form of computation. An important result of the connectionist nature is that artificial neural networks are massively parallel, making them inherently suited to distributed computing and vector processors [8].

2.1.3 Training Methods

The backpropagation method discussed in this thesis updates the weights of the network based on error at the network output. The sum squared error (SSE) between the target

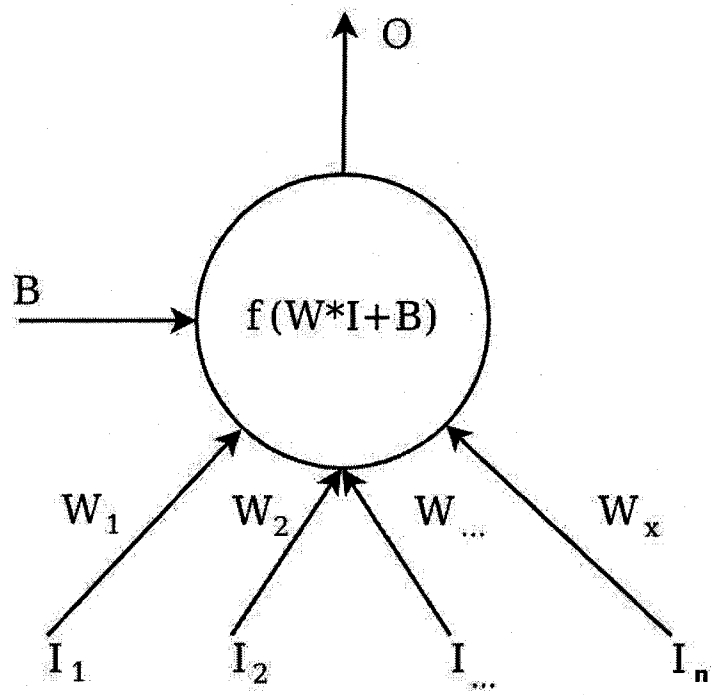

Figure 2.1: Block diagram of a neuron.

value (T) and the networks output (O) is used as the error (also called cost) function to calculate the total error of the system, which we would like to minimize, and used to update the weight values. The (SSE) is divided by half to make the derivative of the error function simpler, thereby reducing the time to calculate the weight updates, each weight of each neuron, written as

$$E = \frac{1}{2}\sum (T - O)^2 \tag{2.2}$$

For example updating the weight of a neural network using sum squared error (2.2) as the error function, and gradient decent (2.13) as the training method and the standard neuron output,

$$O = f(W \cdot I + B) \tag{2.3}$$

Now the B value can be considered a weight with a constant input of 1, doing this simplifies the equation. In this way the bias functions as an additional element in both I and W

vectors. This results in,

$$O = f(W \cdot I) \tag{2.4}$$

The resulting equation (2.4) is used with the error function and the result is,

$$E = \frac{1}{2}\sum (T - f(W \cdot I))^2 \tag{2.5}$$

The gradient decent method is then applied to (2.5) to minimize the new error function,

$$\frac{\partial E}{\partial W} = \frac{\partial \frac{1}{2}\sum (T - f(W \cdot I))^2}{\partial W} \tag{2.6}$$

From this point each weight has its own weight updated, with the error function derived for it,

$$\frac{\partial E}{\partial W} = \frac{\partial \sum (T - f(W \cdot I))}{\partial W} \tag{2.7}$$

$$\frac{\partial E}{\partial W} = \frac{(\sum (T - f(W \cdot I)))\partial f(W \cdot I)}{\partial W} \tag{2.8}$$

$$\frac{\partial E}{\partial W} = \frac{(\sum (T - f(W \cdot I)))f'(W \cdot I)\partial W \cdot I}{\partial W} \tag{2.9}$$

$$\frac{\partial E}{\partial W} = (\sum (T - f(W \cdot I)))f'(W \cdot I)I \tag{2.10}$$

Finally this equation (2.10) can be expressed more contently as,

$$\delta = \frac{\partial E}{\partial W} = (\sum (T - O)) \cdot f'(W \cdot I) \cdot I \tag{2.11}$$

Training neural networks can be viewed as an optimization problem, and a subset of machine learning. Methods such as Newton–Raphson(2.12), gradient descent (2.13), Gauss–Newton (2.14), and Levenberg–Marquardt (2.15) are common methods.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.12}$$

$$x_{n+1} = x_n - \gamma_n \nabla F(x_n) \tag{2.13}$$

$$S(\beta) = \sum_{i=1}^{m} r_i^2(\beta) \tag{2.14}$$

$$S(\beta) = \sum_{i=1}^{m} [y_i - f(x_i, \beta)]^2 \tag{2.15}$$

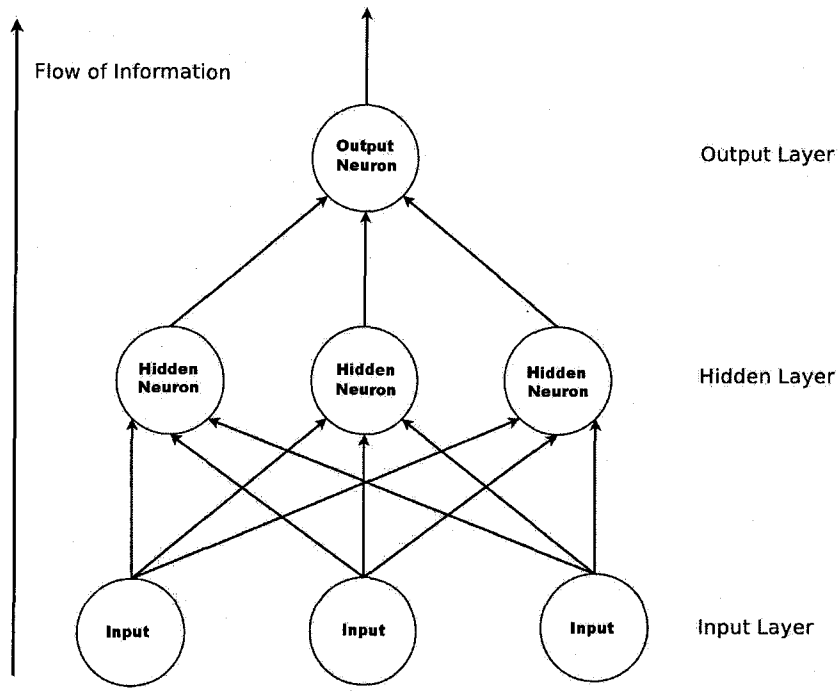

Figure 2.2: Feedforward network.

In addition, a combined approach has also been proven to have some advantages. A variety of combinations such as neural networks utilizing genetic algorithm [9, 10, 11], simulated annealing [12, 13, 14], and fuzzy logic [15, 16, 17] can be utilized to produce hybrid schemes. These methods aim to combine the characteristics of different machine learning techniques to produce improved characteristics.

2.1.4 Network Architectures

While this thesis will focus exclusively on feedforward networks, there exist several architectures for artificial neural networks which remain unexplored for low cost implementation. The feedforward network is the simplest architecture. The network is divided into layers and all outputs flow to the next higher layer. In this way the flow of information is unidirectional with clearly defined connection paths as shown in Fig 2.2. Another popular type on network is the recurrent network. This architecture follows a similar pattern to the feedforward with the addition of loop back connections. In this type of network the output is fed back into

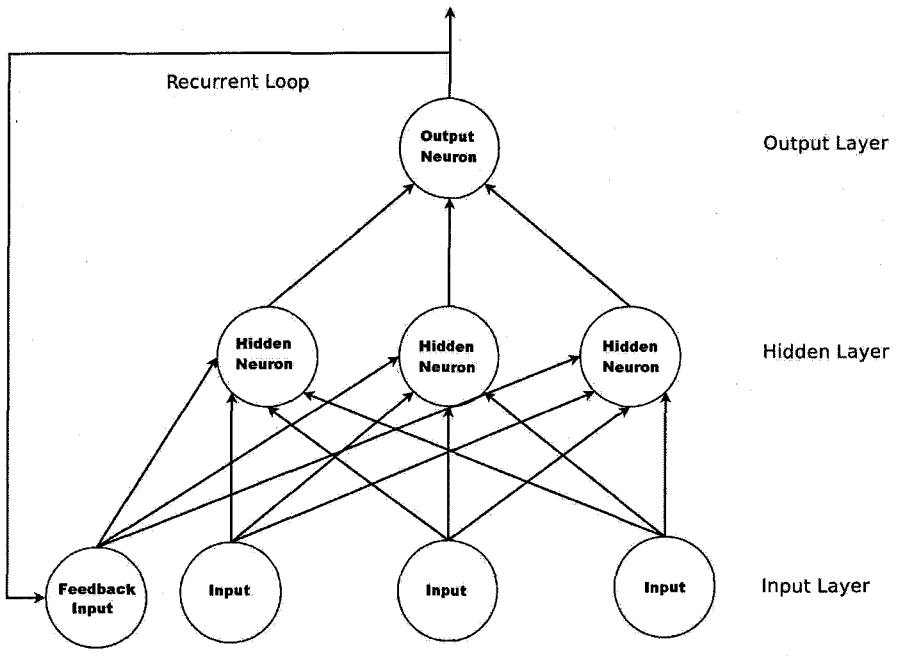

Figure 2.3: A simple recurrent neural network.

itself, in this way the network retains memory of previous values, making it especially useful for learning dynamic systems [18, 19, 20]. In this situation one or more of the inputs (I) is an output (O) from a previous execution (2.16). A simple form of this network with only one recurrent loop is shown in Fig. 2.3.

$$I_N = O_{t-1} \tag{2.16}$$

Other types of network topologies include radial bias function network [21, 22, 23], Hopfield network [24, 25, 26], echo state network [27, 28, 29], stochastic neural network [30, 31, 32] . Additionally there are methods of combining networks by using each network as an isolated independent unit that is attached to a larger network [33, 34, 35]. Sometimes these units compete in what is called a competitive neural network [36, 37, 38]. These types of architectures are based on having different networks in the hopes of providing a more robust and diverse solution. While the number of architectures is broad, little work has been done concerning low cost implementation and they present a large selection of possible future works based on this thesis.

2.2 Introduction To Integer Neural Networks

2.2.1 Motivation

A central theme of this thesis will be the use of integer neural networks which are of particular interest when dealing with very low cost microcontrollers. Integer neural networks are implementations of artificial neural networks that only use integer values, as opposed to conventional implementations that use floating or fixed-point values. Using integer only values can greatly reduce the computational complexity for hardware that does not natively support floating and fixed point instructions.

2.2.2 Challenges

However using only integer values can greatly reduce the accuracy of the network, by rounding to the nearest whole value. Integer values are granular and so limit weight values that network can have. This in turn limits the ability to produce accurate outputs [39, 40, 41]. Additionally integer neural networks create difficulties in training a network. This is again due to the granularity of integers. In this chapter the effects of limiting values to whole numbers will be explored, as well as the reasons behind the need for integer neural networks and the types of hardware that can expect an increase in performance when using integer only networks. Additionally, we will look at the advantages of using DSP operations that are now available on some of these low cost microcontrollers. Later in Chapter 3 integer neural networks will be examined in greater detail, but first the motivation for using them must be made clear.

2.3 Hardware

2.3.1 Floating-point vs. Integer

The motivation behind using integer neural networks is to reduce the final cost and power consumption of a system that uses a neural network. This is possible because microcontrollers that contain the hardware for performing floating or fixed point operations are significantly

more complex than integer versions, and thus more expensive, both in terms of cost and energy consumption. Early Personal Computers (PCs) did not include floating-point hardware for this very reason. IBM computers using the Intel x86 family of processors did not include floating-point hardware until the release of the 486DX, and the Intel 486SX was released as a cheaper version of this processor without floating point hardware. Any floating-point operations had to be done in software which greatly reduced execution speed, or via the Intel 8087 floating-point co-processor. Apple computers using the Motorola 68000 operated in a similar fashion with the 68881/68882 as the coprocessor. While modern computer designs have moved past these limitations, and implemented the floating-point unit in the main processor, selecting a microcontroller for a low cost application makes this a pertinent consideration. Microcontrollers that include floating-point hardware can cost many times more than an equivalent integer only version.

2.3.2 DSP Operations

Recently new integer-only microcontrollers (such as the Microchip dsPIC30fxxxx and dsPIC33fxxxx series chips, example shown in Fig. 2.4) have been released that include some Digital Signal Processor (DSP) operations. DSP operations have previously been shown to greatly increase the performance of floating-point neural networks. In Chapter 4 it will be demonstrated that DSP operations have a similar effect on integer neural networks. These new DSP-capable microcontrollers are more expensive than regular microcontrollers, however they are still much cheaper than floating-point capable units. The DSP operations allow for a neural network to be executed much faster than before, with only a moderate increase in cost of the system. DSP operations, specifically the MAC (Multiply ACcumulate) instruction, are well suited to increasing the performance of neural networks, as shown below in Fig. 2.5. Typically the MAC instruction uses two pairs of registers and a special accumulator register. In each pair, one register holds the address of the data and one register holds the data itself. On every execution of the MAC instruction, data is moved from the addresses stored in the address registers into the data registers. The address registers are then post incremented to

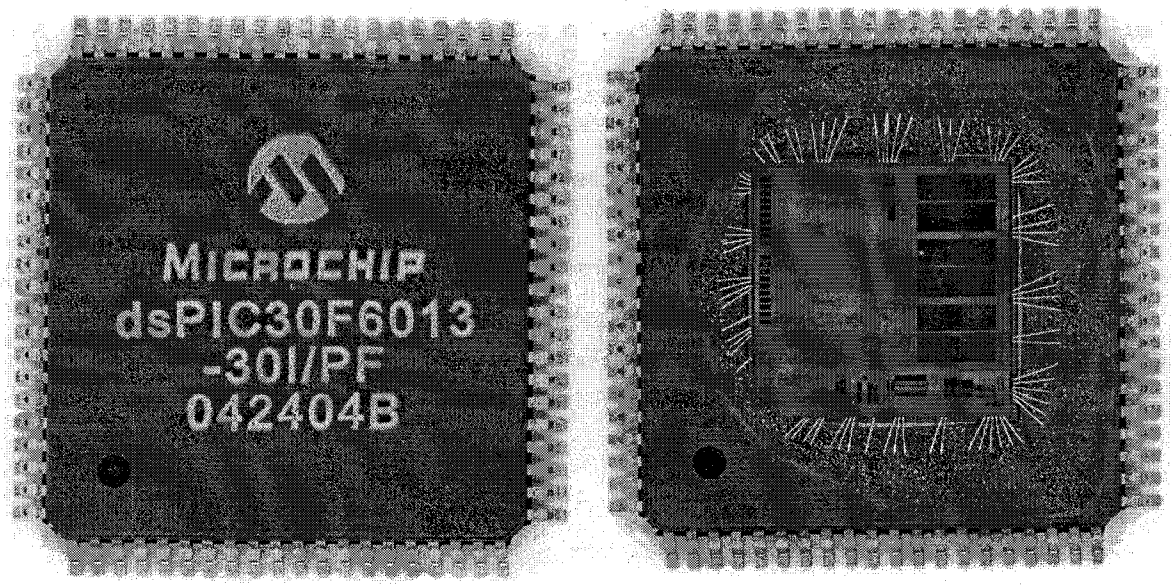

Figure 2.4: dsPIC30f6013 from the dsPIC30fxxxx family.

the next memory location. Finally, the two values in the data registers are multiplied and added to the special accumulator register. The accumulator register is typically much larger than normal registers to prevent the data from becoming so large that the accumulator overflows. The MAC instruction is typically used to accelerate discrete convolution. Discrete convolution (2.17) is used in many applications, but is especially noteworthy for its use in Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) digital filters. These filters are very commonly used, and one of the major applications for DSPs.

$$(f \cdot g)(n) = \sum f(m) \cdot g(n-m) \tag{2.17}$$

$$a \leftarrow a + b \cdot c \tag{2.18}$$

The MAC instruction can accelerate this operation by combining all of the accumulation, multiplication, increments, and move operations in one instruction cycle (2.18). This means that only one instruction cycle is needed for every element in an array, rather than the many instructions that would otherwise be required. This is important for neural networks because convolution and the calculation of the net value of a neuron are mathematically identical.

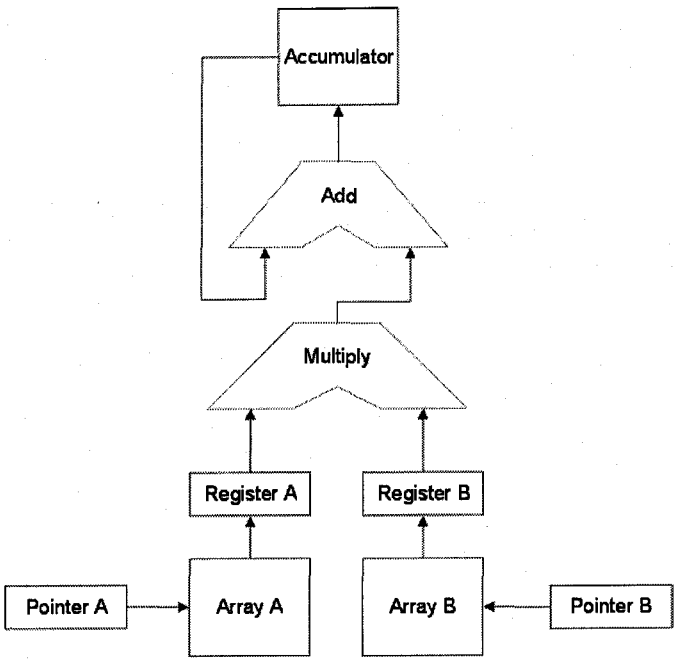

Figure 2.5: Simplified MAC Instruction Block Diagram.

The net value (N) is the sum of the products of the inputs (I) and the weights (W) (2.19). The output (O) of a neuron is a function of the accumulated products of all the weights and inputs in that neuron,

$$N = I \cdot W \tag{2.19}$$

$$O = f(I \cdot W) \tag{2.20}$$

This implies that neural networks can benefit from the MAC instruction in the same way that digital filters do. As we will see later in this chapter, using the MAC instruction can greatly increase the speed at which a neural network can be executed.

2.4 Previous Work

2.4.1 DSP Acceleration

As previously discussed DSP operations can greatly accelerate neural networks. While this has been throughly demonstrated [42, 43, 44, 45, 46] on floating-point hardware, with good

results. In this thesis it will be shown that motivation for using DSP operation remains when using integer neural networks. It is imperative that low cost integer neural networks are capable of performing with equivalent execution speed so that they are a valid option for implementations that require a high sampling rate. Without having comparable speed performance, integer neural networks would be much more limited in their range of application. The issue of comparable execution time also has important implications for power consumption.

2.4.2 Power Consumption

While it is readily apparent that reduced complexity leads to lower cost hardware, power consumption is a more complex issue. While it is clear the simpler hardware will consume less power, this is not always the case. As shown in [47] the power consumption of a processor is dependent on both the number and type of operations being performed. As will be demonstrated in Chapter 4, integer neural networks execute much faster than floating-point networks on low cost hardware. This means that the processor consumes less power executing the network. But further than this, it also means that the processor can spend more time in low power idle mode. This issue of comparative processing time holds true when comparing floating-point DSP and integer DSP processors. The integer DSP system will consume less power because it takes the same amount of calculations while consuming less power per operation due to its simpler hardware. However the case is not clear when contrasting floating-point DSP, and integer processors without DSPs. Here the issue is between the power saved by completing the network faster, and the lower power consumption per operation of the simpler hardware. Because the total power savings is depended on both the idle power consumption and the power consumption for each operation, the total power consumption is highly dependent on the specific processors used. Any comparison made would be specific to the processors tested, however is known that in general power consumption is lower for integer only chips [48]. For this reason it remains an important consideration when selecting the type of network and processor to be used in an actual implementation. While power

consumption is especially important for mobile applications where extending battery life is a key consideration of the design. It is also important for the fact that electricity costs money. Reducing the power consumption can save operating costs in addition to the up front manufacturing savings.

2.4.3 FPGA

By far the most relevant work to this thesis has been implementation of integer neural networks on FPGAs [8]. FPGAs allow for easy construction of custom processors, especially processors which have many parallel components [49]. This ability makes neural networks and FPGAs a natural fit. The FPGA is able to execute all of the neurons in the neural network in parallel making the system very fast. The only limitation is the size of network that the FPGA can contain. This is determined by the number of neuron as well as the circuit complexity of each neuron. It is for this reason integer neural networks are of interest for FPGA implementations. The simplicity of the hardware for calculating integer neural networks means that an integer neural network consumes much less fabric space and power consumption than an equivalent floating-point unit [48]. This has two major advantages. First this allows FPGAs to hold much larger networks. And second for small networks the FPGA needs to operate less gates, resulting in lower power consumption. These benefits mean that the largest body of work for integer neural networks on embedded system comes from the investigation of integer neural networks on FPGAs. This range of applications explored via FPGA implementation [8, 49] offers a solid foundation for the establishing the practicality of integer neural networks for embedded system. A comparison between FPGA and low cost DSP integer neural network is outside the scope of this thesis, but it is clearly a possible point for future investigation into reducing power consumption and decreasing execution time. The major concern with FPGA implementations is the cost, FPGAs are significantly more expensive than the microcontrollers, due to their complex circuity. Additionally FPGAs are known to be relatively inefficient in power consumption for the same reason. This contrast suffers from the same subjectivity issue as floating-point

verses integer hardware. As a results a comparison would be difficult. However, generally a low cost microprocessor consumes much less power than FPGAs for low speed integer neural networks.

Chapter 3

Integer Neural Networks

Integer neural networks are the key to extracting maximum execution speeds from low cost microcontrollers. The feedforward operation of these networks is quite simple and will be throughly tested in Chapters 5 and 6. Creating an efficient integer only feedforward network is important to making integer neural networks practical on low cost systems. However training an integer neural networks without floating-point values is a significant task and much literature [40, 39] is dedicated to this topic. A simple method of backpropagation is described in this section and is used to create the test network in Chapter 4.

3.1 Feedforward Networks

3.1.1 Topology

The simplest form of neural network is a feed forward network. In this network, the inputs are fed into one end of the network and the output is produced at the other. All the neurons in one layer connect to the neurons in the next layer. An integer neural network of this type will be explored later in this chapter. However, there are many other types, such as stochastic networks [30, 31, 32], recurrent networks [18, 19, 20], radial bias networks [21, 22, 23],, etc. Each type of network has its own advantages and disadvantages. The feedforward network was chosen as the example network for its simplicity. Fig. 3.1 shows a simple 2-2-1 neural network that is used later for testing. The output of a neuron is a function of the accumulated product of the neurons inputs and its weights. The function is

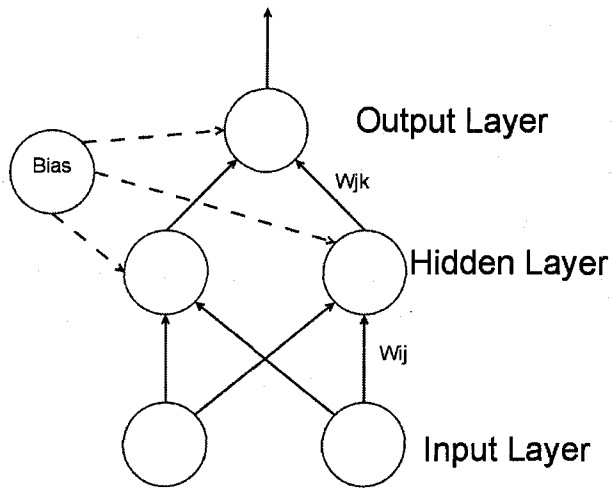

Figure 3.1: Diagram of a 2-2-1 neural network.

called the activation function and it is most often a sigmoid to simplify the calculation of the derivative when performing backpropagation. There are several types of activation functions such as the logistic function (3.1), arc-tan (3.2), and hyperbolic tan (3.3), but generally they perform the same purpose. Most importantly it is from the activation function that a neural network achieves non-linearity in the output of a neuron. Also the activation function places limits on the magnitude of the values inside the network. Limiting the size of the values in the network is an important consideration for integer neural networks as the range of values is much more limited than floating-point values, due to the need to express all values as whole numbers.

$$P(t) = \frac{1}{1+e^{-t}} \tag{3.1}$$

$$y = arctan(x) \tag{3.2}$$

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{3.3}$$

3.1.2 Activation Function

Typically an activation function output ranges from 0 to 1 or -1 to +1. While this is not an issue for floating-point networks, it is for integer neural networks. Taking the second

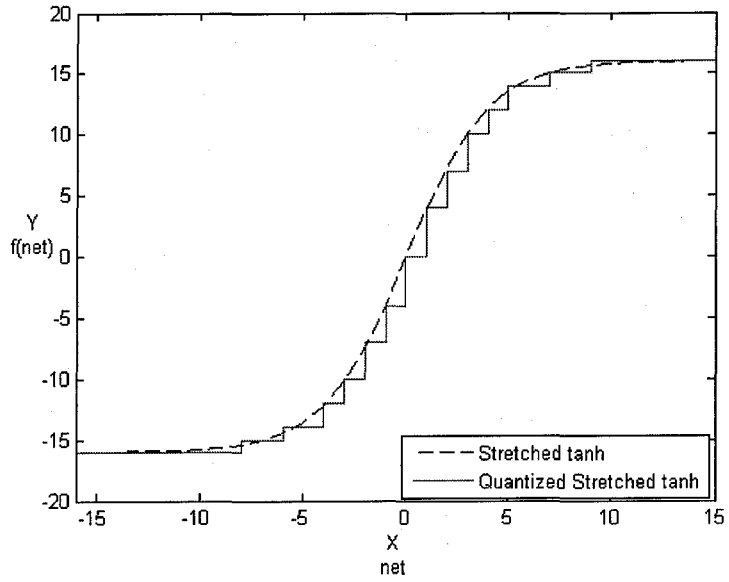

Figure 3.2: Stretched activation function (tanh).

case, an integer neural network would quantize the output of the activation function into -1, 0, +1. This does not provide nearly enough selectivity to provide accurate classifications or predictions. One method to avoid this is to stretch the activation function so that more function outputs exist as whole values. Fig. 3.2 shows the activation function, tanh, stretched according to,

$$Y = 16 \cdot \tanh\left(\frac{X}{4}\right) \tag{3.4}$$

This quantized activation function can be stored in a Look Up Table (LUT) so that a tanh function, such as its Taylor series expansion, does not need to be executed, which further increases performance. A LUT stores all of the output values sequentially in memory. When the function is called, the input (I) is added to a constant offset (C) to determine the memory address. The value at that memory address (M(I+C)) is the output (O) of the function (3.5). Fig. 3.3 provides and example of the operation of a LUT.

$$O = M(I + C) \tag{3.5}$$

In this way only one addition needs to be used to get the result. Using a LUT is much

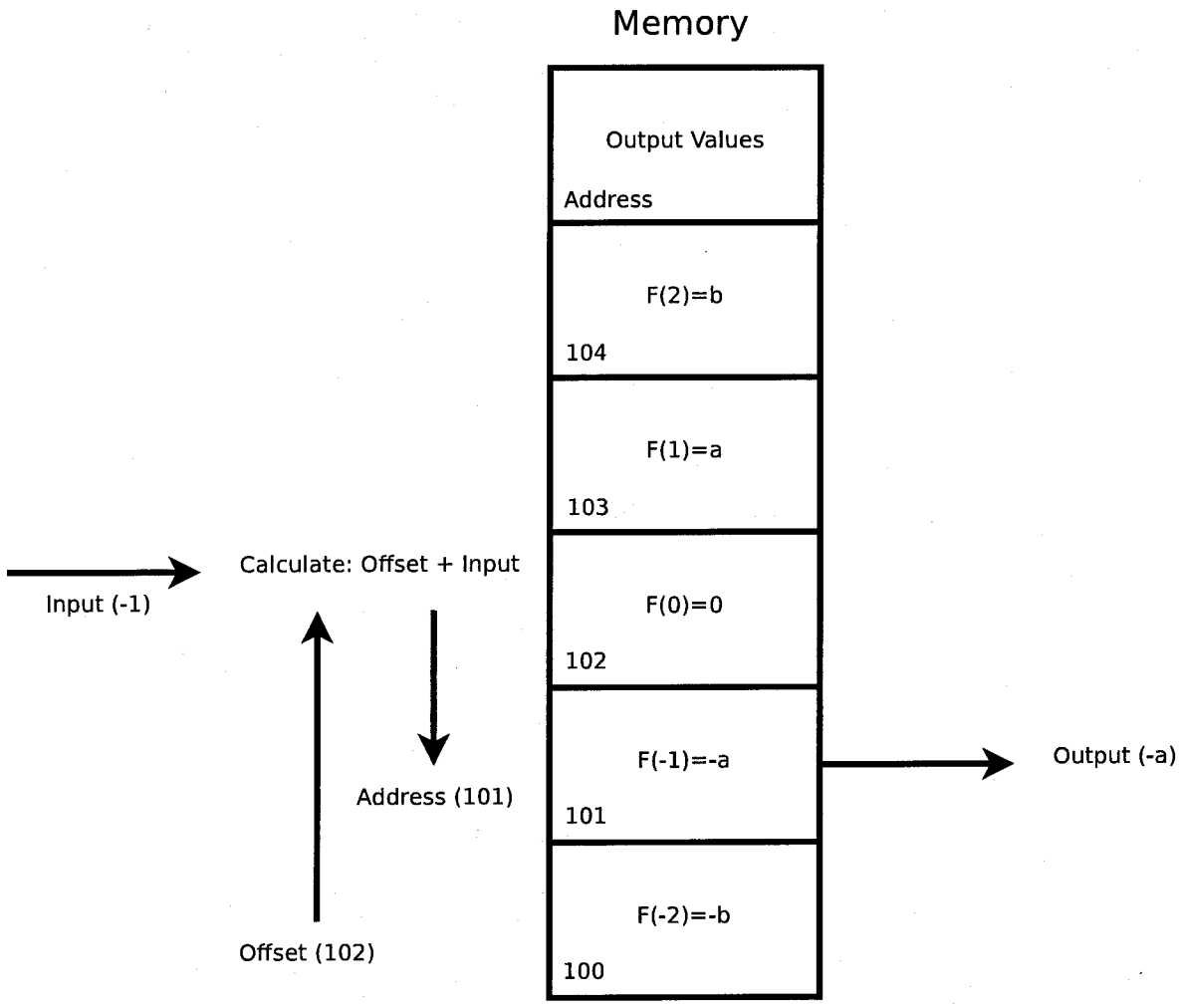

Figure 3.3: Example operation of a LUT.

faster than calculating tanh even if the microcontroller supports floating-point operations. This means that the tanh value does not have to be calculated which can be a time and resource consuming task. Another time saving feature is that because sigmoids have simple derivatives calculation time can be saved in training. For example, the derivative of the tanh function (3.6) contains the original tanh calculation in it,

$$f(x) = tanh(x) \tag{3.6}$$

$$f'(x) = (1 - tanh^2(x)) \tag{3.7}$$

From this we can modify the equation to allow for any amount of scaling in either the output space, A, or the input space, B. Then solve for the derivative,

$$f(x) = A \cdot tanh(\frac{x}{B}) \tag{3.8}$$

$$f'(x) = \frac{dA \cdot tanh(\frac{x}{B})}{dx} \tag{3.9}$$

$$f'(x) = A \cdot \frac{dtanh(\frac{x}{B})}{dx} \tag{3.10}$$

$$f'(x) = A \cdot (1 - tanh(\frac{x}{B})) \cdot \frac{d\frac{x}{B}}{dx} \tag{3.11}$$

$$f'(x) = \frac{A}{B} \cdot (1 - tanh(\frac{x}{B})) \tag{3.12}$$

Here it is convenient to introduce $g(x)$ to make explicit the final derivative,

$$g(x) = tanh(\frac{x}{B}) \tag{3.13}$$

$$f(x) = A \cdot g(x) \tag{3.14}$$

$$f'(x) = \frac{A}{B}(1 - g^2(x)) \tag{3.15}$$

From here is can be seen how by saving the output of $g(x)$ (3.13) when performing the feedforward operation (3.14) the value can be reused for performing backpropagation (3.15). In the case used for the first test (3.4) $A = 16$ and $B = 4$. By substituting these constants into the equations derived above (3.13)(3.14) (3.15) we can arrive at both the original activation function (3.17) and its derivative (3.19) with only one tanh calculation (3.16), or call to the

LUT.

$$g(x) = tanh(\frac{x}{4}) \tag{3.16}$$

$$f(x) = 16 \cdot g(x) \tag{3.17}$$

$$f'(x) = \frac{16}{4}(1 - g^2(x)) \tag{3.18}$$

$$f'(x) = 4 \cdot (1 - g^2(x)) \tag{3.19}$$

3.2 Backpropagation

3.2.1 Floating-Point Network

Training of a neural network is usually done with floating point values, and this method has been quite successful. The most common method of training a neural substituting is back-propagation. In a normal neural network the magnitude of the weight updates is attenuated by the decimal values in the network. Additionally a small constant, η, is multiplied directly to the weight update ($\Delta\omega$) to further reduce the rate at which the weights are updated as seen in (3.21). This is important because the network must gradually move to a solution, by having very small weight updates the network does not skip over possible solutions. This allows the network to gradually move towards a solution.

3.2.2 Integer Neural Network

However in an integer neural network, the magnitude of the values in the network are always equal to or greater than one. This means that the weight updates have the potential to become very large after successive multiplications (3.20). When the weight updates are very large the output of the network will change drastically after each update, preventing the network from moving toward a solution. The second major challenge is in the derivative of the stretched tanh function. Fig. 3.4 shows that only a small portion of the input space for the derivative of the activation function is non-zero. This means that for a large range of net values the weight update is zero. This zero cancels out the rest of the factors of the

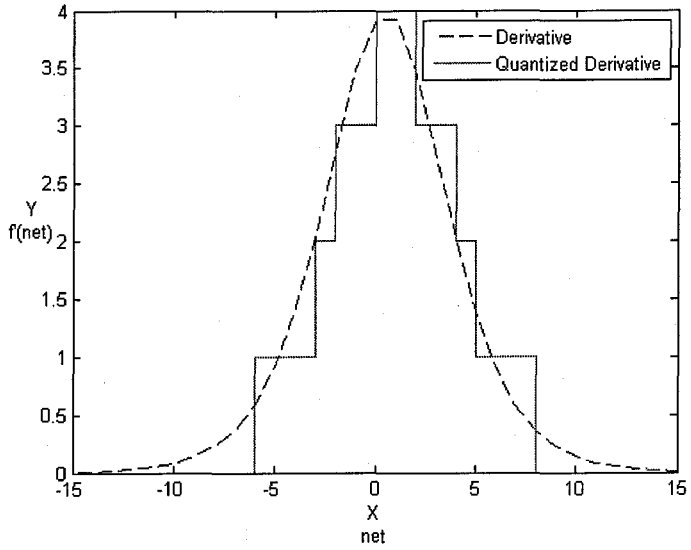

Figure 3.4: Stretched differential of the activation function (tanh).

equation and the network does not move towards a solution,

$$\delta = (T - O) \cdot f'(N) \cdot I \tag{3.20}$$

$$\Delta\omega = \eta\delta \tag{3.21}$$

Equation (3.21) shows how the weight update is determined for the output neuron, and how the value η is used to control the rate of weight convergence. The input weight for a neuron is calculated by multiplying the input, I, the derivative of the activation function $f(N)$, and the difference between the target value and the output (3.20) . For values of net where $f'(N)$ is zero, δ is also zero, leading to $\Delta\omega$ the weight update, to also be zero (3.20). Without the use of the derivative of the stretched tanh function, normal backpropagation cannot be performed.

3.2.3 Fixed Weight Update Magnitude

There are many approaches to this problem, such as fixed weight updates [7], as shown in the following equations,

$$\delta = (T - O) \cdot I \tag{3.22}$$

$$\Delta w = \begin{cases} 1 & \text{if } \delta > 0 \\ -1 & \text{if } \delta < 0 \end{cases} \tag{3.23}$$

This simplified backpropagation is used in Chapter 4. It allows the weights to be updated by small steps, avoiding the need for η to reduce the magnitude of the weight update. It also removes the need to calculate the derivative of the activation function. However the down side to this approach is that the network is trained very slowly and it can be difficult for the weights to diverge because they are being changed at the same rate every epoch. Other methods such as [7] and [50] provide some other methods of weight updates for integer neural networks using different solution, however these solutions, while an improvement, suffer from their own drawbacks. Training integer neural networks remains much more difficult than training floating-point networks.

Chapter 4

Performance Comparison

This chapter explains the test setup for contrasting execution times for both integer and floating-point networks on low cost microcontrollers. In addition this chapter also compares the execution time for integer neural networks with and without DSP operations. The evidence provided in this chapter proves that integer neural networks offer a significant performance improvement, and the DSP operations can further decrease execution time.

4.1 Test Setup

4.1.1 Execution Time Considerations

The first consideration when choosing an integer neural network is execution time. If the target system has timing constraints that are easy to meet, then it may be possible to use a floating-point network on low cost hardware despite the performance drawbacks. However it is often the case that faster execution is required to meet the demands of the application. Additionally a faster network allows the microcontroller to operate at a reduced clock speed or spend more time in an idle state, and thereby consume less power.

4.1.2 Hardware Selection

To compare the performance of integer neural networks and floating-point neural networks a very simple 2-2-1 neural network, shown in Fig. 4.1, was trained to solve the XOR problem and then implemented in hardware. The XOR problem is one of the simplest problems

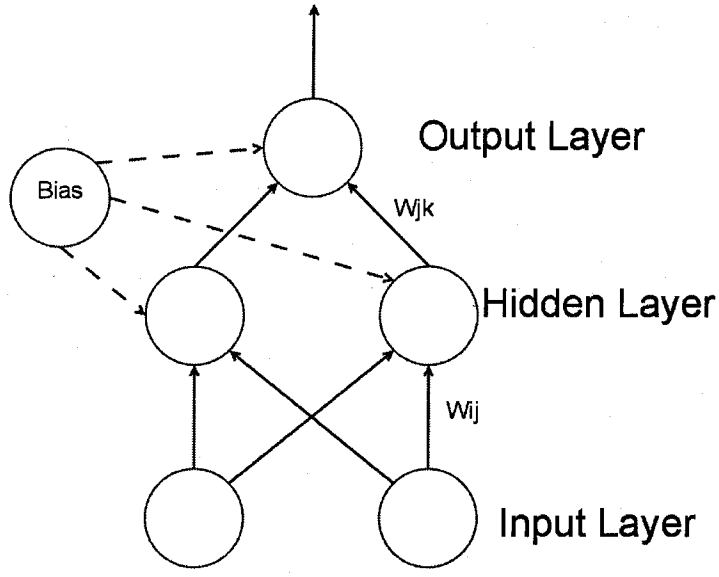

Figure 4.1: Feedforward neural network for solving XOR.

Table 4.1: XOR truth table.

Input A	Input B	Output
0	0	0
1	0	1
0	1	1
1	1	0

that requires a multilayer network, and is quite common in testing new neural networks techniques. The XOR problem is also useful because the solutions are definitive, the network either solves the XOR problem or it does not, comparisons of accuracy are not necessary. Table 4.1 shows the truth table for the XOR problem. For testing, a dsPIC30F2011 was chosen as being representative of the type of low cost DSP capable microcontrollers that are best able to benefit from integer neural networks, as well as demonstrate the effects of DSP acceleration(see Appendix B for a comparison of a selection of microcontrollers). For timing purposes the number of clock cycles used was chosen as the base measurement as it is independent of the type and clock speed of the microcontroller.

4.1.3 Network Implementation

Equations (4.1), (4.2), and (4.3) show the output of the neurons at the output, hidden, and inputs layers respectively.

$$Out_k = f(\sum W_{jk} \cdot Out_j) \tag{4.1}$$

$$Out_j = f(\sum W_{ij} \cdot Out_i) \tag{4.2}$$

$$Out_i = x_i \tag{4.3}$$

At the output layer (4.1) the output is the sum of the weights (W_{jk}) and the outputs from the previous layer (Out_j), which is also called the Net value. The Net is then transformed by the activation function, and the result is the output. The output of the hidden layer (4.2) is the same, except the previous layer is now the input layer (4.3). The input layer (4.3) is simply the inputs to the network. No operations are performed at the input layer (4.3). As can be seen from Table 4.1 the sign of the output (4.4), O, of the network output determines whether the result, R, should be 1 or 0.

$$R = \begin{cases} 1 & \text{if } O > 0 \\ 0 & \text{if } O < 0 \end{cases} \tag{4.4}$$

The magnitude of the networks output is only used to demonstrate maximum separation between the two output and network robustness. Other values can still provide an accurate result, as long as there is a clear separation at the output between the results that should evaluate as 1 and those that should evaluate as 0.

4.1.4 Training Method

The training method used for this network is discussed in Chapter 3. The fixed magnitude weight update method is a very simple and fast way to train a network. It was used to demonstrate the potential for training a neural network entirely with integers, however due to the drawbacks discussed in Chapter 3, this method is not used for the second test. To ensure that maximum accuracy is provided, the network was trained so that output values of -16 represent false, and 15 represent true as shown in Table 4.2. This ensures that the gap

between true and false are at the limits of the lookup table, giving maximum separation. Because there are only four possible input and output combination the network is trained

Table 4.2: XOR network output.

Input A	Input B	Output
0	0	-16
1	0	15
0	1	15
1	1	-16

with the entire input space. All of the values are initialized to small (magnitude less then 16) random values. The network is then trained until the error for each output is zero. The network will not always converge to a solution due to the limitations of the constant magnitude update method. In these cases the network must be re-trained with new random initial variables. Once the network was trained the weights were programmed into the microcontroller for testing.

4.2 Results

4.2.1 Comparison Accuracy Considerations

Three variants of the code were written: one with floating-point variables and constants; one with integer variables and constants; and one with integer variables, constants, and DSP operations. Note that there is no floating-point with DSP because the hardware does not support this. The DSP operations are not available for floating-point operations as they are special instructions built into the microcontroller which does not support floating-point operations natively. To maintain an accurate comparison, the output of the floating-point net calculation was set to zero so that it may use a LUT. This was done to ensure that the performance increase shown is the result of using integers and not because of the LUT.

4.2.2 Execution Time

The net calculation is the number of clock cycles needed to calculate the net value for one neuron. "One Neuron" is the number of clock cycles needed to calculate the output of a single neuron, the net calculation plus the use of the LUT. Finally whole sample set is the number of clock cycles needed to calculate the output of the network for the whole input space. The time to execute these parts of the neural network in clock cycles is shown in Table 1. Fastest times are in bold. As seen in Table 1, the integer neural network is the

Table 4.3: Execution Time In Clock Cycles (CC)

	INN with DSP	INN without DSP	Floating-Point
Net Calculation	**21cc**	87cc	776cc
One Neuron	**49cc**	115cc	923cc
Whole Sample Set	**717cc**	1509cc	11996cc

fastest method in all of the cases. The DSP operations allow the Net value to be calculated four times faster than would be possible without DSP operations. For the calculation of the whole sample set, the speed increase provided by the DSP operations has been reduced because the Net calculation accounts for only a small portion of the overall calculations. The integer calculations without DSP operations also perform quite well. While slower than the DSP version, it is still significantly faster than the floating-point version. The last tested version was the floating-point version. As expected the lack of floating-point hardware greatly degrades the performance of this network. The integer versions are faster by a factor of 7.9 without DSP acceleration and by a factor of 16.7 with DSP operations when compared to the floating-point method. These results very strongly supports the case for using both integers and DSP operations on low cost hardware.

4.2.3 Effects Of Network Size

Table 4.4 shows the percentage of the whole sample set a single net calculation occupies. The results of Table 4.4 show that relative to the INN without DSP and Floating-Point, the INN with DSP spends a significantly smaller portion of the total execution time calculating

Table 4.4: Net Calculation vs. Whole Sample Set Percentage.

	INN with DSP	INN Without DSP	Floating-Point
Ratio %	2.93	5.765	6.47

net values. This is an important consideration because as the network size increases, the number of inter-neuron connections will greatly increase. The small size of this network means that there are few inter-neuron connections, and as a results fewer components to the net calculation. However for larger networks the percentage of the time taken to calculate the Net values will increase. This means that the larger the network is, the more incentive there is to use DSP operations, as they have been shown to be effective in reducing the Net Calculation time. Despite the impressive performance of the INN with DSP, these results should be considered conservative because this network is so small. On larger networks the performance gap between the INN with DSP and the other two methods will increase relative to the number of inter-neuron connections.

4.3 Discussion

4.3.1 Performance Bias Considerations

It is important to note that the activation function is not actually calculated to remove any testing bias against the floating-point version. A full implementation that calculates the activation function will require many more clock cycles. This in combination with the small network size, which biases the results against the INN with DSP, results in a performance comparison that leaves little doubt about the superiority of the INN with DSP method. Despite operating at an unavoidable disadvantage, it remains the most capable in terms of execution speed of the three methods. The INN without DSP operations also runs at a disadvantage relative to the floating-point method, but remains significantly faster. The floating-point would perform much better on hardware that supports floating-point operations, but on low cost chips the performance is significantly less than that of integer only neural networks.

4.3.2 Integer Neural Networks Practicality

These results creates a strong case for the practicality of low cost integer neural networks in cases where the both cost and execution time are important factors. Especially important is the performance of the integer neural networks when compared to the floating-point network. The results clearly show the performance improvement in using integer neural networks. Where execution time is not critical, a low cost microcontroller can be used to implement a floating-point neural network. But for situations where execution time is critical integer neural networks are a faster option. Due to the complexitites of neural networks, and it is impossible to state exactly how the use if integer neural networks will effect accuracy, execution time, and power consumption. However it can be claimed in general, that the floating point method offers the most accuracy, in exchange for the highest cost, and worst execution time. While INN with DSP offers the best execution time at a mid range cost, with reduced accuracy. The INN without DSP offers the lowest cost solution, while having a mid range execution time, and reduced accuracy. Finally it should be noted that each application is different and it is entirely possible that low cost microcontrollers may not be suitable. In these circumstances the only solution is to use microcontrollers with floating-point DSP operations to increase accuracy, or to use FPGAs for especially large networks. Both of these options are more expensive and consume more power, but may be the only solution available.

Chapter 5

Accuracy Comparison

With Chapter 4 demonstrating the performance improvement it is now necessary to investigate the accuracy lost due to quantization. It is imperative to demonstrate that integer neural networks can provide accurate results. While it is known that integer neural networks can never be as accurate as floating-point networks [40], it is possible to create integer networks that are accurate enough to meet the requirements of the application.

5.1 Test Setup

5.1.1 Integer Constraints

The second major consideration when choosing to use an integer neural network is the accuracy of the network. All of the values in an integer neural network must be whole numbers. For this reason all of the values in the integer neural network must fit into a limited range of predefined values. The range of values affects the accuracy, as well as the amount of memory used by the network. A larger range of values will be more accurate, but also require more memory to store values and LUTs. To explore the effects of quantization we will use the following example system.

5.1.2 Real World Test Case

The goal of this system is to determine the temperature inside a building (T_{avg}), using external measurements [51]. The external measurements used are the outside temperature

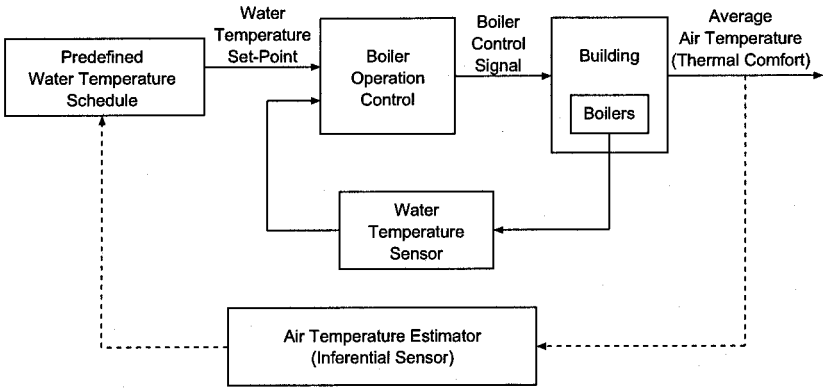

Figure 5.1: Closed Loop Boiler Control Scheme

(T_0), the amount of solar radiation (Q_{sol}), and the energy consumed by the boiler (Q_{in}) [52]. The purpose of this application is to replace many expensive temperature monitoring points with just three sensors. This information can then be used to create a closed loop boiler control scheme as seen in Fig 5.1. By being able to cheaply estimate the average temperature T_{avg}, the control system can more economically maintain a constant temperature. All of the data was collected a special testing facility [53] so that the actual T_{avg} is available for comparison. A 3x20x3x1 network, shown in Fig. 5.2, is then trained using backpropagation and the Levenberg–Marquardt method of optimization. This creates a base network from which the integer network is derived and compared.

5.1.3 Network Topology

The network is trained as an floating-point network. The Levenberg–Marquardt (5.1) method of optimization is used to train feedforward networks of various topologies.

$$S(\beta) = \sum_{i=1}^{m} [y_i - f(x_i, \beta)]^2 \tag{5.1}$$

The hyperbolic tan (tanh) was again used as the activation function for the network, and RMSE was used for the error function. Table 5.1 shows the performance (averaged across three runs) of each network topology when trained with the whole data set. These resulting network are not usable because when the network is trained with the whole data set it will

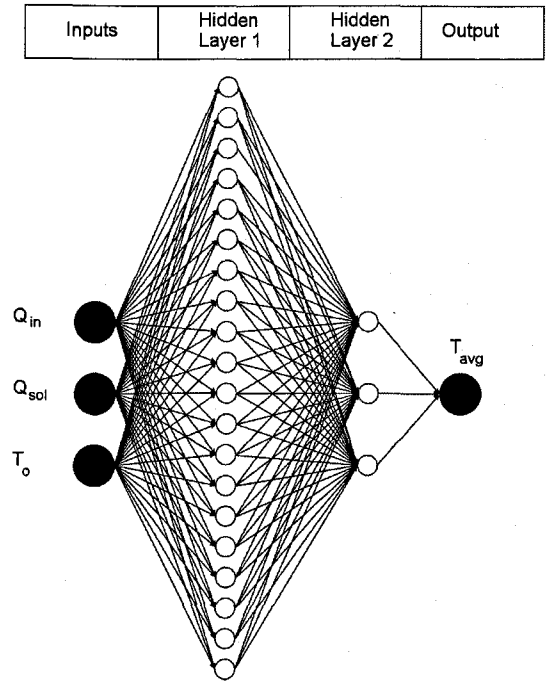

Figure 5.2: A 3x20x3x1 neural network.

become over-trained to the specific data set and not extract the underlying relationships. However this does provide a maximum level of performance a topology is capable of. Due to the complexity of the input output relationships it was judged that a single hidden layer network would have poor performance and this was born out during a brief test. Two hidden layer networks were believed to be the most appropriate and in testing were shown to have the best performance. According to [54] and [55] networks beyond two hidden layer rarely show improvement and a brief testing showed this to be true in this case as well. Two layer networks were significantly better The best performing topology (shown in bold in Table

Table 5.1: Network Topologies

Layer 1 \ Layer 2	1	3	5	10	20
5	0.654	0.648	0.597	0.625	0.608
10	0.636	0.617	0.599	0.628	0.611
20	0.669	**0.568**	0.585	0.673	0.683
50	0.785	0.700	0.658	0.795	0.700

5.1) is used as the base network topology. The network was then trained with approximately one eighth of the total data set. This was necessary to include a full day cycle and part of the weekend plateau. The network was tested with various sections of the data set to determine which section would provide a good generalized sample. Additionally a small range of training cutoffs to increase generalization, setting the Root Mean Squared Error (RMSE) cutoff to 0.60 °C proved most effective. From these tests the resulting network with the lowest RMSE and the smallest maximum error across the whole data set was selected as the base network.

5.2 INN Model

5.2.1 Scaling And Quantization

All of the training data was collected is normalized to values between -1 and +1 before use to maintain a consistency between the floating-point and integer networks, as well as all of the scaled version of the integer network, using the mapminmax function (5.2). The pre-scaling can be reversed be re-arranging the mapminmax function to isolate for the original inputs,

$$y = \frac{(y_{max} - y_{min})(x - x_{min})}{x_{max} - x_{min}} + y_{min} \tag{5.2}$$

$$y - y_{min} = \frac{(y_{max} - y_{min})(x - x_{min})}{x_{max} - x_{min}} \tag{5.3}$$

$$(y - y_{min})(x_{max} - x_{min}) = (y_{max} - y_{min})(x - x_{min}) \tag{5.4}$$

$$(x - x_{min}) = \frac{(y - y_{min})(x_{max} - x_{min})}{y_{max} - y_{min}} \tag{5.5}$$

$$x = \frac{(y - y_{min})(x_{max} - x_{min})}{y_{max} - y_{min}} + x_{min} \tag{5.6}$$

To create an integer neural network from this, all of the input and weights of the network are multiplied by a scaling factor S_f and then quantized into whole values. By scaling and quantizing each part of the network an integer neural network model is created. All values are adjusted to this new scale before being used in any operations. For the activation function the scaling is reversed, bringing the value back into the normal range. The activation function, in this case tanh, is then applied. The values are then rescaled and quantized for

use in the rest of the network. The reason for doing this is to maintain a consistent network structure, only the value of S_f is changed for each level of quantization. This allows for many different levels of quantization to be tested with the same network while at the same time ensuring that model is accurate. In an actual implementation, the activation function would be replaced with a LUT for the chosen scaling factor.

5.2.2 Mathematical Model

Mathematically the scaling and quantizing are substituted into the normal function of a neuron, where O is the output, I is the input vector to that neuron, W is the weight vector, and B is the bias. Equation (5.7) shows the output for a neuron in a normal neural network. The scaling factor S_f is applied to each of the values used by the network before they are used. The resulting value is then quantized to a whole value as seen in,

$$O = \tanh(I \cdot W + B) \tag{5.7}$$

$$O = \tanh(\frac{q(I \cdot S_f) \cdot q(W \cdot S_f) + q(B \cdot S_f^2)}{S_f^2}) \tag{5.8}$$

The scaling factor for the bias must be S_f^2 to maintain proportionality to the rest of the equation as shown by the simplification in equations the equations below,

$$O = \tanh(\frac{I \cdot S_f \cdot W \cdot S_f + B \cdot S_f^2}{S_f^2}) \tag{5.9}$$

$$O = \tanh(\frac{I \cdot W \cdot S_f^2 + B \cdot S_f^2}{S_f^2}) \tag{5.10}$$

$$O = \tanh(I \cdot W \cdot + B) \tag{5.11}$$

Once the net value has been calculated the scaling is removed by dividing the total amount of scaling, S_f^2. As can be seen in (5.11), canceling out the S_f^2 term will return the equation to its original form, proving that the transformation is correct. This is then used by the activation function. The output from the activation function is used as the input for the next layer of the network, at this point the scaling and quantization is reapplied because the output, O, of this layer becomes the input, I, for the next layer. The result is that the input

first hidden layer are the inputs to the network, called the traditionally called the output of the input layer O_{IL} to maintain the idea of one layer feeding the next. These inputs are passed through first hidden layer to produce an output O_{HL1},

$$O_{HL1} = \tanh(\frac{q(O_{IL} \cdot S_f) \cdot q(W \cdot S_f) + q(B \cdot S_f^2)}{S_f^2}) \tag{5.12}$$

This output is now in the range -1 to +1 as this is the limit of the tanh function. This means that the range of values is the same for both a floating-point and integer networks. However the integer networks will have experienced quantization. Being able to easily compare the effects of quantization at each layer is very useful in troubleshooting the network. It provides a very simple way to see the effects of quantization at each stage and potentially identify where the effects of quantization are most severe. The output O_{HL1} is then used as the input for the second hidden layer. At this point the output is scaled again by S_f so that it is in the scaled range of values. The downscaling before the tanh function, and upscaling afterwords allows for the model to test many values of S_f without changing the activation function. The output of the second hidden layer O_{HL2},

$$O_{HL2} = \tanh(\frac{q(O_{HL1} \cdot S_f) \cdot q(W \cdot S_f) + q(B \cdot S_f^2)}{S_f^2}) \tag{5.13}$$

Is then used as the input a single neuron at the output layer.

$$O_{OL} = \tanh(\frac{q(O_{HL2} \cdot S_f) \cdot q(W \cdot S_f) + q(B \cdot S_f^2)}{S_f^2}) \tag{5.14}$$

Because the network has only one output, the estimated average temperature $T_{avg_{est}}$, only one neuron is needed. The output from this final layer, O_{OL}, is the final output of the network. This is the direct output from the output layers tanh function, and so is in the range of -1 to +1. These values are feed through the reversed mapminmax function derived in (5.6) using the same constants as the input to return the values to proper temperature values,

$$T_{avg_{est}} = \frac{(O_{OL} - T_{avg_{min}})((1) - (-1))}{T_{avg_{max}} - T_{avg_{min}}} + (-1) \tag{5.15}$$

This produces the estimated average temperature $T_{avg_{est}}$ which can then be compared to the measured T_{avg} to determine the accuracy of the network, and judge the effects of quantization.

5.3 Results

5.3.1 Base Network Accuracy

The results of the trained base network is shown in Fig. 5.3. This is the base accuracy from which the integer versions of this network can be compared. Below are the outputs of the integer neural network with different values for S_f. This shows the effects of varying levels of quantization. Also included is the is the network error, this is the difference between the measured temperature and the estimated temperature. Fig. 5.3 shows the measured T_{avg}

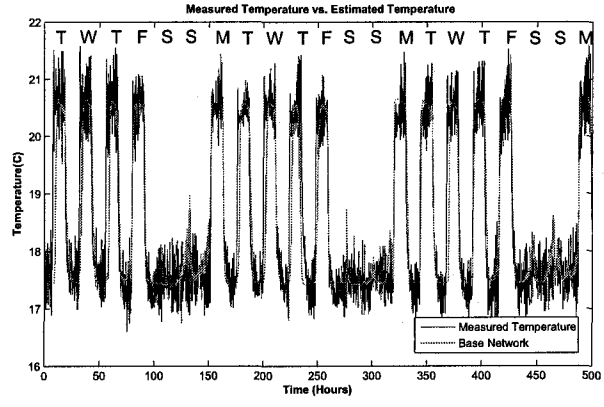

Figure 5.3: Output of the base neural network vs. Measured Temperature.

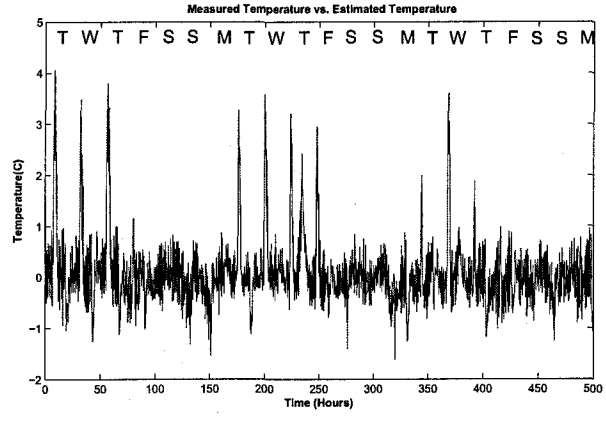

Figure 5.4: Error of the base neural network.

and the output of the base network. While this network contains some inaccuracies it is the

relative performance between this network and the quantized integer neural networks that we are interested in.

5.3.2 INN At High Values Of S_f

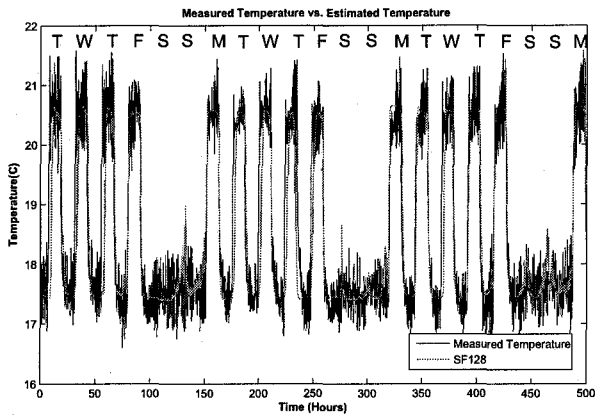

Figure 5.5: Output of an integer neural network with $S_f = 128$ vs. Measured Temperature.

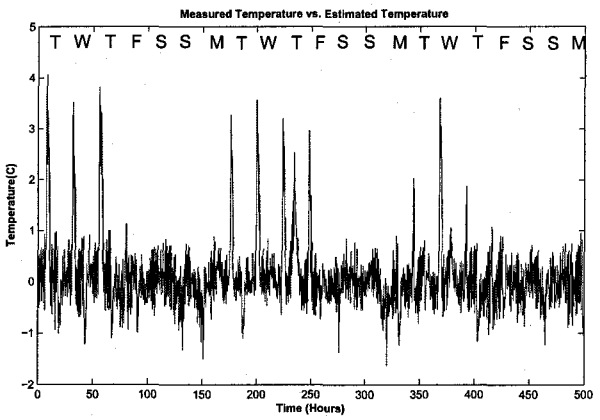

Figure 5.6: Error of an integer neural network with $S_f = 128$.

Fig. 5.5 and Fig. 5.7 show the network outputs when S_f is equal to 128 and 64 respectively. At this level of quantization, the integer and base networks are almost identical.

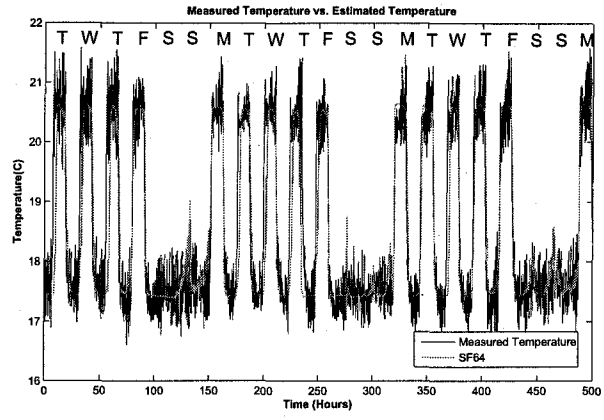

Figure 5.7: Output of an integer neural network with $S_f = 64$ vs. Measured Temperature.

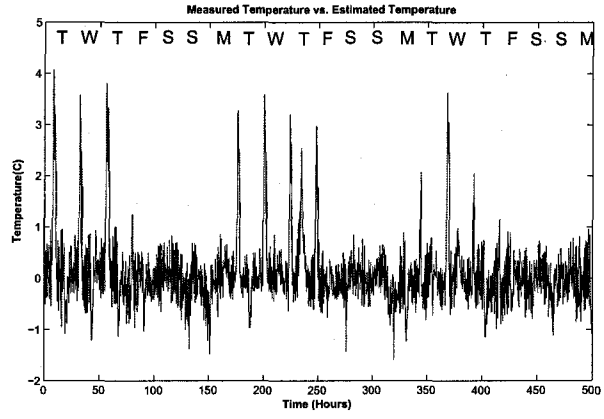

Figure 5.8: Error of an integer neural network with $S_f = 64$.

5.3.3 LUT Considerations

However memory usage is significant as the range of input values for the LUT are from $-S_f \cdot 2$ to $+S_f \cdot 2$. This results in 32 768 addresses and 8 192 addresses for the LUT for $S_f = 128$ and $S_f = 64$ respectively. On a low cost microcontroller, this may exceed the amount of available RAM. However some microcontrollers will also allow LUTs to be stored in the program memory. In these cases, this is not an issue.

5.3.4 INN At Mid Range Values of S_f

Fig. 5.9 and Fig. 5.11 show the network output for S_f of 8 and 4 respectively. Here the effects of quantization become much more pronounced and the accuracy of the networks degrades. At this level of quantization the network is clearly not as accurate as the base network. This

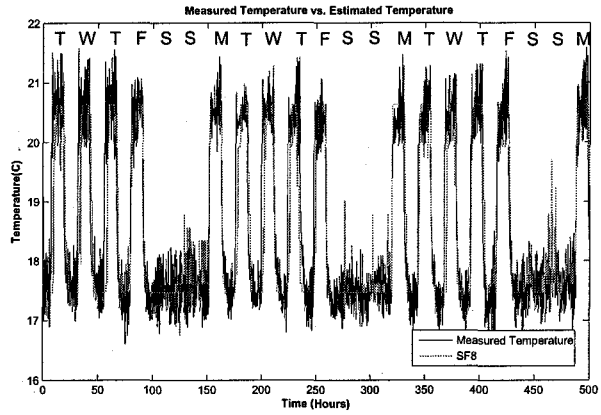

Figure 5.9: Output of an integer neural network with $S_f = 8$ vs. Measured Temperature.

level of accuracy is the minimum that can be accepted as offering a comparable level of accuracy.

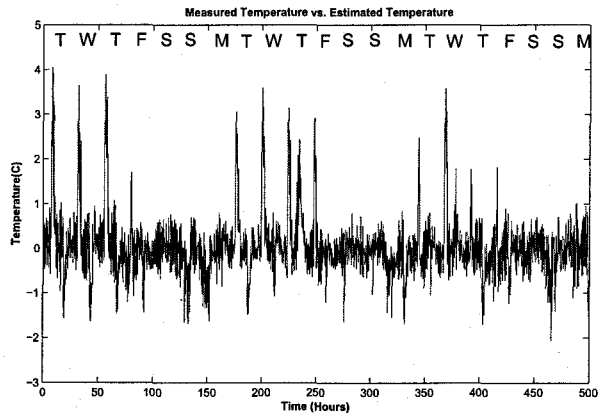

Figure 5.10: Error of an integer neural network with $S_f = 8$.

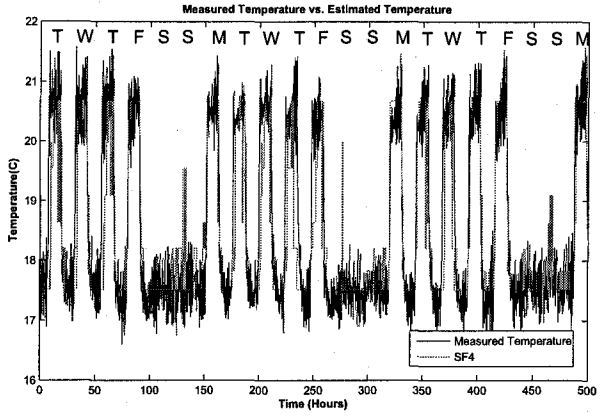

Figure 5.11: Output of an integer neural network with $S_f = 4$ vs. Measured Temperature.

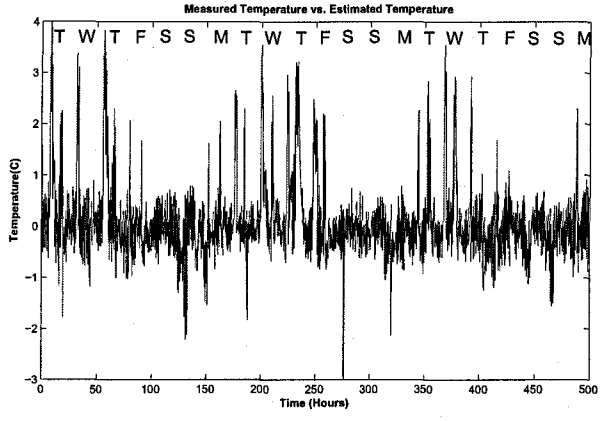

Figure 5.12: Error of an integer neural network with $S_f = 4$.

5.3.5 INN At Low Values of S_f

Fig. 5.13 shows the network output at S_f equal to 2. Here the network is very inaccurate, the effects of quantization have degraded the network to such an extent that it is unusable.

5.3.6 Statistical Comparison

Table 5.2 shows a comparison of the accuracy of the networks at each level of quantization, using three types of statistical measures; the Root Mean Squared Error (RMSE), the correlation of determination (R^2), and the Sum Squared Error (SSE). As can be seen from Fig.

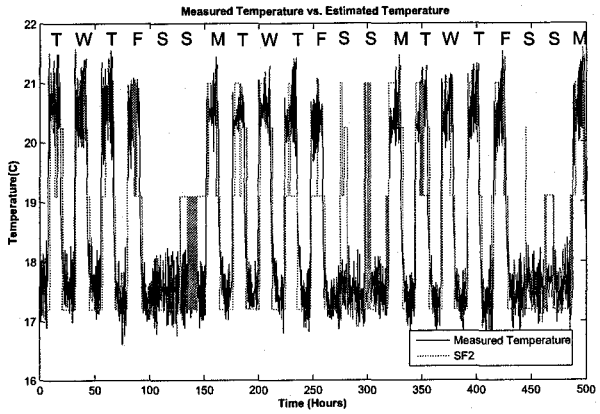

Figure 5.13: Output of an integer neural network with $S_f = 2$ vs. Measured Temperature.

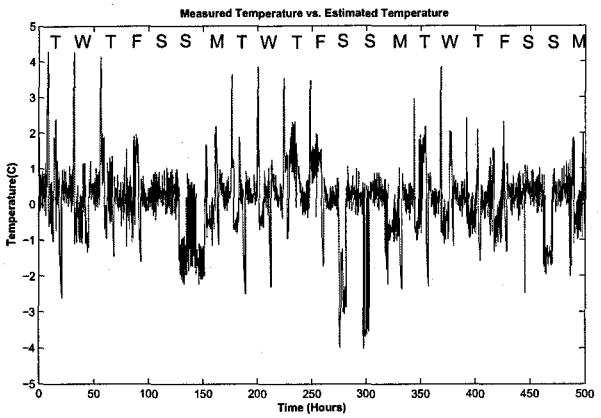

Figure 5.14: Error of an integer neural network with $S_f = 2$.

Table 5.2: Accuracy comparison for selected levels of quantization.

Network	RMSE (°C)	R^2	SSE (°C)
Base Network	0.60	0.7944	2724
$S_f = 128$	0.60	0.7942	2423
$S_f = 64$	0.60	0.7947	2729
$S_f = 8$	0.63	0.7779	2955
$S_f = 4$	0.66	0.7331	3281
$S_f = 2$	0.95	0.6058	6770

5.3 to Fig. 5.13 and Table 5.2, when the value of S_f is high the integer neural networks performance is nearly identical to that of the base network. However as the rate of quantization

increases, the accuracy of the network gradually degrades. These results are comparable to performance of a physical model developed in [56] which achieved a RMSE of 0.54 °C, and the work in [52] which improved this performance to 0.22 °C. The base network tested here is only marginally less accurate then these attempts while being much simpler to implement in a low cost embedded system.

5.3.7 Example Subjectivity

The graceful degradation seen in this example will not be true for all systems. The network may degrade faster or slower but it may also hit a point where the level of quantization is too much and the network accuracy degrades very quickly. The key consideration for the level of quantization is the trade-off between network accuracy, memory consumption, and register size. The larger the value of S_f, the smaller the amount quantization, the more accurate the network will be. However, at the same time this will increase the amount of memory needed to store the weights, inputs and LUTs.

5.3.8 Practical Consideration

Additionally if the quantization results in weights or inputs larger than can be stored in one register on the microcontroller, the network will suffer an additional performance penalty. This can aid in the selection of the type of microcontroller to use. If the level of quantization allows for the inputs and weights to be stored in an 8 bit register, then a very low cost microcontroller can be used. However if the values are larger, then a 16 bit or possibly even 32 bit microcontroller would be better suited to implementing the network.

5.4 Discussion

5.4.1 Trade-offs

Almost all system designs involve trade-offs between performance, speed, and cost. In this chapter, we have explored integer neural networks as a method of reducing the cost of a system, while attempting to retain as much of the accuracy of a floating-point neural

network. It is important to note that integer neural networks are just one possible method of meeting design constraints. Integer neural networks excel on low cost microcontrollers. However integer neural networks can never achieve the level of accuracy that floating-point networks can.

5.4.2 Network And Hardware Choice

For this reason the decision on which type of network to use is depended on the type of data being used and must be viewed in light of the design goals for the system. In addition, while in this example the accuracy of the network degraded gracefully, this will not be the case for all implementations. The level of quality depredation is dependent on the application and the base network. It is entirely possible that for certain networks INNs are not practical. However when cost is the primary design concern, integer neural networks offer a very compelling combination of accuracy, speed, and cost that can reduce the hardware costs to implement an effective neural network.

Chapter 6

Conclusion

This thesis has been very successful in achieving our goals. It has proven the execution speed advantages of integer neural networks on low cost DSPs. It then demonstrated the effects of quantization at different scaling factors, and provided a convenient method for converting floating-point neural networks to integer neural networks. The results from these tests provide a very strong case for using integer neural networks on low cost DSPs and microcontrollers.

6.1 Review Of Purpose

The purpose of this thesis is to investigate neural networks for low cost systems. This investigation centered around the use of integer neural networks. These networks have the ability to run natively on very low cost integer only microcontrollers. Integer only microcontrollers are cheaper and consume less power due to the fact that integer calculations are simpler than floating-point calculations, and therefore require less complicated hardware. With these facts at hand two important aspects about the feasibility of using integer neural networks was investigated. First, an investigation of the hypothesized performance improvement on low cost integer only microcontrollers, and the possible performance enhancement offered by DSP operations. Second, a case study of the effects of quantization on neural networks, the effects of different scaling rates, and a method for creating an integer neural network with a variable scaling rate.

6.2 Performance

In Chapter 4 the contrast between the performance of integer and floating-point networks was investigated in detail. The results show that integer neural network do indeed offer greatly increased performance. In all cases the floating-point network was by far the worst performer of the three. These tests also demonstrated the effectiveness of DSP operations, which proved to have the best performance in all cases. By using a worst case network, and biasing the results against the desired conclusion, it was demonstrated that a performance improvement by a factor of ten or greater is entirely possible. This provides a very strong case for using integer neural networks in cases where execution time is critical. In addition, the significantly shorter execution time supports the case for using integer neural networks in conditions where low power consumption is desirable, such as portable or battery operated equipment.

6.3 Accuracy

In Chapter 5 the effect of quantization inherent in integer neural networks was investigated. A real world test case was used to compare the effects of quantization. The test case required a neural network was trained to be an inferential sensor for use in building heating control schemes. This network was then converted into an integer neural network with a variable scaling rate. Having a variable scaling rate allowed the network to be compared at different levels of quantization and showed that with a large scaling factor the results are nearly indistinguishable from a floating-point network, and that at medium levels of scaling the results are still usable. This testing also showed that the quality of the output does not degrade in a linear fashion. These results provide strong support for the use of integer neural networks by showing that they are capable of accurate results.

6.4 Final Remarks

The research conducted in this thesis has been very successful in demonstrating the utility of using low cost microcontrollers to execute integer neural networks. Some of the results of the this thesis include:

- The combined use of these low cost processors, both with and without DSP operations, when used in conjunction with integer neural networks reduces the entry cost into embedded neural networks. These networks can offer greatly reduced cost due to the less complicated chip design.
- The reduced circuit complexity can also reduce power consumption, which can be very important for low power and portable applications.
- A reduction in power consumption can reduce operating costs when compared with floating point networks implemented on more expensive floating-point capable hardware.
- For a modest increase in cost, integer only microcontrollers with DSP operations are available. The DSP operations greatly increase the rate of execution for integer neural networks, as demonstrated in Chapter 4.
- The use of integer only neural networks on low cost microcontrollers offer a significant increase in performance over floating-point networks on micrcontorollers without hardware support for floating-point operations.
- The DSP operations allow integer neural networks to compete with floating-point DSP hardware which is commonly used for embedded neural networks.
- The results from Chapter 5 show that not only are integer neural networks capable of fast execution, they can also be very accurate.
- A method of training an integer neural network completely without floating-point values was also presented, however the method suffers from some serious drawbacks.

- No integer neural network can be as accurate as a floating-point version. However in the examined case in Chapter 5 the accuracy of the base network was the primary factor in network accuracy, not the effects of quantization due to the use on integers. While this may not be the case for all scenarios, in example scaling factors as low as 8 offered good accuracy.

- A method for training a neural network in a conventional method with conventional methods and then converting to an integer neural network was presented in Chapter 5. This allows for much easier training of integer neural networks when the network can be trained offline.

- The conversion method of creating integer neural networks presented in Chapter 5 allows for varying levels of quantization to be compared with the original floating-point network. This method also allows for each layer of the network to be easy contrasted with the base network to assist in troubleshooting.

- The trade-offs between execution speed, accuracy, power consumption, and cost are subjective to the processors being considered. However it can be said that in general, that it is possible to use integer neural networks on low cost microcontrollers to decrease cost and power consumption in exchange for a decrease in accuracy.

- The fast execution speed, low cost, and good accuracy, combine to provide a compelling case for the use of integer neural networks on low cost integer only DSPs for embedded systems.

The research conducted in this thesis has been successful in providing a case for low cost embedded neural networks. The combined use of low cost integer only microcontrollers, both with and without DSP operations, in conjunction with integer neural networks, offers the potential for embedded neural networks with lower cost and power consumption then is currently being considered. These results are practical, and have immediate applications in the real world.

6.5 Future Work

This area of research is just beginning, and there exists many further avenues of investigation. Some of the areas for future research include:

- More test case investigations. The full investigation of the effects of quantization has just begun, the results will likely differ with each case. A battery of tests is needed to fully ascertain the quantization effects.

- Comparisons of execution times across networks with varying sizes. Chapter 4 investigated a small simple network, where DSP performance is at a minimum relative to the other implementations. While clear conclusions can be drawn from these results, an investigation into the performance of larger networks may reveal new information.

- An investigation networks of different architectures. The feedforward network was chosen for these tests for its simplicity, and directness. Using a feedforward network minimized the number of variables in network training, performance, and accuracy. Attempts to convert other types of network architectures discussed in Chapter 3 such as recurrent networks can also be investigated. The higher rate of inter neuron connectivity suggest that some of these network may exact greater performance gains from DSP operations. However the effects of quantization will likely effect these networks differently, so their utility is still in question.

- Creating neural networks with integer only backpropagation can also be explored. The method utilized in Chapter 3 has many drawbacks that can be imported upon. Additionally the execution time for training a neural network on an low cost microcontroller could also be an area of interest for applications that demand online learning and adaption.

- Performance comparisons of neural networks that are trained online (while in use). As discussed in Chapter 3 training a neural network with only integers is possible but has great difficulty reaching an optimum. Floating-point training methods are much faster

and more reliable. However because of the performance increase afforded by using only integers, this method can make many training attempts in the time a floating-point method can make one attempt. It is not at all clear which method is superior or how different variables such as optimization method, activation function, network size, etc. Effect the time to train a network on a low cost microcontroller.

- Comparisons across a range of processors. It is known that processors with simpler circuitry general consume less power, the extant it unknown. The comparison between processors of different designs may yield more information concerning execution time and power consumption. The dsPIC used in these tests uses a modified Harvard architecture, while desktop computers usually use von Neumann architecture. The use of caching is also completely unexplored in this thesis. There are a great many considerations in the design of a processor and it is likely that some will be better suited to low cost neural networks then others.

- Power, and performance comparisons can be made with neural networks embedded in FPGAs. While it is apparent the FPGAs have a clear advantage in terms of performance, FPGAs also are at a clear disadvantage in power consumption. For low power applications it is possible that an integer neural network on a low cost DSP may provide a superior set of trade-offs then currently exist.

Now that lower cost embedded systems with neural networks have shown to be practical, the full range and extent of their capabilities are open to further research.

Appendix A

Abbreviation List

ANN	Artificial Neural Network
CC	Clock Cycle
DSP	Digital Signal Processor
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
IIR	Infinite Impulse Response
INN	Integer Neural Network
LUT	Lookup Table
MAC	Multiply ACcumulate
PC	Personal Computer
RMSE	Root Mean Squared Error
SSE	Sum Squared Error

Appendix B

Microcontroller Cost Comparison

This appendix contrasts the prices for a variety of processors. All of the prices were obtained from the Digikey Corporation, a leader in the North American electronics retail and warehousing. All of the prices listed are in Canadian Dollars. Due to the wide variety of chip packaging and features, the highest and lowest costs are listed. This provides are more accurate depiction of the price range for these processors then any chip type or average could. Table B.1 shows the various costs and capabilities of the processors.

Table B.1: Comparison of cost and functionality of various processors

Model	Core	Type	DSP	Floating-Point	Lowest Price	Hightest Price
PIC10F220	PIC	8bit	No	No	$ 0.48	$ 0.82
PIC18F1220	PIC	8bit	No	No	$ 2.68	$ 4.75
MSP430F1101	MSP430	16bit	No	No	$ 1.45	$ 2.80
MSP430F5438	MSP430	16bit	No	No	$ 7.13	$ 11.41
[1]dsPIC30f2011	dsPIC	16bit	Yes	No	$ 3.05	$ 5.10
dsPIC30f4016	dsPIC	16bit	Yes	No	$ 9.78	$ 29.97
TMS320C6711	TMS320	32bit	Yes	Yes	$ 26.46	$ 28.58
OMAP5910	ARM9	32bit	Yes	Yes	$ 38.38	$ 44.81

[1]Processor used for testing.

References

[1] K. Wong, "Artificial intelligence and neural network applications in power systems," in *Advances in Power System Control, Operation and Management, 1993. APSCOM-93., 2nd International Conference on*, Dec 1993, pp. 37–46 vol.1.

[2] M. Schlang, T. Poppe, and O. Gramchow, "Neural networks for steel manufacturing," *IEEE Expert*, vol. 11, no. 4, pp. 8–9, Aug 1996.

[3] D. Casasent, "Optical processing in neural networks," *IEEE Expert*, vol. 7, no. 5, pp. 55–61, Oct 1992.

[4] Z. Xiong, "A modified adaptive genetic bp neural network with application to financial distress analysis," in *Genetic and Evolutionary Computing, 2008. WGEC '08. Second International Conference on*, Sept. 2008, pp. 149–152.

[5] X. Yao, M. Fischer, and G. Brown, "Neural network ensembles and their application to traffic flow prediction in telecommunications networks," in *Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on*, 2001, vol. 1, pp. 693–698 vol.1.

[6] C. Lin, "Application of neural network in control problem," in *Machine Learning and Cybernetics, 2007 International Conference on*, Aug. 2007, vol. 6, pp. 3465–3471.

[7] J. Tang, M. Varley, and M. Peak, "Hardware implementations of multi-layer feedforward neural networks and error backpropagation using 8-bit pic microcontrollers," in *Neural*

and Fuzzy Systems: Design, Hardware and Applications (Digest No: 1997/133), IEE Colloquium on, 1997, pp. 2/1–2/5.

[8] J Zhu and P Sutton, "Fpga implementations of neural networks a survey of a decade of progress," in *Proceedings. of the 13th International Conference on Field-Programmable Logic and Applications*, 2003, pp. 1062–1066.

[9] G. Rovithakis, M. Maniadakis, and M. Zervakis, "A hybrid neural network/genetic algorithm approach to optimizing feature extraction for signal classification," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, no. 1, pp. 695–703, Feb. 2004.

[10] Z. Yuanhui, L. Yuchang, and S. Chunyi, "Combining neural network, genetic algorithm and symbolic learning approach to discover knowledge from databases," in *Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on*, Oct 1997, vol. 5, pp. 4388–4393 vol.5.

[11] P. Georgilakis, N. Doulamis, A. Doulamis, N. Hatziargyriou, and S. Kollias, "A novel iron loss reduction technique for distribution transformers based on a combined genetic algorithm - neural network approach," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 31, no. 1, pp. 16–34, Feb 2001.

[12] Y. Jiang, "Sub-optimal multiuser detector using a wavelet chaotic simulated annealing neural network," in *Natural Computation, 2008. ICNC '08. Fourth International Conference on*, Oct. 2008, vol. 3, pp. 358–362.

[13] V. Kroumov and J. Yu, "3d path planning for mobile robots using annealing neural network," in *Networking, Sensing and Control, 2009. ICNSC '09. International Conference on*, March 2009, pp. 130–135.

[14] C. Xiu and Y. Li, "Bp neural network based on chaos simulated annealing," in *Neural Networks and Brain, 2005. ICNNB '05. International Conference on*, Oct. 2005, vol. 1, pp. 358–360.

[15] B. Wilamowski, "Neuro-fuzzy systems and their applications," in *Industrial Electronics Society, 1998. IECON '98. Proceedings of the 24th Annual Conference of the IEEE*, Aug-4 Sep 1998, vol. 1, pp. T35–T49 vol.1.

[16] L. Arafeh, H. Singh, and S. Putatunda, "A neuro fuzzy logic approach to material processing," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 29, no. 3, pp. 362–370, Aug 1999.

[17] E. Fonseca, P. Vellasco, M. Vellasco, and S. de Andrade, "A neuro-fuzzy system for steel beams patch load prediction," in *Hybrid Intelligent Systems, 2005. HIS '05. Fifth International Conference on*, Nov. 2005, pp. 6 pp.–.

[18] G. Puskorius and L. Feldkamp, "Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 279–297, Mar 1994.

[19] L. Zhang, S. Quan, and K. Xiang, "Recurrent neural network optimization for model predictive control," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, June 2008, pp. 751–757.

[20] H. Song, S. Kang, and S. Lee, "A new recurrent neural network architecture for pattern recognition," in *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, Aug 1996, vol. 4, pp. 718–722 vol.4.

[21] Y. Kuwahara and T. Matsumoto, "Experiments on direction finder using rbf neural network with post-processing," *Electronics Letters*, vol. 41, no. 10, pp. 602–603, May 2005.

[22] C. Chang and S. Fu, "Image classification using a module rbf neural network," in *Innovative Computing, Information and Control, 2006. ICICIC '06. First International Conference on*, 30 2006-Sept. 1 2006, vol. 2, pp. 270–273.

[23] Q. Xiong, K. Hirasawa, J. Hu, and J. Murata, "Growing rbf structures using self-organizing maps," in *Robot and Human Interactive Communication, 2000. RO-MAN 2000. Proceedings. 9th IEEE International Workshop on*, 2000, pp. 107–111.

[24] P. Wang, H. Huang, and X. Zhang, "Hopfield network based optimization of mechanical design," in *Neural Networks and Brain, 2005. ICNN B '05. International Conference on*, Oct. 2005, vol. 3, pp. 1766–1769.

[25] C. Tsai and Y. Sun, "Minimizing the energy of active contour by using a hopfield network," in *Systems Engineering, 1992., IEEE International Conference on*, Sep 1992, pp. 495–498.

[26] Y. Yoshida, W. Benjapolakul, and A. Iwata, "An application of hopfield network to discrete fourier transform," in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, Jul 1991, vol. ii, pp. 890 vol.2–.

[27] O. Obst, X. Wang, and M. Prokopenko, "Using echo state networks for anomaly detection in underground coal mines," in *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, April 2008, pp. 219–229.

[28] Q. Ge and C. Wei, "Multiresolution-based echo state network and its application to the long-term prediction of network traffic," in *Computational Intelligence and Design, 2008. ISCID '08. International Symposium on*, Oct. 2008, vol. 1, pp. 469–472.

[29] E. Mathews and A. Poigne, "An echo state network based pedestrian counting system using wireless sensor networks," in *Intelligent Solutions in Embedded Systems, 2008 International Workshop on*, July 2008, pp. 1–14.

[30] J. Chen, G. Xi, Y. Xing, J. Wang, and C. Zheng, "An unsupervised multi-valued stochastic neural network algorithm to cluster in coronary heart disease data," in *Convergence Information Technology, 2007. International Conference on*, Nov. 2007, pp. 640–644.

[31] B. Cui and X. Lou, "Robust stability analysis of neutral stochastic neural networks with delay: An lmi approach," in *Control and Automation, 2007. ICCA 2007. IEEE International Conference on*, 30 2007-June 1 2007, pp. 117–122.

[32] S. Wang, S. Wang, G. Li, and Z. Gao, "Robust asymptotical stability for uncertain stochastic neural networks with discrete and distributed delays," in *Machine Learning and Cybernetics, 2008 International Conference on*, July 2008, vol. 2, pp. 815–819.

[33] A. Namatame and Y. Tsukamoto, "Composite neural network models and their application," in *Neural Networks, 1993. IJCNN '93-Nagoya. Proceedings of 1993 International Joint Conference on*, Oct. 1993, vol. 1, pp. 738–741 vol.1.

[34] M. Su, W. Jean, and H. Chang, "A static hand gesture recognition system using a composite neural network," in *Fuzzy Systems, 1996., Proceedings of the Fifth IEEE International Conference on*, Sep 1996, vol. 2, pp. 786–792 vol.2.

[35] M. Su, C. Kao, K. Liu, and C. Liu, "Rule extraction using a novel class of fuzzy degraded hyperellipsoidal composite neural networks," in *Fuzzy Systems, 1995. International Joint Conference of the Fourth IEEE International Conference on Fuzzy Systems and The Second International Fuzzy Engineering Symposium., Proceedings of 1995 IEEE International Conference on*, Mar 1995, vol. 1, pp. 233–238 vol.1.

[36] S. Rizvi and N. Nasrabadi, "Residual vector quantization using a multilayer competitive neural network," *Selected Areas in Communications, IEEE Journal on*, vol. 12, no. 9, pp. 1452–1459, Dec 1994.

[37] R. Allan and W. Kinsner, "A study of microscopic images of human breast disease using competitive neural networks," in *Electrical and Computer Engineering, 2001. Canadian Conference on*, 2001, vol. 1, pp. 289–293 vol.1.

[38] P. Engel and R. Molz, "A new proposal for implementation of competitive neural networks in analog hardware," in *Neural Networks, 1998. Proceedings. Vth Brazilian Symposium on*, Dec 1998, pp. 186–191.

[39] S. Draghici, "Some new results on the capabilities of integer weights neural networks in classification problems," in *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, 1999, vol. 1, pp. 519–524.

[40] A. Khan and E. Hines, "Integer-weight neural nets," *Electronics Letters*, vol. 30, no. 15, pp. 1237–1238, 1994.

[41] V. Plagianakos and M. Vrahatis, "Neural network training with constrained integer weights," in *Evolutionary Computation, Proceedings of the 1999 Congress on*, 1999, vol. 3, p. 2013.

[42] J. Onuki, "Ann accelerator by parallel processor based on DSP," in *Neural Networks, 1993. IJCNN '93-Nagoya. Proceedings of 1993 International Joint Conference on*, 1993, vol. 2, pp. 1913–1916.

[43] M. Mohamadian, E. Nowicki, F. Ashrafzadeh, A. Chu, R. Sachdeva, and E. Evanik, "A novel neural network controller and its efficient DSP implementation for vector-controlled induction motor drives," *Industry Applications, IEEE Transactions on*, vol. 39, no. 6, pp. 1622–1629, 2003.

[44] S. Chen, C. Hsu, and W. Wang, "DSP-based fuzzy neural network and its application in speech recognition," in *Systems, Man, and Cybernetics, 1999 IEEE International Conference on*, 1999, vol. 6, pp. 110–114.

[45] C. Wolff, G. Hartmann, and U. Ruckert, "Parspike-a parallel dsp-accelerator for dynamic simulation of large spiking neural networks," in *Microelectronics for Neural, Fuzzy and Bio-Inspired Systems, 1999. MicroNeuro '99. Proceedings of the Seventh International Conference on*, 1999, pp. 324–331.

[46] R. Schrott, A. Keuer, J. Taube, D. Schmuck, H. Beikirch, W. Baumann, and E. Schreiber, "Real-time data analysis of action potentials," in *Computational Intelligence for Measurement Systems and Applications, 2004. CIMSA. 2004 IEEE International Conference on*, July 2004, pp. 26–29.

[47] D. Folegnani and A. Gonzalez, "Reducing power consumption of the issue logic," in *Proceedings of Workshop on Complexity-Effective Design held in conjunction with ISCA 2000*, 2000.

[48] D. Cambre, E. Boemo, and E. Todorovich, "Arithmetic operations and their energy consumption in the nios ii embedded processor," in *International Conference on Reconfigurable Computing and FPGAs*, 2008, pp. 151–156.

[49] J. Eldredge and B. Hutchings, "Density enhancement of a neural network using fpgas and run-time recon guration," in *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, 1994, pp. 180–188.

[50] H. Xu, G. Wang, and C. Baird, "A fuzzy neural networks technique with fast backpropagation learning," in *Neural Networks, International Joint Conference on*, 1992, vol. 1, pp. 214–219.

[51] Z. Liao and A Dexter, "An inferential control scheme for optimizing the operation of boilers in multi-zone heating systems," in *Buiding Services Engineering Research Technology*, 2003, vol. 24, pp. 245–256.

[52] S. Jassar, Z. Liao, and L Zhao, "Adaptive neuro-fuzzy based inferential sensor model for estimating the average air temperature in space heating systems," in *Building Enviroment*, 2009, vol. 44, pp. 1609–1616.

[53] BRE and ICITE, "Controller efficiency improvement for commercial and industrial gas and oil fired boilers," in *A Craft Project*, 1999-2001, Contract JOE-CT98-7010.

[54] M. Beale M Hagan, H. Demuth, *Neural Network Design*, PWS Publishing Company, 1996.

[55] Jeff Heaton, *Introduction to Neural Networks for Java, 2nd Edition*, Heaton Research, 2008.

[56] Z. Liao and A Dexter, "Aa simplified physical model for estimating the average air temperature in multi-zone heating systems," in *Building Env*, 2004, vol. 39, p. 10131022.