World Academy of Science, Engineering and Technology
International Journal of Electrical and Computer Engineering
Vol:2, No:12, 2008

Accelerating Integer Neural Networks On Low Cost DSPs

Thomas Behan, Zaiyi Liao, Lian Zhao, Chunting Yang

Abstract**—** In this paper, low end Digital Signal Processors (DSPs) are applied to accelerate integer neural networks. The use of DSPs to accelerate neural networks has been a topic of study for some time, and has demonstrated significant performance improvements. Recently, work has been done on integer only neural networks, which greatly reduces hardware requirements, and thus allows for cheaper hardware implementation. DSPs with Arithmetic Logic Units (ALUs) that support floating or fixed point arithmetic are generally more expensive than their integer only counterparts due to increased circuit complexity. However if the need for floating or fixed point math operation can be removed, then simpler, lower cost DSPs can be used. To achieve this, an integer only neural network is created in this paper, which is then accelerated by using DSP instructions to improve performance.

Keywords**—** Digital Signal Processor (DSP), Integer Neural Network (INN), Low Cost Neural Network, Integer Neural Network DSP Implementation.

I. Introduction

Integer Neural Networks (INNs) have been a topic of research [1]-[3], usually with the goal to reduce hardware complexity for implementation on Field-Programmable Gate Arrays (FPGAs). INNs are, from hardware perspective, much simpler, and so take up less space and consume less power. However, FPGAs are not the only system that can benefit from this research, many low cost micro-controllers also lack the hardware to perform floating or fixed point calculations. While these functions can be reproduced in software, this entails a significant degradation in performance. The level of performance degradation usually depends on the quality of the compiler. In addition, there are now low cost micro-controllers with DSP functionality. DSP operations have been used to greatly improve the performance of neural networks [4], and have been implemented in a diverse range of applications such as motor control [5] and speed recognition [6]. This paper aims to demonstrate the feasibility of INNs accelerated by DSP instructions on low cost micro-controllers.

II. Hardware

There exists a wide selection of micro-controllers which have DSP operations, but with ALUs that lack floating/fixed point arithmetic. For the purposes of this paper any micro-controller will work provided that its ALU can perform the most basic DSP operations, in particular the Multiply ACcumulate (MAC) instruction. For this paper the dsPIC30F2011 is used as it is cheap while still meeting the minimum requirements. Typically the MAC instruction uses two registers as pointers to hold the memory locations of the data, two registers to hold the data taken from the memory and one accumulator as shown in Fig. 1. The accumulator may be wider than other registers to accommodate the large values that can be generated by a MAC instruction. On the dsPIC30F2011 the accumulator is 40bits rather than the usual 16bit, this makes it much less likely for the accumulator to overflow. The pointers automatically post increment to the next memory locations so that they are ready for the next call of the MAC instruction. The MAC instruction is used to perform convolution in typical DSP systems expressed as,

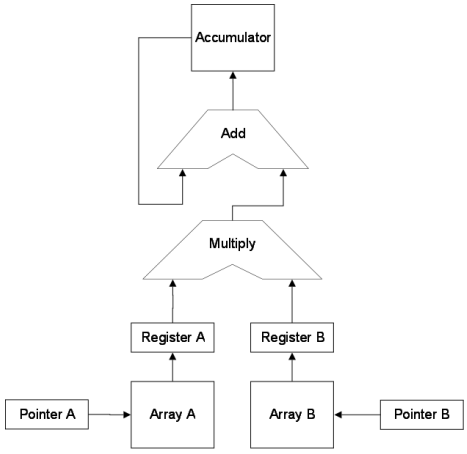

Fig. 1. Simplified block diagram of MAC instruction.

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m) \cdot g(n-m). \tag{1}$$

The operation provided by the MAC instruction is the sum of the product to two vectors, accumulated over successive iterations represented as,

$$a \leftarrow a + b \times c. \tag{2}$$

The output of a neuron can be expressed as,

$$\text{Output} = f(\text{Net}). \tag{3}$$

Thomas Behan is a graduate student with the Department of Electrical and Computer Engieering at Ryerson University, Canada (email:thomas.behan@ryerson.ca)

Zaiyi Liao is with the Department of Architectural Science at Ryerson University, Canada (email:zliao@ryerson.ca)

Lian Zhao is with the Department of Electrical and Computer Engieering at Ryerson University, Canada (email:lzhao@ee.ryerson.ca)

Chunting Yang is with the Faculty of Information and Electronic Engineering, Zhejiang University of Science and Technology, P.R.China (email:yangct@zust.edu.cn)

$$\text{Net} = \text{Inputs} \times \text{Weights} \tag{4}$$

It happens that the largest portion of the time required to calculate the above output is in the multiplication and accumulation of inputs and weights, that is, calculating the net (4) of a neuron. In a neural network the input and weight vectors operate similar to two waveforms conducting convolution, but produce one output rather than another waveform.

Because of this similarity the MAC instruction can accelerate a neural network in much the same way as it accelerates convolution, by decreasing total number of instructions needed to perform the operation. Therefore, the efficiency can be increased and the amount of time needed to simulate a neuron can be decreased.

III. Neural Network

A. Feedforward

The values for the weights are limited by the microcontroller to 16 bits (representing values from -32 768 to +32 767). Further restricting the range of values for the weights may provide additional benefits in training the neural network [2]. However it has no effect on the speed of the neural network and so was not implemented. For an activation function a stretched hyperbolic tan (tanh) is used. Normally tanh provides values from -1 to 1 which is not acceptable for an INN. This is also true for other popular activation functions such as the logistic sigmoid. To get around this problem tanh must be stretched to include more integer numbers. Then once the tanh function is stretched it must also be quantized for use in a Look Up Table (LUT). Quantizing the sigmoid along with the use of integer weights present the primary problem of INNs in that they limit the expressive power of the network as described in [3]. However, it also provides the advantage of using LUTs, which are much faster than performing the calculations even if the hardware supports floating or fixed point math operations. The activation function used is shown as

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

which is then quantized to the nearest integer number. Fig. 2 shows the original stretched tanh and the quantized version that is used in the LUT.

B. Backpropagation

Serious challenges arise when implementing backpropagation on an INN. The most serious problem is the lack of weight update mitigation. In a normal neural network, the rate at which the weights change is mitigated by the decimal numbers from the backpropagation itself as well as a learning rate factor η . However without these attenuating factors the weight updates are just the product of two integers, which makes the system unstable because the weight updates can never decrease in magnitude. This is especially true for small LUTs where a large weight update can cause the output swing from one end of the LUT to the other in a single update. When this happens the weights can trap the neuron at the extremes of the activation function. Another source of difficulty is the

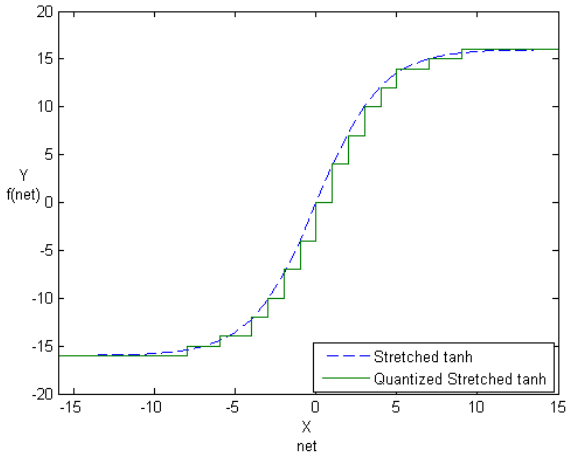

Fig. 2. Stretched tanh and quantized stretched tanh.

derivative of the stretched tanh function. The extremities of the derivative of the stretched tanh function approach zero as shown in Fig. 3.

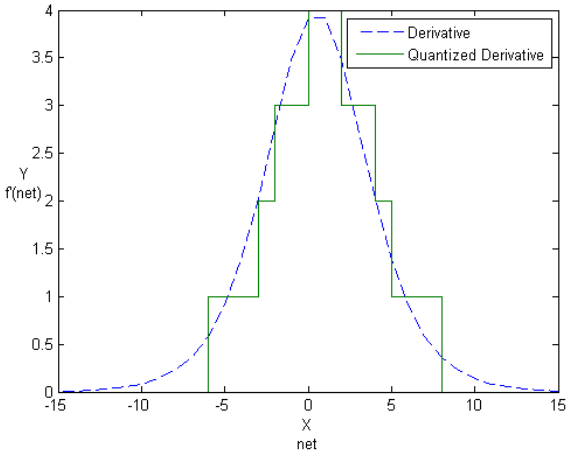

Fig. 3. Derivatives of stretched tanh and quantized stretched tanh functions.

When the derivative is quantized, all of the values are rounded to the nearest integer and values less then 0.5 become zero. This can be seen in Fig. 3 where the quantized values drop to zero as the Net values approachs the extermes. This means that for a large portion of the net values f'(net) are zero.

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

The weight updates (6) are normally determined by the backpropagated error of the weight, δ, and the learning rate η.

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

Through the delta rule we can determine the backpropagated error for the output weights (7). The error is determined by the difference between the target output t, and the actual output o, multiplied by the derivative of the activation function $f'(Net)$ and the input assosiated with the weight being updated. In the

case of the output neuron, the input is the output y of a lower layer neuron. This δ_o is then used to determine the weight update (6). However if the Net value is near the extremes of the input space, then $f'(Net)$ will become zero if the quantized sigmoid is used. The result is that backpropagated error δ_o becomes zero (7). This causes weight update Δw to also become zero (6), meaning that the weight will not update and the network will not move towards a solution.

Without the use of the derivative of the stretched tanh function, normal backpropagation cannot be performed. While there are many approaches to this problem, the goal of using only integers restricts the use of many of these methods. A simple solution is to use a fixed interval for the weight update as done in [7]. In [7] the integer value one is used as the interval.

$$\begin{aligned} \delta &= Net \times Input \\ \Delta w &= \begin{cases} 1 & \text{if } \delta > 0 \\ -1 & \text{if } \delta < 0 \end{cases} \end{aligned} \tag{8}$$

By using a simplified backpropagation (8), the sign of the weight update interval is determined to ensure that the fixed update interval moves the network towards a stable solution. In this case, $\delta = Net \times Input$ because the derivative of tanh is symetrical about the y-axis and the magnitude is ignored. This method is very simple and provides an integer only approach but suffers from some significate drawbacks. First, if the initial weights are far from a solution it will take many more epochs to reach the solution, because the proper magnitude of the weight update is not determined. Also, because all of the weight updates are the same magnitude, the network has a propensity to get caught in weight update loops. Some weight update loops can be avoided by using an odd number of samples so that the final weight update cannot equal zero, otherwise it is possible for all of the weight updates to equal zero and halt the learning process. However the method remains very suspectable to weight update loops, therefore it is often necessary to make multiple attempts with different initial weights. For backpropagation implementations that are not restricted to integer calculations, there are other methods [7], [8] that can produce integer solutions while avoiding some of these problems.

IV. Implementation

For comparison, a 2-2-1 network is used as shown in Fig.4. The drawbacks of the backpropagation method, particularly the need for many weight reinstallations, make it unsuitable for online training. Therefore the network is trained offline on a computer. Once the computer has found a solution the weights from the computer's network are programmed into the network on the dsPIC30F2011. The four possible input combinations are tested to ensure that the network is performing the XOR operation correctly. For timing purposes, a simulator is used because it can track the number of clock cycles needed to execute the program. This provides the most accurate timing measurement.

The output for this neural network is given by:

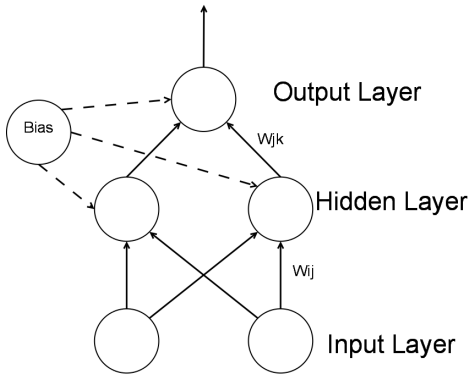

Fig. 4. 2-2-1 Neural Network

$$Out_k = f(\sum w_{jk} \times Out_j) \tag{9}$$

$$Out_j = f(\sum w_{ij} \times Out_i) \tag{10}$$

$$Out_i = x_i \tag{11}$$

Neurons in the output layer (9) and the hidden layer (10) must calculate a net value given as

$$Net = \sum W \times Out \tag{12}$$

which is passed to the activation function. While on DSP systems the MAC instruction is used to provide acceleration for convolution in (1), in this system it is used to accelerate the calculation of the net value for a neuron given in (12).

For this reason, the net calculation code will be the main focus of the testing, all other parts of the code are identical. The network also runs with floating point inputs and weights to demonstrate the performance degradation caused by using floating point operations on the dsPIC30F2011, which lacks floating point hardware. The main program is written in C and the function for determining the net of the neuron is programmed in both C and assembler (to directly access the DSP operations). The program is executed in three variations: Integer variables with DSP instructions, integer variables but without DSP instructions, and variables as floats without DSP operations (which the chip cannot do). The program is re-complied each time, and the number of clock cycles to complete the net calculation, one neuron firing, and the whole sample set is recorded.

V. Results

A. Data Collection

The test results are shown in Table I. All results are shown in Clock Cycles (CC). Values that are averaged are rounded up the nearest whole clock cycle. Each clock cycle takes $33.\overline{3}$ns when the dsPIC30F2011 is operated at its maximum clock speed of 120MHz. Net Calculation shows how many cycles are required to determine the net of a neuron. This value is an averaged value across one sample. The shortest execution time is highlighted with bold face numbers.

TABLE I
EXECUTION TIME IN CLOCK CYCLES

	INN with DSP	INN no DSP	Floating Point
Net Calculation	**21cc**	87cc	776cc
One Neuron	**49cc**	115cc	923cc
Whole Sample Set	**717cc**	1509cc	11996cc

One Neuron shows the time required to determine the output of a neuron. This value is the average across one sample. It shows that using the LUT requires 28-29 clock cycles when using integer values, however for floating point it takes much longer. Also because floating point values cannot be used to address memory locations the output is always zeroed and the same position of the lookup table is always accessed. This means that the output for the floating point neural network is never correct. However this prevents the floating point method from being converted to an integer or having to calculate the activation function. Both of which would significantly decrease performance and bias the results against floating point.

Whole Sample Set is the execution time to run through all four possible input combinations. These values include the time taken to change the input values and perform a while loop, as well as various other operation not directly related to the neural net.

B. Discussion

From Table I, we can see that INN with DSP is the fastest method, more than twice as fast as INN in all aspects. The DSP operations provide a 4 $\times$ speed increase when calculating the net of a neuron. Showing that DSP operations do significantly increase the performance. The calculation of the net value for a 2 input (plus bias) neuron when using DSP operations accounts for less than half of the overall execution time. The additional clock cycles are due to accessing the LUT and function calls. For the calculation of the whole sample set the speed increase afforded by the DSP operations has been diluted because of all of the other operations that must be performed. However even at this stage the DSP operations provide a 2$\times$ performance increase over an INN without DSP operations. Performing operations with floating point variables produces very poor results. Calculating the net value with floating point takes longer than the other two methods by a large margin. The INN without DSP is 7.9$\times$ faster than floating point for calculating the whole sample set. While the INN with DSP is 16.7$\times$ faster than floating point when calculating the entire sample set. This performance most clearly demonstrates the utility of INNs on low cost micro-controllers, especially those with DSP functions. For this reason neural networks using floating point operations, while possible on low end micro-controllers are often not practical for implementations with timing constraints. Neural networks using floating or fixed point are best implemented on chips that can support these operations, despite their cost and power consumption. Lastly it should be noted that because all of the performance improvement comes from the calculation of the net values, the performance scales relative to the total number of neural connections rather than the total number of neurons. This means that the relative performance gain increases as the number of connections inside the neural network increases. The 2-2-1 network is one of the simplest networks having only 9 connections, and so the relative performance increase should be considered modest.

VI. CONCLUSION

An INN has been implemented on a micro-controller with an ALU lacking floating point arithmetic and has been accelerated by using DSP operations. Also, a purely integer method for implementing backpropagation has been demonstrated. However the integer backpropagation was not implemented as it does not benefit from DSP operations. The results show that using DSP operation greatly enhanced the performance of the INN. Both the INN with DSP operations and the INN without DSP operations were significantly faster than the neural network using floating point on the dsPIC30F2011. The performance increase also scales favorably. As the complexity of the network increases, the performance gap between INNs with DSP and INNs without DSP also increases. Finally, the primary objective of this paper has been demonstrated: that if a neural network solution can be found using only integers, then an INN can be implemented on a micro-controller and accelerated with DSP operations. This eliminates the need for floating point circuitry, and can greatly reduce the cost and power consumption without increasing complexity of the system.

ACKNOWLEDGMENT

This work is supported by Ontario Centre of Excellence (OCE) under grant number EE50196.
Thanks to Richard Rzeszutak for proof reading and debugging.

REFERENCES

[1] A. H. Khan and E. L. Hines, "Integer-weight neural nets," *Electronics Letters*, vol. 30, no. 15, pp. 1237–1238, 1994.
[2] V. P. Plagianakos and M. N. Vrahatis, "Neural network training with constrained integer weights," in *Evolutionary Computation, Proceedings of the 1999 Congress on*, vol. 3, 1999, p. 2013.
[3] 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*, vol. 1, 1999, pp. 519–524.
[4] J. Onuki, "Ann accelerator by parallel processor based on DSP," in *Neural Networks, 1993. IJCNN '93-Nagoya. Proceedings of 1993 International Joint Conference on*, vol. 2, 1993, pp. 1913–1916.
[5] 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.
[6] S.-C. Chen, C.-C. Hsu, and W.-Y. Wang, "DSP-based fuzzy neural network and its application in speech recognition," in *Systems, Man, and Cybernetics, 1999 IEEE International Conference on*, vol. 6, 1999, pp. 110–114.
[7] J. Tang, M. R. Varley, and M. S. 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] H. Y. Xu, G. Z. Wang, and C. B. Baird, "A fuzzy neural networks technique with fast backpropagation learning," in *Neural Networks, International Joint Conference on*, vol. 1, 1992, pp. 214–219.