International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249 – 8958, Volume-3, Issue-5, June 2014

Artificial Neural Network Implementation in Microchip PIC 18F45J10 8-Bit Microcontroller

Jnana Ranjan Tripathy, Hrudaya Kumar Tripathy, S.S.Nayak

Abstract-Implementing neural networks on an 8-bit microcontroller with limited computing power presents several programming challenges. In order for the network to perform as quickly as possible, creating the software at the assembly level was chosen. Writing the software in assembly allows a level of customization that cannot be achieved with C. However, the need for hardware portability was also a motivating factor and a more generic C implementation was also created. It was also very important to manually manage the very limited amount of data memory. Several assembly routines were created with this purpose in mind. A pseudo floating point arithmetic protocol was created exclusively for neural network calculations along with a multiplication routine for multiplying large numbers. A tanh compatible activation function was also needed. The final procedure is capable of implementing any neural network architecture on a single operating platform.

Keywords: Neural Architecture (NA), Microcontroller, Embedded C, Pseudo Floating Point, Activation Function

I. INTRODUCTION

The first method was to use 16 bits to represent the weights, nodes, and inputs for the neural network. These 16-bits are all significant digits in this pseudo floating point protocol. The 16 bits consist of an 8-bit signed integer and an 8-bit fraction fractional part. The nonconventional part of this floating point routine is the way the exponent and mantissa are stored. Essentially all sixteen bits are the mantissa and the exponent for the neuron is stored elsewhere. This has several advantages. It allows more significant digits for every weight using less memory.

II. HARDWARE IMPLEMENTATIONS

The tools created to build the neural network on the microcontroller resulted in an equally challenging project as the embedded network. However, creating and debugging the assembly version of the neural network would never have been possible without the tools. Now with the automated system almost any trained network can be implemented on the microcontroller in a matter of seconds.
A pseudo floating point arithmetic protocol was created exclusively for neural network calculations along with a multiplication routine for multiplying large numbers. A tanh compatible activation function was also needed. The final procedure is capable of implementing any neural network architecture on a single operating platform. This robust base removes the need to modify the structure of the software to make network architecture changes.

Manuscript Received on June 2014.
Er. Jnana Ranjan Tripathy, Department of Computer Science & Engineering, Biju Pattnaik University of Technology, Orissa Engineering College Bhubaneswar, Odisha-752050, India.
Dr.Hrudaya Kumar Tripathy, Department of Computer Science & Engineering, KIIT University,Bhubaneswar, Odisha, India.
Dr. S.S.Nayak,, Centurion University of Technology & Management Paralakhemundi, Odisha, India.

2.1. PSEUDO FLOATING POINT

The first method was to use 16 bits to represent the weights, nodes, and inputs for the neural network. These 16-bits are all significant digits in this pseudo floating point protocol. The 16 bits consist of an 8-bit signed integer and an 8-bit fraction fractional part. The nonconventional part of this floating point routine is the way the exponent and mantissa are stored. Essentially all sixteen bits are the mantissa and the exponent for the neuron is stored elsewhere. This has several advantages. It allows more significant digits for every weight using less memory. This pseudo floating point protocol is tailored directly to the needs of the neural network forward calculations. This solution requiresthe analysis of the weights of each neuron and scales them accordingly and assigns an exponent for the entire neuron. A similar process is used for the inputs so the entire range will share a single scale factor. This scaling is done off chip before programming in order to save valuable processing time on each and every forward calculation. Scaling does two things, first it prevents overflow by keeping the numbers within operating regions, and secondly automatically filters out inactive weights. For example, if a neuron has weights that are several orders of magnitudes larger than others it will automatically round the smallest weights to zero. These weights being zero allow the calculations to be optimized, unlike using traditional floating point arithmetic. However, if all of the weights are the same magnitude they are all scaled to values that preserve maximum precision and significant digits. In other words, the weights are stored in a manner that minimizes error on a system with limited accuracy. Thus far, all of thesedecisions for scaling the weights are made before the network is programmed on the microcontroller. This process has been automated for ease of use. The Neural Network Trainer was modified to automatically scale the weights and inputs after it trains the network. This is done in Matlab and an example of the scaling process can be seen below.

2.2. MULTIPLICATION

The Pic18F45J10 microcontroller has an 8-bit by 8-bit unsigned hardwaremultiplier. Considering that the hardware multiplier cannot handle floating point values or negative numbers, a routine was needed to allow fast multiplication of fractional values. The multiply routine is passed two sixteen bit numbers, consisting of an eight-bit integer and an eight bit fraction portion. The routine returns a 32-bit product. The result of the multiplication routine is a 32-bit fixed point result shown in Figure 1.

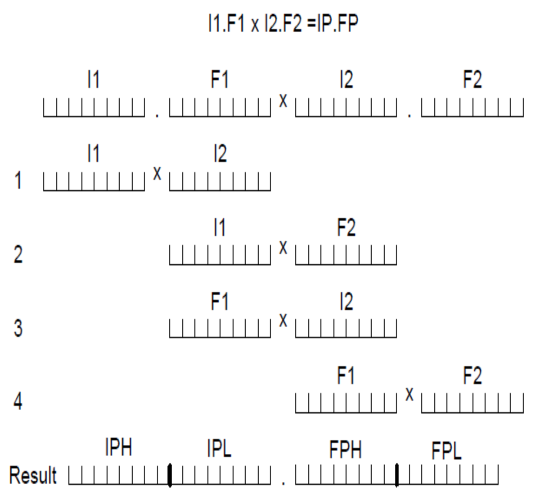


Figure 1: Implementation of 16-bit fixed point multiplication using 8-bit hardware multiplier. Steps 1-4 are summed with place holders to give the final product on the result line. Abbreviations: Integer (I) Fractional (F) Product (P) Lower-Byte (L) Higher-Byte (H).

2.3. Activation Function

A soft activation function was needed for the neural network. The most common activation function is tanh and the definition is shown back in Equation 1. The pure definition tanh was not a reasonable solution for several reasons. Specifically, the exponents would be very difficult to calculate accurately with the limited hardware in a timely fashion. A second order approximation of tanh was chosen for its accuracy as well as its simple arithmetic calculations. Several features were added to the activation function besides simply calculating a second order approximation of tanh. One of these features analyses the inputs to the activation function and converts negative numbers to positive numbers to make the internal calculations faster and reduce the number of values that must be stored in the lookup table. The sign is restored at the end of the activation function. Another feature is a check to see if the neuron is in saturation. In other words, make sure the net value is within a given range. In this case the second order approximation is skipped and the neuron is put into saturation. These features of the second order approximation can be seen in better detail in Figure 2. The routine requires that 30 values be stored in program memory. This is not simply a lookup table for tanh because a much more precise value is required. The tanh equivalent of 25 numbers between zero and four are stored. These numbers, which are the end points of the linear approximation, are rounded off to 16-bits of accuracy. Then a point between each pair from the linear approximation is stored. These points are the peaks of a second-order polynomial that crosses at the same points as the linear approximations. Based on the four most significant bits that are input into the activation function, a linear approximation of tangent hyperbolic is selected. The remaining bits of the number are used in the second-order polynomial. The coefficients for this polynomial were previously indexed by the integer value in the first step.The approximation of tanh is calculated by reading the values of yA, yB and Δy from memory and then the first linear approximation is calculated using yA andyB.

$$y_1(x) = y_A + \frac{(y_B - y_A).x}{2\Delta x} \qquad (1)$$

The next step is the second-order function that corrects most of the error that was introduced by the linearization of the tangent hyperbolic function.

$$y_2(x) = \frac{\Delta y}{\Delta x^2}(\Delta x^2 - (x-\Delta x)^2) \qquad (2)$$

Or

$$y_2(x) = \frac{\Delta y \cdot x(2\Delta x - x)}{\Delta x^2} \qquad (3)$$

In order to utilize 8-bit hardware multiplication, the size of Δx was selected as128. This way the division operation in both equations can be replaced by the right shift operation. Calculation of y1 requires one subtraction, one 8-bit multiplication, one shift right by 7 bits, and one addition. Calculation of y2 requires one 8-bit subtraction, two 8- bit multiplications and shift right by 14-bits. Ideally this activation function would work without any modification, but when the neurons are operating in the linear region (when the net values are between -1 and 1) the activation function is not making full use of the available.

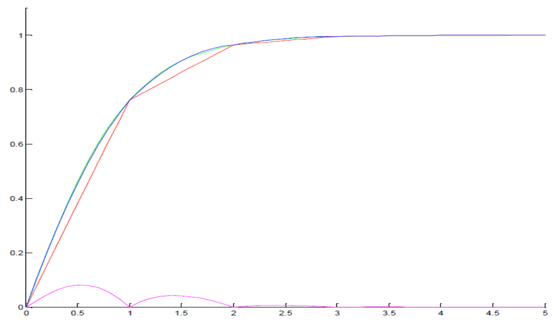


Figure 2: Example of linear approximations (red) and parabolas between 0 and 4(magenta). Tanh (green) and the approximation (blue) are also shown on the graph. Only 4 divisions were used for demonstration purposes.

The activation function is tested in hardware by sending a set of numbers from -5 to +5 and comparing them to the output of the tanh function. The difference between the sets of numbers can be seen in Figure 3.

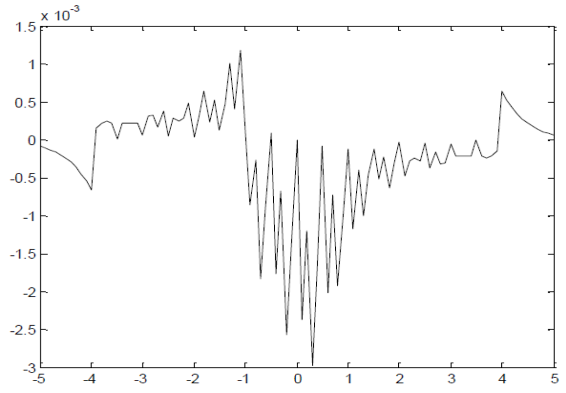


Figure 3. Error from tanh approximation using 6 divisions from -5 to +5.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

2.4. Memory Structures

The Microchip PIC 18F45J10 microcontroller was used to implement the neuralnetwork. The microcontroller has only one true register that can be used for holding data, passing data, and ALU calculations. It has 1 Kbyte of ram memory and when the neural network has 255 .This memory is divided into four 256 byte banks. Only one of these banks can be accessed directly without the use of extra addressing instructions. This one bank has 128 bytes of general purpose memory and 128 bytes of processor configuration memory. This general purpose memory is used as global and temporary variables for calculations. The other three banks are used for the weights and the individual nodes of the neural network. The weights are stored as 16-bit numbers, which consist of an 8-bit integer and an 8-bit fractional part. Two banks are used to store the high and low byte of each weight. This allows for 255 weights to be stored. The zero location is not used for indexing reasons. Figure 13 shows the memory mapping. As the output of the neural network is calculated the output of each neuron and the inputs need to be stored throughout the entire calculation to allow multi-layer connections. These node values are also 16-bit values. This poses a problem because there is only one ram bank left and two banks are needed. This problem is solved by splitting this bank into two separate banks; the low bank and high bank hold the low byte and high byte respectively. Notice this adds an additional limitation to the neural network size. The network may only have 127 total inputs and nodes. This limitation will most likely not be the dominant factor in many cases. Typically the weight limitation would be met prior to approaching the node limit. This memory limitation is only relevant to this microcontroller. This concept could be extended to other microcontrollers or systems with extended ram. More ram could easily allow for even larger networks with greater numbers of neurons and weights. The C version of the software stores all weights and architecture values in program memory not in RAM. There simply is not enough ram for the C version to function if these values are in ram.

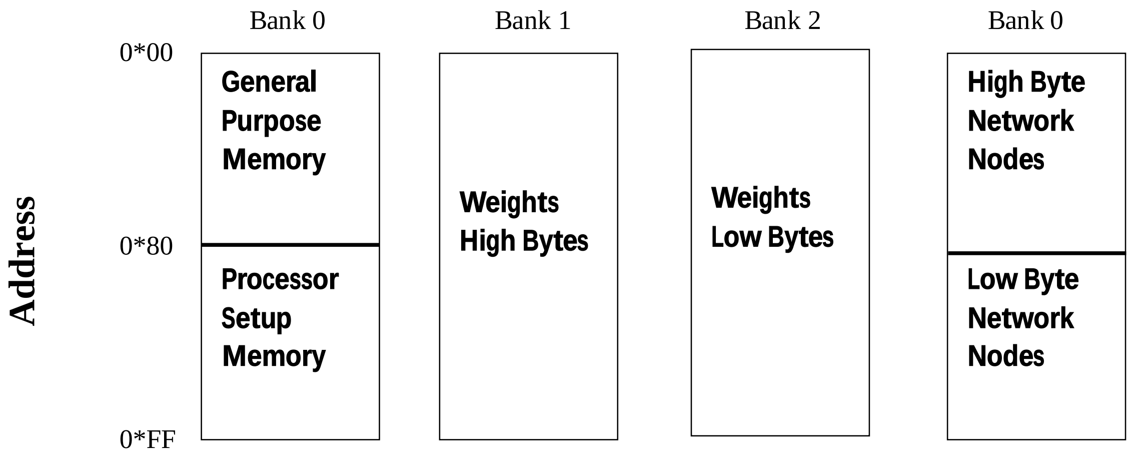


Figure 4. Memory Allocation Table for Pic18F45J10

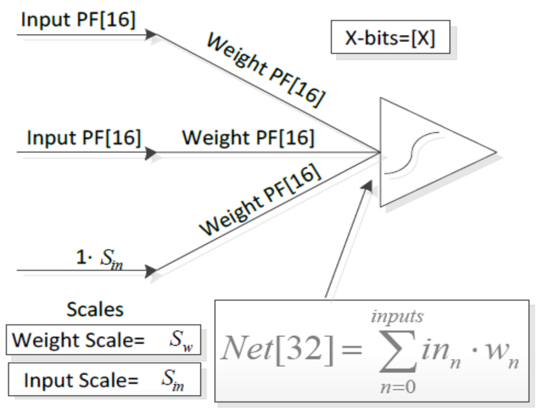
III. Neuron By Neuron Computation Process

3.1. Forward Calculations

This process of forward calculations is a unique method compared to most neural network implementations because it uses the Neuron By Neuron method. This method requires special modifications due to the fact that assembly language is used with very limited memory resources. The process is written so that each neuron is calculated individually in a series of nested loops; see Figure 14. The number of calculations for each loop and values for each node are all stored in two simple arrays in memory. The assembly language code does not require any modification to change the network's architecture. The only change that is required is to update these two arrays that are loaded into program memory. These arrays contain the architecture and the weights of the network and are generated by NNT. The weights are stored in ROM or off chip and are loaded into RAM for faster calculations. Finally there are numerous constants that are configured such as scale values and saturated neuron values. After the initialization block, the Main Loop begins. This is an infinite loop that keeps the network sampling new inputs and then starting the forward calculations. With the next input sampled the network resets pointers and index values and enters the Network Loop.

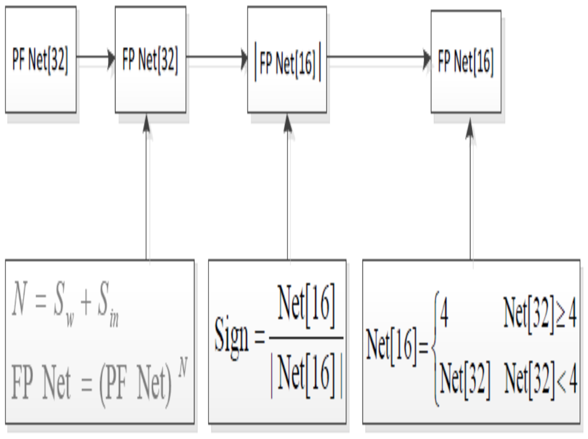
3.2. Individual Neuron Calculations

The Neuron calculations go through several steps in order to process the pseudofloating point arithmetic. The first step is the net value calculation which is shown in Figure 5.



Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

The inputs are multiplied by the corresponding weights and the result is stored in the 32-bit Net register. This is essentially a multiply and accumulate register designed for this particular stage. It is very important to keep all 32 bits in this stage for adding and subtracting. Without the 32 bits of precision at this step itwould be very easy for an overflow to occur during the summing process that would not be reflected in the final net value.The next stage is to turn the pseudo floating point number into a fixed pointnumber.
This process can be seen in Figure 6


PF Net[32]
FP Net[32]
|FP Net[16]|
FP Net[16]
$N = S_w + S_{in}$
$FP\ Net = (PF\ Net)^N$
$Sign = \frac{Net[16]}{|Net[16]|}$
$Net[16] = \begin{cases} 4 & Net[32] \geq 4 \\ Net[32] & Net[32] < 4 \end{cases}$

//Weights
Number of inputs; Number of outputs; (8-Bit)
Number of Weights; (8-bit)
Weight(1), Weight(2), Weight(3)....Weight(N);
Number of Neurons;
//Neuron 1
Neuron Scale, Number of Inputs, Output Node, Inputs(1-N)
//Neuron 2
Neuron Scale, Number of Inputs, Output Node, Inputs(1-N)
.
.
.
//Neuron N
Neuron Scale, Number of Inputs, Output Node, Inputs(1-N)
[Forward Calculations of Neurons]

IV. Application

In order to demonstrate that the microcontroller neural network is performingcorrectly several example control problems were tested. Neural networks have the unique ability to solve multi-dimensional problems with many inputs and many outputs, however these types of problems are not easy to test and verify visually. For this reason the network was tested mainly with two input and one output problems in order to plot the output as a function of the input on a three dimensional surface. This is not the only type of problem that can be solved, it is just to demonstrate. A two input and two output system is also shown by graphing the outputs separately to demonstrate that other types of networks will work as well.The process is tested with the microcontroller hardware in the loop. In otherwords, the sensor data is transmitted via the serial port from Mat lab to the microcontroller. The microcontroller then calculates the results and transmits this data via the serial port back to Matlab. The reason for this simulation is to isolate the errors in the system to those produced by the microcontroller calculations. The following examples will have some or all of the images that are described:

Training Data --- The training data is the data used to train the neural network.The number of points will vary with the application.

Ideal Neural Network -- This refers to a neural network running on a computeror a system using the IEEE floating point standard. The word ideal refers tomost practical applications where there is no significant data loss due to theprecision of the calculations. However, this is still a neural networkapproximation of the training data and not an identical representation.

PIC Based Neural Network -- This is the output of the neural network runningon the PIC hardware. This approximation will not be identical to the ideal neural network because of the approximations that are made on the microcontroller.

Error Surfaces -- The error surfaces are differences between two of thepreviously shown surfaces. The surfaces will give a visual description ofdifferences between surfaces shown on the same scale as the original surface.This comparison separates the error of using an ideal neural network and using a neural network with on the PIC.

Error Surfaces Tight -- These surfaces are the same as the error surfaces except on a much narrower scale to show what shape the errors have taken. This allows the user to identify problem areas or to confirm the error is evenly distributed.

Histograms -- The histograms show the errors of different surfaces in a numerical manner. This shows the distributions of the errors, in order to identify the distribution of the errors. The X-axis is the errors and the Y-axis is the number of data points within the corresponding error range.

V. Conclusion

The software offers the user the option of installing the network on a Microchip's 18Fxxxx series microcontroller using custom made neural network software written in assembly language and optimized for both the microcontroller and the neural network application. This version offers a very fast and accurate solution on a very inexpensive microcontroller. If the user prefers to use a different platform then the C code generated can be used to implement the trained network on any C capable platform. This can be used on other microcontrollers as well as PC based neural networks. This accomplishment demonstrates that neural networks can be used to solve problems that in the past would require custom software programs to be written for each problem. In other words, if three separate microcontrollers were needed to control three different processes for a single project then three unique programs would need to be written. This solution offers one standard solution for controlling all three. The user simply needs to train three separate networks, which is an automated process. Then the user has the solutions for unique problems without having to write code for the mathematics.

REFERENCES

[1] A. M. Zin, M. Rukonuzzaman, H. Shaibon, and K. I. Lo, "Neural network approach of harmonics detection," in Proc. Int. Conf. Energy Management and Power Delivery EMPD '98, 1998, pp. 467-472.
[2] H. C. Lin, "Dynamic power system harmonic detection using neural network," in Proc. IEEE Conf. Cybernetics and Intelligent Systems, 2004, pp. 757-762.
[3] S. Osowski, "Neural network for estimation of harmonic components in a power system," IEE Proceedings C Generation, Transmission and Distribution, vol. 139, pp. 129-135, 1992.
[4] Z. Jin and B. K. Bose, "Neural-network-based waveform Processing andDelayless filtering in power electronics and AC drives," Industrial Electronics, IEEE Transactions on, vol. 51, pp. 981-991, 2004.
[5] M. J. Embrechts and S. Benedek, "Hybrid identification of nuclear power plant transients with artificial neural networks," Industrial Electronics, IEEE Transactions on, vol. 51, pp. 686-693, 2004.
[6] L. Hsiung Cheng, "Intelligent Neural Network-Based Fast Power SystemHarmonic Detection," Industrial Electronics, IEEE Transactions on, vol. 54, pp. 43-52, 2007.
[7] H. C. Lin, "Intelligent Neural Network-Based Fast Power System Harmonic Detection," IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, vol. 54, pp. 43-52, 2007.
[8] W. Qiao and R. G. Harley, "Indirect Adaptive External Neuro-Control for a Series Capacitive Reactance Compensator Based on a Voltage Source PWM Converter in Damping Power Oscillations," IEEE Transactions on Industrial Electronics, vol. 54, pp. 77-85, 2007.
[9] B. Singh, V. Verma, and J. Solanki, "Neural Network-Based SelectiveCompensation of Current Quality Problems in Distribution System," IEEE Transactions on Industrial Electronics, vol. 54, pp. 53-60, 2007.
[10] S. S. Ge and W. Cong, "Adaptive neural control of uncertain MIMO nonlinear systems," Neural Networks, IEEE Transactions on, vol. 15, pp. 674-692, 2004.
[11] E. B. Kosmatopoulos, M. M. Polycarpou, M. A. Christodoulou, and P. A.Ioannou, "High-order neural network structures for identification of dynamical systems," Neural Networks, IEEE Transactions on, vol. 6, pp. 422-431, 1995.

Er. Jnana Ranjan Tripathy, Pusruing PhD in Centurion University of Technology and Management in “ANN Implementation in Embedded Systems” M.Tech in Computer Science,Berhampur University
B.Tech in Information Technology, BPUT, Currently working in Orissa Engineering College, Odisha, Worked at Centurion University previously. Member of IACSIT

Dr.Hrudaya Kumar Tripathy, Ph.D in Computer Science from Berhampur University. M.Tech in CSE from IIT, Guwahati, B.Tech (Ceramic Technology) from IIC (CG&CRI), Kolkatta, KIIT University
Chandrasekhpur, Bhubaneswar, Odisha. Published around 20 No.(s) of research papers in reputedinternational referred journals & IEEE conferences. Technicalreviewer and member of technical committee of manyInternational conferences.

Dr. S.S.Nayak, Dean, R & D, Centurion University of Technology & Management Published around 30 No.(s) of research papers in reputed international referred journals & IEEE conferences.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.