5

Integer Neural Networks On Embedded Systems

Thomas Behan, Ziayi Liao and Lian Zhao
Ryerson University Canada

1. Introduction

Neural networks are one of the many types of intelligent systems that are in use today. These systems allow for machines to make decisions based on relationships that are either not fully understood or are very complex. Neural networks are trained by providing a series of inputs and the corresponding outputs for the task being performed. Once the network has been trained, it will produce an output the same as, or very close to, the actual value, without the relationship between the inputs and outputs being explicitly programmed into the network. For this reason intelligent system have a very wide range of applications, ranging from voice and face recognition to simpler things like motor control and inferential sensors. In this chapter we will explore the use of neural networks on the lowest end of devices capable of implementing neural networks. Specifically we will look at integer neural networks.

Integer Neural Networks (INNs) are of particular interest when dealing with very low cost microcontrollers. Integer neural networks are implementations of Artificial Neural Networks (ANN) that only use integer values, as opposed to conventional implementations that use floating or fixed-point values. The type of values used can have a large impact on both the accuracy and execution speed of a network, especially when dealing with low cost microcontrollers that lack the hardware to perform floating or fixed-point arithmetic.

In this chapter we will explore the reasons behind the need for integer neural networks, the types of hardware that can expect an increase in performance when using integer-only networks, and the relative increase in performance on those types of hardware.

Additionally, we will look at the advantages of using DSP operations that are now available on these low cost microcontrollers. Finally we will look at an example implementation of an integer neural network to see how quantization affects the accuracy of the output of the network (Dragaci, 1999).

2. Integer and Floating Point Hardware

The motivation behind using integer neural networks is to reduce the final cost 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 then integer versions, and thus more expensive. 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 performance, 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 long since moved past these limitations, 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. Recently new integer-only microcontrollers (such as the Microchip dsPIC30fxxxx and dsPIC33fxxxx series chips) have been released that included some Digital Signal Processor (DSP) functions, which have been shown to greatly increase the performance of 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, Figure 1. Typically the MAC instruction uses two pairs of registers and a special accumulator register. In each pair, one register hold 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 to the data registers. The address registers are then post incremented to 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 become so large that the register overflows.

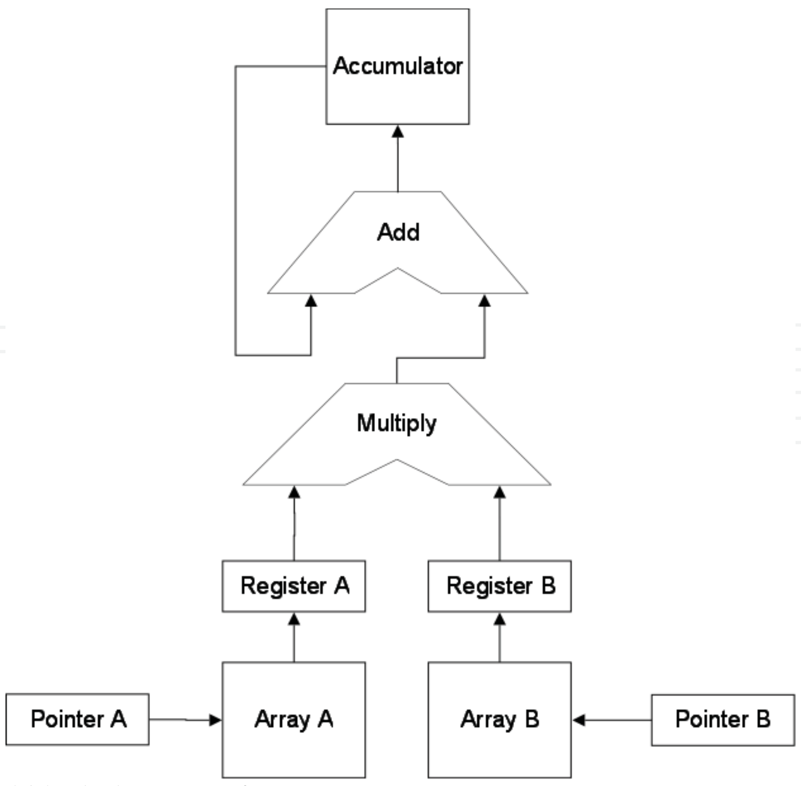

Fig. 1. Simplified block diagram of MAC instruction.

The MAC instruction is typically used to accelerate convolution. Convolution (1) is used in many applications, but is especially noteworthy for its use in FIR and IIR digital filters.

$$(f \times g)(n) = \sum f(m) \cdot g(n - m) \tag{1}$$

$$a \leftarrow a + b \cdot c \tag{2}$$

The MAC instruction can accelerate this operation by combining all of the accumulation, multiplication, increment, and move operations in one instruction cycle (2). 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 network because convolution and the calculation of the net value of a neural network are mathematically identical (3). The output of a neuron is a function of the accumulated products of all the weights and inputs in that neuron (4).

$$Net = Inputs \cdot Weights \tag{3}$$

$$Output = f(Net) \tag{4}$$

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.

3. Integer Neural Networks

3.1 Feedforward

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, recurrent networks, radial networks, etc. Each type of network has its own advantages and disadvantages; the feedforward network was chosen as the example network for its simplicity. Figure 2 sows a simple 2-2-1 neural network that is used later for testing.

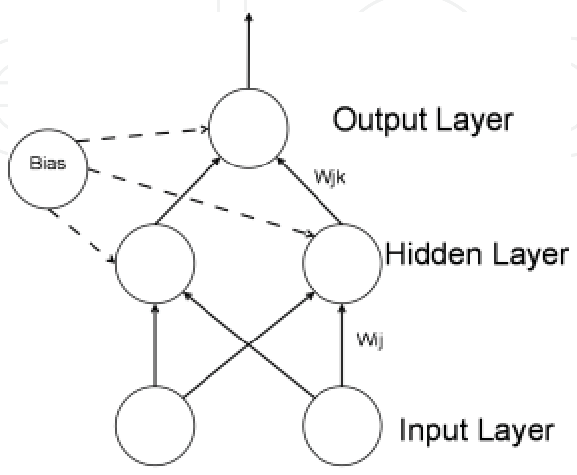

Fig. 2. Diagram of a 2-2-1 neural network.

The output of a neuron is a function of the accumulated product of the neurons inputs and its weights. The function is 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, 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 then floating-point values.
Typically an activation function output ranges from 0 to 1 or -1 to +1. While this is not an issue for a floating point network, it is an integer neural network. Taking the second case, an integer neural network would quantize the output of the activation function into -1, 0, +1. The does not provide nearly enough selectivity to provide accurate classification. One method to avoid this is to stretch the activation function so that more function outputs exist as whole values. Figure 3 shows the activation function (tanh) stretched according to (5).

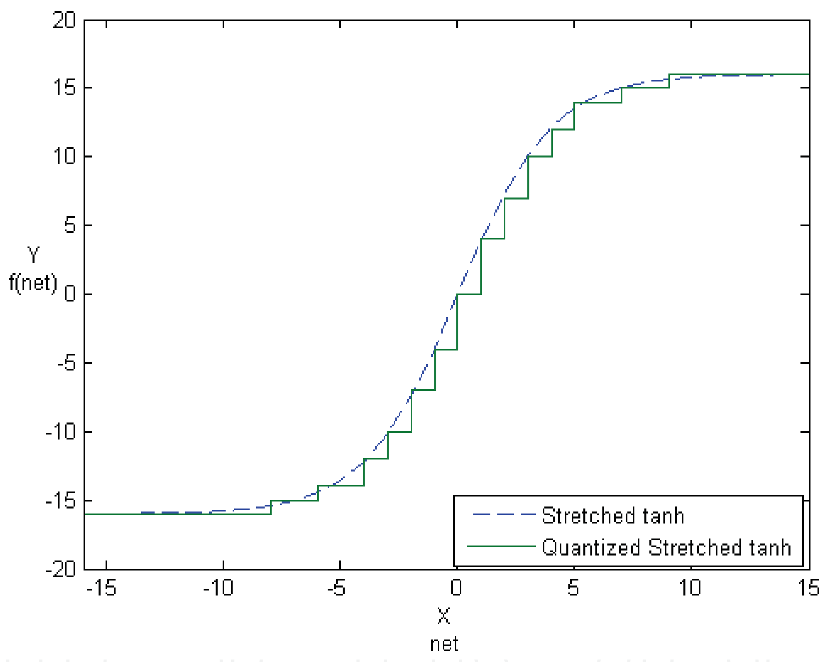

Fig. 3. Stretched activtion function (tanh).

$$Y = 16 \bullet \tanh\left(\frac{X}{4}\right) \tag{5}$$

This quantized activation function can be stored in a Look Up Table (LUT) so that the tanh function does not need to implemented. Using a LUT is much faster than calculating tanh even if the microcontroller supports floating-point operations. In a LUT the output value of the activation function is retrieved from memory based on the value of Net. This means that the tanh value does not have to be calculated which can be a time and resource consuming task.

3.2 Backpropagation

Training an integer neural network remains a serious challenge. The most common method of training a neural network is backpropagation. 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 to further reduce the rate at which the weights are updated as seen in (6). 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. 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 (7). 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.

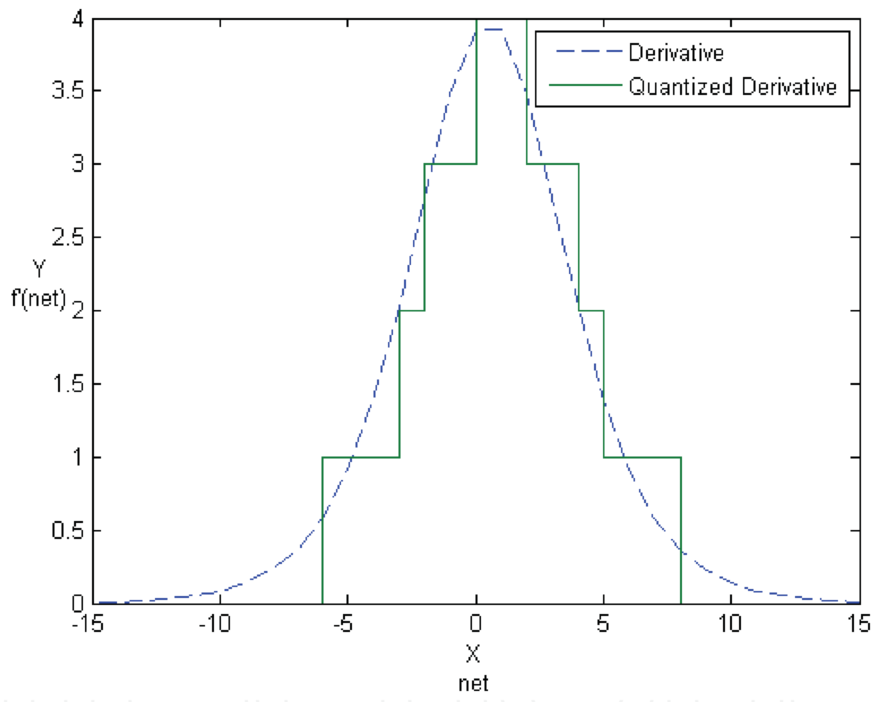

Fig. 4. Derivative of stretched tanh.

Figure 4 shows that only a small portion of the input space for the derivative of the activation function is non-zero. This implies that the weight update is zero and the network does not move towards a solution.

$$\Delta\omega = \eta\delta \tag{6}$$

$$\delta = (t - o) f'(Net) y \tag{7}$$

Equation (6) 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, *y*, the derivative of the activation function *f'(Net)*, and the difference between the target value and the output (7) . For values of Net where f'(Net) is zero, δ is also zero, leading to Δω the weight update, to also be zero (7).
Without the use of the derivative of the stretched tanh function, normal backpropagation cannot be performed. There are many approaches to this problem, such as fixed weight updates (Tang et al., 1997), as shown in equations (8) and (9).

$$\delta = Net \bullet Input \tag{8}$$

$$\Delta\omega = \begin{cases} +1 & if \quad \delta > 0 \\ -1 & if \quad \delta < 0 \end{cases} \tag{9}$$

This simplified backpropagation 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 iteration. Other methods such as (Tang et al., 1997) and (Xu et al., 1992) provide some other methods of weight updates for integer neural networks using different solutions. Training integer neural networks remains much more difficult then training floating-point networks.

4. Execution Time Comparison

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 can allow the microcontroller to operate at a reduced clock speed and thereby consume less power. To compare the performance of integer neural networks and floating point neural networks a very simple 2-2-1 neural network, shown in Figure 2, was trained to solve the XOR problem and then implemented in hardware. For testing, a dsPIC30F2011 (Behan et al., 2008) 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. 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. Equations (10), (11), and (12) show the output of the neurons at the output, hidden, and inputs layers respectively.

$$Out_k = f(\sum W_{jk} \bullet Out_j) \tag{10}$$

$$Out_j = f(\sum W_{ij} \bullet Out_i) \tag{11}$$

$$Out_i = xi \tag{12}$$

At the output layer (10) the output is the sum of the weights (*W_{jk}*) and the outputs from the previous layer (*Out_j*), this 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 (11) is the same, except the previous layer is now the input layer. The input (12) layer is simply the inputs to the network.
Three variants of the code were written, one with floating point variables and constants, and one with integer variables and constants. The integer version has two subversions; one with DSP acceleration and one without. 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. One neuron shows the time needed to calculate the output of one neuron. Further details on the testing and performance comparison are available in (Behan et al., 2008). The time to execute different parts of the neural network in clock cycles is shown in Table 1. Fastest times are in bold.

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

Table 1. Comparison of execution times.

As seen in Table 1, the integer neural network is the 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. For this small network with few interconnects the integer version operates at half the speed for the whole sample set. For larger networks where the Net calculation accounts for more of the overall calculation time, the performance gap between DSP and non-DSP will increase.
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. As with the difference between the DSP and non-DSP versions, as the size of the network increases so too will the performance gap increase between integer and floating point networks. The gap in performance is directly proportional to the amount of processing time needed for the Net calculation relative to the rest of the system. It is important to remember that the activation function was 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. 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.

5. Accuracy and Quantization

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 effects the of quantization we will use the following example system. The goal of this system is to determine the temperature inside of a building (T_{avg}), using external measurements (Liao & Dexter, 2003). The external measurements used are the outside temperature T_0, the amount of solar radiation (Q_{sol}), and the energy consumed by the boiler (Q_{in}) (Jassar et al., 2009). All of the training data was collected from a specialist experimental facility (BRE & INCITE. 2001) and are normalized to values between -1 and +1. A 3x20x3x1 network, shown in Figure 5, is then trained using backpropagation.

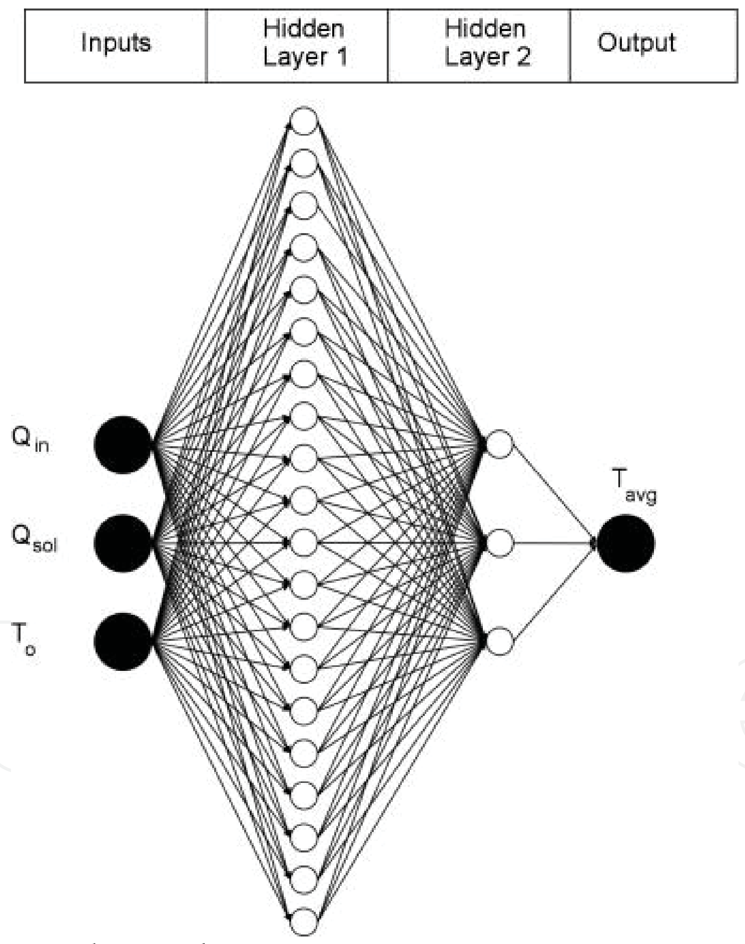

Fig. 5. A 3x20x3x1 neural network.

This creates a base network from which the integer network are derived and compared.
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. 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 (13) 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 (14).

$$O = \tanh(I \bullet W + B) \tag{13}$$

$$O = \tanh((\, q(I * S_f) * q(W * S_f) + q(B * S_f^{\,2}) \,/\, S_f^{\,2}) \tag{14}$$

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 (15) and (16).

$$O = \tanh((I * S_f * W * S_f + B * S_f^{\,2}) \,/\, S_f^{\,2}) \tag{15}$$

$$O = \tanh((I * W * S_f^{\,2} + B * S_f^{\,2}) \,/\, S_f^{\,2}) \tag{16}$$

Once the net value has been calculated the scaling is removed by dividing by the total amount of scaling, S_f^2. 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 O of this layer becomes the Input for the next layer.

The results of the trained base network is shown in Figure 6. 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.

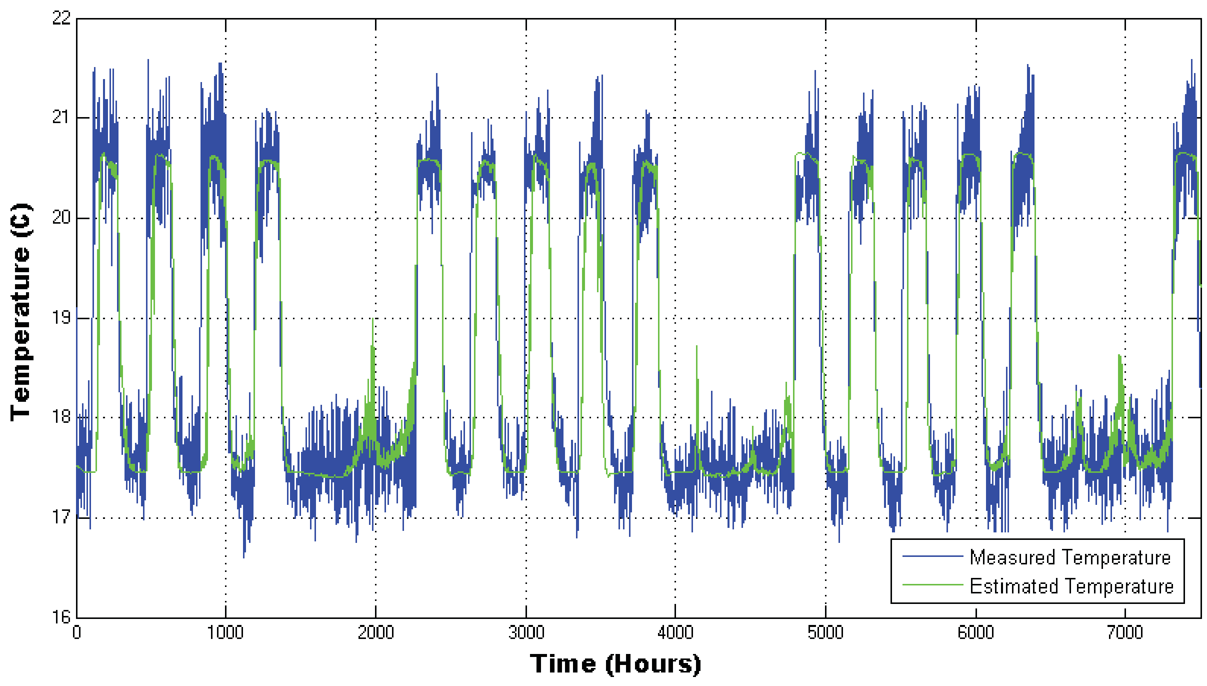

Fig. 6. Output of the base neural network.

Figure 6 shows the actual T_{avg} 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.

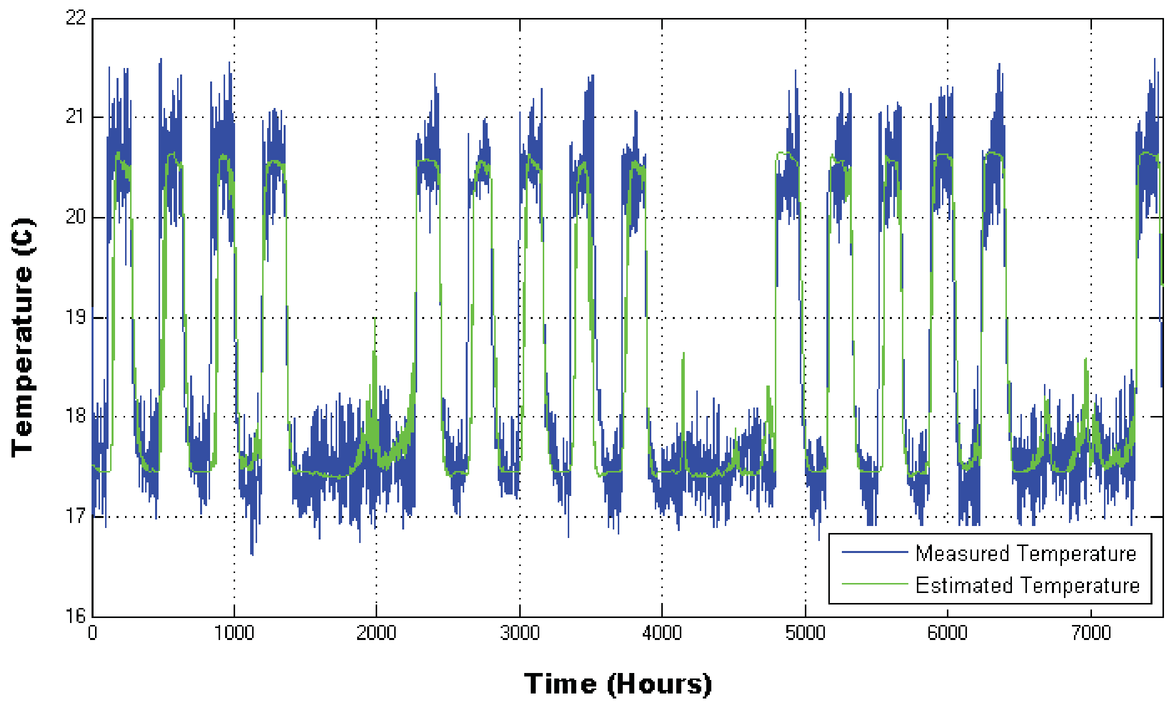

Fig. 7. Output of an integer neural network with S_f = 128.

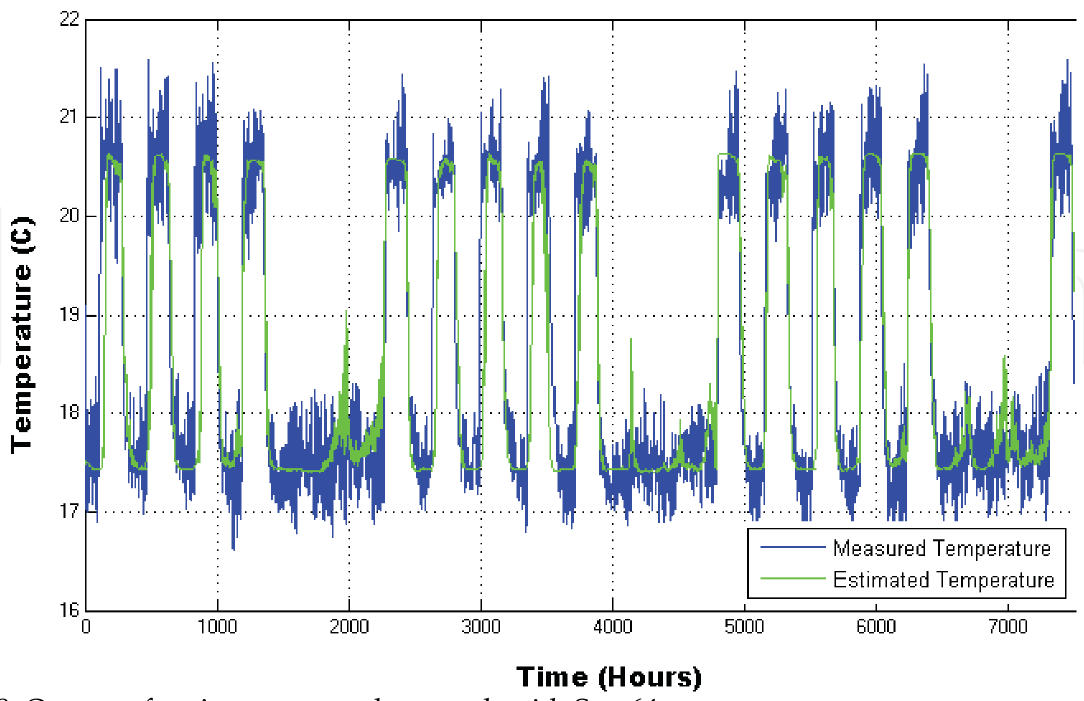

Fig. 8. Output of an integer neural network with $S_f = 64$.

Figure 7 and Figure 8 show the network outputs when S_f is equal to 128 and 64 respectively. At this level of quantization, the integer neural networks and the base network are almost identical. However memory usage is significant as the range of input values for the LUT are from - S_f^2 to $+S_f^2$. 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.

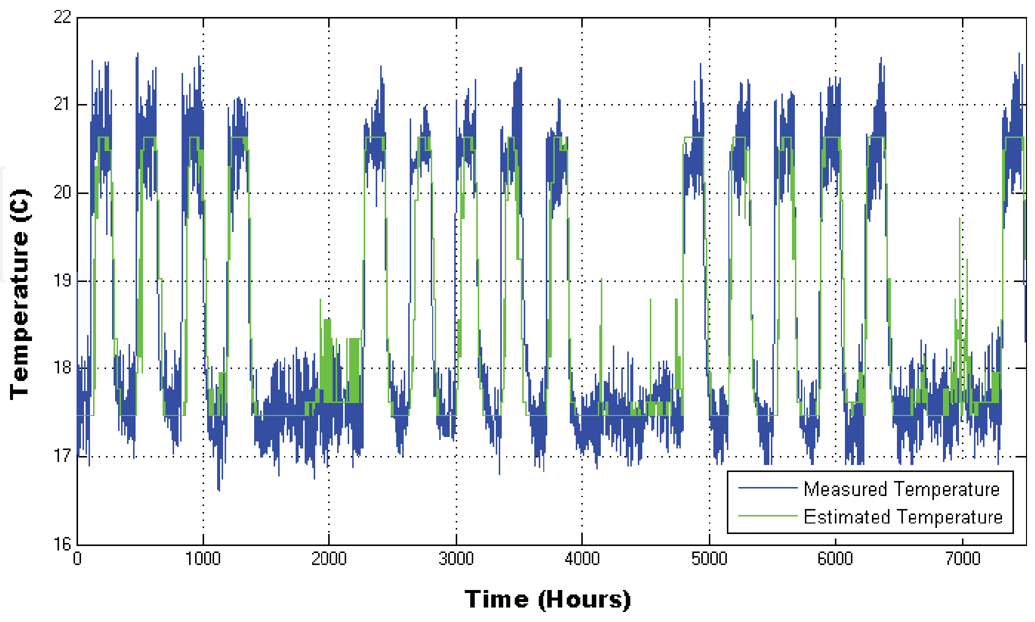

Fig. 9. Output of an integer neural network with $S_f = 8$.

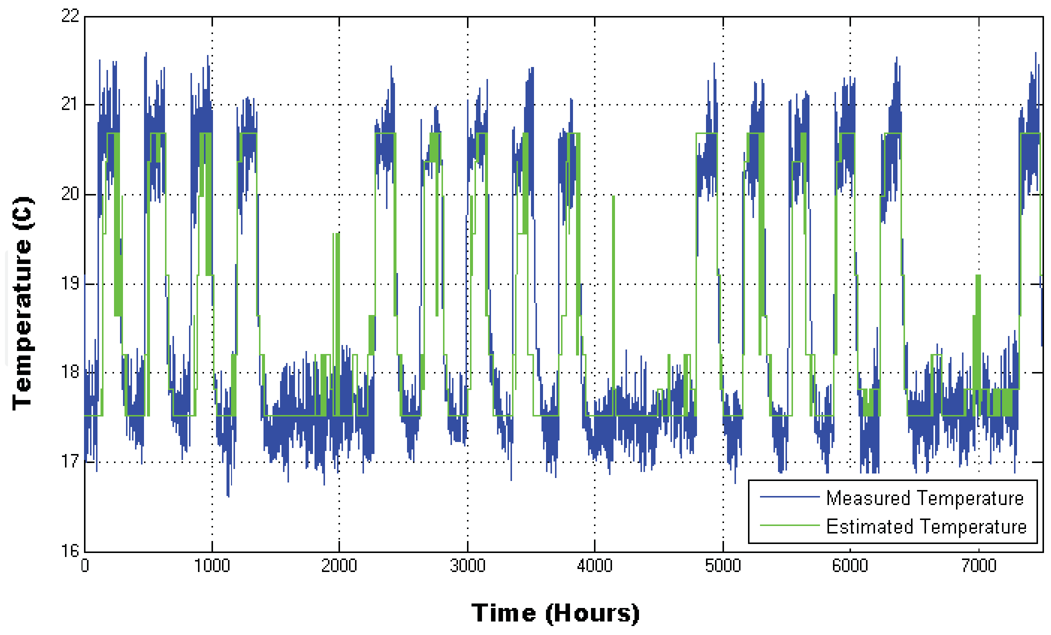

Fig. 10. Output of an integer neural network with S_f = 4.

Figure 9 and Figure 10 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 becomes degraded.

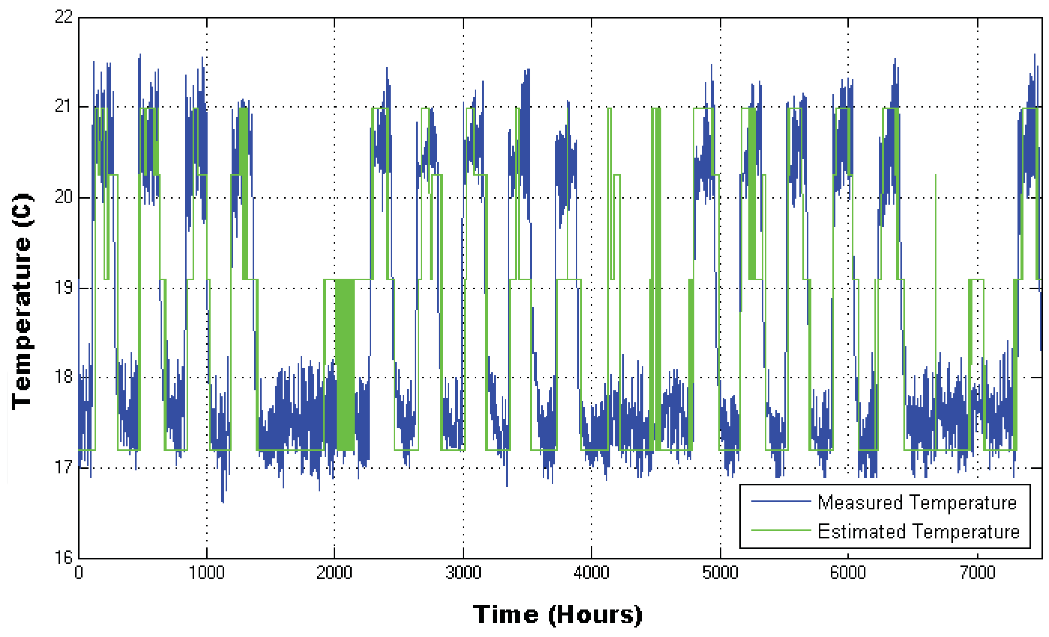

Fig. 11. Output of an integer neural network with S_f = 2.

Figure 11 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. Table 1 shows a comparison of the accuracy of the networks at each level of quantization, using three types of statistical measures.

Network	RMSE	R^2	SSE
Normal	0.60	0.7944	2724
S_f =128	0.60	0.7942	2423
S_f =64	0.60	0.7947	2729
Sf =8	0.63	0.7779	2955
S_f =4	0.66	0.7331	3281
S_f =2	0.95	0.6058	6770

Table 2. Accuracy comparison for various levels of quantization.

As can be seen from Figures Figure 6 to Figure 11 and Table 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. 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. Additionally if the quantization results in weights or inputs larger then 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.

6. Conclusion

Almost all system designs involve tradeoffs 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 and speed 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. For this reason the decision on which type of network to use is subjective and must be viewed in light of the design goals for the system. When cost is the primary design concern, integer neural networks offer a very compelling combination of accuracy, speed, and cost that can significantly improve the price to performance ratio of the final product.

7. References

Liao, Z. & Dexter, A.L (2003). An Inferential Control Scheme for Optimizing the Operation of Boilers in Multi-zone Heating Systems. *Buiding Services Engineering Research Technology,* Vol. 24, No. 4, pp. 245-256.

Behan, T.; Liao, Z.; Zhao, L. & Yang C. (2008). Accelerating Integer Neural Networks On Low Cost DSPs. *WASET Proceedings*, Vol. 36, No. 7, pp 1270-1273.

Draghici, S. (1999). Some new results on the capabilities of integer weight neural networks in classification problems. *International Joint Conference on Neural Networks,* Vol. 23, No. 3, pp 519-524.

Jassar, S.; Liao, Z. & Zhao, L (2009). Adaptive neuro-fuzzy based inferential sensor model for estimating the average air temperature in space heating systems. *Building Enviroment,* Vol. 44, pp 1609-1616.

BRE & INCITE (1999-2001). Controller efficiency improvement for commercial and industrial gas and oil fired boilers, *A Craft Project,* Contract JOE-CT98-7010.

Tang, J.; Varley, M.R. and Peak, M.S. (1997). Hardware implementations of multi-layer feedforward neural networks and error backpropagation using 8-bit pic microcontrollers. *Neural and Fuzzy Systesms: Design Hardware and Applications*, Vol
133, pp 2/1-1/5.

Xu, H.Y.; Wang, G.Z. and Baird, C.B. (1992). A fuzzy neural netwrorks technique with fast backpropagation learning. *International Joint Conference on Neural Networks,* Vol. 1, pp 214-219.