Implementation of the Hybrid Multilayered Perceptron Network in Assembly Language

Tan Kuan Liung[1] Mohd Yusoff Mashor[2] Ahmad Nazri Ali[3] Nor Ashidi Mat Isa[4]

[1]*Control and Electronic Intelligent Research Group,*
School of Electrical & Electronic Engineering, Universiti Sains Malaysia, Engineering Campus,
14300 Nibong Tebal, Penang, Malaysia.
E-mail: kuanliung@hotmail.com

Abstract

In this research, an approach to implement the Hybrid Multilayered Perceptron Network (HMLP) in assembly language is discussed in detail. After the code is assembled, it is loaded into an 8051 embedded system. This system then is tested against a proven HMLP based software. It is discovered that the system can produce 100% accuracy within test specifications.

Keywords:

Assembly Language, Cervical Cancer, HMLP Neural Network, MRPE Algorithm.

Introduction

The portability of a system is fast becoming a major factor in consumer electronics. Mobile computers, handphones, Personal Digital Assistants are fast replacing their bigger, heavier counterparts. The rationale behind this research, to implement the Hybrid Multilayered Perceptron Network in assembly language, is to provide an avenue for developers to design a portable HMLP microcontroller embedded system. This system can be used in a variety of situations, be it diagnosing cervical [1] and breast cancer, or detect the levels of carbon monoxide in the air. In this research, the implementation of the HMLP network in assembly language is discussed and verified against proven software.

Hybrid Multilayered Perceptron Network

A hybrid multilayered perceptron (HMLP) is an enhanced version of the multilayered perceptron (MLP) network. The proposed network allows network inputs to be connected directly to the input nodes via some weighted connections to form a linear model in parallel with the nonlinear, original MLP model [2].

A HMLP network with one hidden layer is shown in Fig. 1. HMLP network with one hidden layer can be expressed by the equation (1) [2].

$$\hat{y}_k(t) = \sum_{j=1}^{n_h} w_{jk}^2 F\left(\sum_{i=1}^{n_i} w_{ij}^1 v_i^0(t) + b_j^1\right) + \sum_{i=0}^{n_i} w_{ik}^2 v_i^0(t) \quad \text{for } 1 \le k \le m \qquad (1)$$

where w_{ij}^1, w_{jk}^2, w_{ik}^3 denote the weights between input and hidden layer, weights between hidden and output layer, and weights between input and output layes respectively. b_j^1 and v_i^0 denote the thresholds in hidden nodes and inputs that are supplied to the input layer respectively. F(·) is an activation function. A sigmoid function is normally selected as the activation function.

Modified Recursive Prediction Error Algorithm

The Modified Recursive Prediction Error Algorithm is used as the training algorithm of the HMLP network. It is based on structured learning error correction and is a modified version of the Recursive Prediction Error Algorithm (RPE). It converges at a smaller MSE value and has a faster convergence rate compared to its predecessor [3]. More information on this algorithm can be obtained in studies conducted by Mashor [2][3].

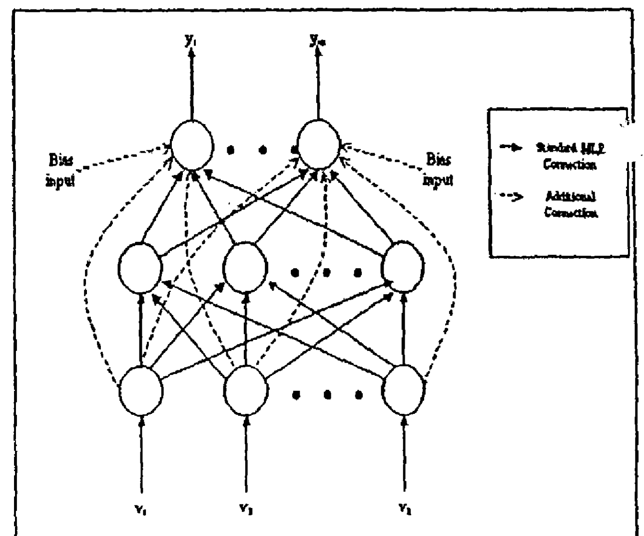

Figure 1- Hybrid Multilayered Perceptron Network

Binary Coded Decimal Floating Point Package

To compute HMLP network based calculations, a floating point mathematics package in assembly language is required. A Binary Coded Decimal (BCD) Floating Point Package, named fp-52.inc, is downloaded from reference

[4]. This package utilizes 6 bytes of memory space for each floating point number. Representation of each byte is shown in Fig. 2.

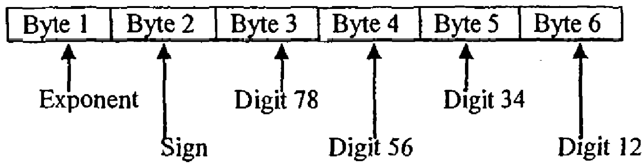

Figure 2 - Representation of Each Byte in a Floating Point Number

There are 5 main functions in this package. They are:

FLOATING_ADD: Perform an addition.

FLOATING_SUB: Perform a subtraction.

FLOATING_MUL: Perform a multiplication.

FLOATING_DIV: Perform a division.

FLOATING_COMP: Perform a comparison.

Before performing a calculation, 2 addresses of 6 bytes each has to be allocated in the register of the 8051 microcomputer. They are referred to as ARG_STACK_1 and ARG_STACK_2. When a floating point calculation is performed, ARG_STACK_1 will store Argument 1 whereas ARG_STACK_2 will store Argument 2. Then a function will be called to perform its related calculation (eg: if FLOATING_MUL is called, Argument 1 will be multiplied with Argument 2). The results of this operation will be stored in ARG_STACK_2.

Additional Function Algorithms

Besides the floating point mathematic package, additional functions has to be developed to ease the task of the assembly language programmer. They are:

ARRAY_ADD: Array addition.

ARRAY_MUL: Array multiplication.

ARRAY_DIV: Array division

ARRAY_ACC: Array accumulation.

FLOATING_MAC: Computation of $\frac{1}{1+e^{-x}}$.

ARRAY_MAC: Array computation of $\frac{1}{1+e^{-x}}$.

ARRAY_ADD, ARRAY_MUL, ARRAY_DIV, ARRAY_ACC and ARRAY_MAC allows the programmer to do multiple arithmetic calculations. FLOATING_MAC allows the programmer to compute the complex function of $\frac{1}{1+e^{-x}}$.

To use these functions, a few parameters are defined beforehand. They are:

DPTR_1: DPTR_2: DPTR_3:	Data pointers. Each data pointer
Counter_X Counter_Y Counter_Z Temp_Count_X	Counters. Each counter value is stored in 1 byte of register address

The algorithm and explanation of each function is given below:

Array_Add, Array_Mul, Array_Div

These are allow multiple arithmetic calculations. Prior to calling this function, the programmer has to set DPTR_1 to point to the address of the first argument in Array 1. On the other hand, DPTR_2 will point at the address of the first argument in Array 2. Next, DPTR_3 should be set to point at the location to store the results of the computations. Lastly, the number of computations required will be stor' in Counter_X and Counter_Y. The algorithm is as below:

- Copy the value in Counter_X to register address Temp_Count_X.
- Store the value of DPTR_1 in Temp_DPTR_1
- Transfer the value pointed by Temp_DPTR_1 to ARG_STACK_1.
- Transfer the value pointed by DPTR_2 to ARG_STACK_2.
- Call the function FLOATING_ADD, FLOATING_MUL or FLOATING_DIV.
- Transfer the computed results in ARG_STACK_2 to the address pointed by DPTR_3.
- Decrement Temp_Count_X.
- Repeat steps b-g until Temp_Count_X is 0.
- Decrement Counter_Y.
- Repeat Steps a-i until Counter_Y = 0.

Array_Acc

The function ARRAY_ACC allows the programmer to accumulate all the values stored in a range or array of memory addresses. The programmer must first set DPTR_1 to be pointing to the address of the first argument in the array. Counter_X and Counter_Y must also be set to store the number of computations required. The algorithm is as follows:

- Decrement Counter_X.
- If Counter_X = 0, end function.
- Set the value DPTR_3 = DPTR_1.
- Store the value of DPTR_1 in Temp_DPTR_1.
- Store the value of Counter_X in Temp_Count_X.
- Set the value of DPTR_2 = DPTR_1 + 1.
- Transfer the value pointed by Temp_DPTR_1 to ARG_STACK_1.
- Transfer the value pointed by DPTR_2 to ARG_STACK_2.
- Call the function FLOATING_ADD.

- Store the results in memory location pointed by DPTR_3.
- Set Temp_DPTR_1 = DPTR_3.
- Decrement Temp_Count_X.
- Increment Temp_DPTR_1 and DPTR_2.
- Repeat Steps g-m until Temp_Count_X = 0.
- Increment Counter_X.
- Set Counter_Z = Counter_X
- Set DPTR_1 = DPTR_1 + Counter_Z.
- Set DPTR_3 = DPTR_3 + 1
- Set Temp_DPTR_1 = DPTR_1.
- Decrement Counter_Y.
- Repeat Steps e-t until Counter_Y = 0.

Floating_Mac

In this study, the HMLP algorithm utilizes a sigmoid function as its activation function. The sigmoid function is expressed by equation (2).

$$y = \frac{1}{1+e^{-x}} \quad (2)$$

In assembly language, an approximation is needed for the variable e^{-x} prior to computation. Therefore, the Maclaurin series is used. The Maclaurin series is expressed by equation (3).

$$f(x) = f(0) + f'(0)x + f''(0)x^2/2 + f'''(0)x^3/3! + \ldots + f^{(n-1)}(0)x^{n-1}/(n-1)! + R_n \quad (3)$$

R_n is a remainder term, and can be written in the same forms as that of a Taylor series [5].

In assembly language, the algorithm to calculate $y = \frac{1}{1+e^{-x}}$ is implemented in the function FLOATING_MAC. The value of x is stored in ARG_STACK_2 prior to calling this function. The algorithm is as below:

- Store all Maclaurin parameters adjacent to each other, where the first parameter is stored in address M.
- Store the iteration value in a register address named Counter.
- Store the value of x in ARG_STACK_2 to a register address named Temp.
- Store the value of 1 in address A and address i..
- Store the value of x in memory location A + i.
- Set DPTR_1 to point at Temp. DPTR_2 to point at A + i and DPTR_3 to point at A + i + 1.
- Get the value pointed by DPTR_1 to ARG_STACK_1, value pointed by DPTR_2 to ARG_STACK_2.
- Call the function FLOATING_MUL.
- Store the results in ARG_STACK_2 to location pointed by DPTR_3
- Increment i.
- Decrement Counter.
- Repeat steps g - k until Counter = 0.
- Store the iteration value + 1 in Counter.
- Set DPTR_1 to point at A + i, DPTR_2 to point at M, and DPTR_3 to point at A + i.
- Call the function ARRAY_DIV.
- Set DPTR_1 and DPTR_3 to memory location A.
- Call the function ARRAY_ACC to accumulate the values. The accumulated value will be stored in A (memory location pointed by DPTR_3). The accumulated value is approximately the same as e^{-x}.
- Store the value 1 in ARG_STACK_1, and move the value e^{-x} from A to ARG_STACK_2.
- Call FLOATING_ADD to get $1+e^{-x}$.
- Store the value $1+e^{-x}$ in ARG_STACK_1 and the value 1 in ARG_STACK_2.
- Call FLOATING_DIV to get $\frac{1}{1+e^{-x}}$.

Array_Mac

This function allows the programmer to perform an array of calculations of $y = \frac{1}{1+e^{-x}}$ by just specifying the input and output memory locations. DPTR_1 points to the address of the first input. DPTR_3 points to the address to store the results of the computation. Counter_X will store the number of calculations required. The algorithm is as below:

- Transfer the value in DPTR_1 to ARG_STACK_2
- Call the function FLOATING_MAC.
- Transfer the computed value in ARG_STACK_2 to the address pointed by DPTR_3.
- Decrement Counter_X.
- Increment DPTR_1 and DPTR_3 by 1.
- Repeat until Counter_X is 0.

HMLP Implementation

The implementation of HMLP in assembly language is divided into 10 steps. Detailed algorithms of the steps are discussed below.

Step 1: Multiply the inputs with w^{l}_{ij}

- Set DPTR_1 to the address of the first input.
- Set DPTR_2 to the address of the first w^{l}_{ij}, w^{l}_{11}.

- Set DPTR_3 to the address of the first storage location.
- Set Counter_X = number of inputs.
- Set Counter Y = number of hidden nodes.
- Call ARRAY_MUL.

Step 2: Accumulate the multiplied values to get A_j.

- Set DPTR_1 to the address of the first multiplied value from Step 1.
- Set Counter_X = number of inputs.
- Set Counter Y = number of hidden nodes.
- Call ARRAY_ACC to get A_j.

Step 3: Add A_j with b_j^1 to get B_j.

- Set DPTR_1 to the address of the first A_j, A_1.
- Set DPTR_2 to the address of the first b_j^1, b_1^1.
- Set DPTR_3 to the address of the first storage location.
- Set Counter_X = the number of hidden nodes.
- Set Counter_Y = the number of outputs.
- Call ARRAY_ADD to get B_j.

Step 4: Find activation function of B_j to get C_j.

- Set DPTR_1 to the address of the first B_j, B_1.
- Set Counter_X = the number of hidden nodes.
- Call ARRAY_MAC to get C_j.

Step 5: Multiply C_j with w_{jk}^2.

- Set DPTR_1 to the address of the first C_j, C_1.
- Set DPTR_2 to the address of the first w_{jk}^2, w_{11}^2.
- Set Counter_X = the number of hidden nodes.
- Set Counter_Y = the number of outputs.
- Call ARRAY_MUL.

Step 6: Accumulate the multiplied values to get D_k.

- Set DPTR_1 to the address of the first multiplied value from Step 5.
- Set Counter X = number of hidden nodes.
- Set Counter_Y = number of outputs.
- Call ARRAY_ACC to get D_k.

Step 7: Multiply the inputs with w_{ik}^3.

- Set DPTR_1 to the address of the first input.
- Set DPTR_2 to the address of the first w_{ik}^3, w_{11}^3.
- Set DPTR_3 to the address of the first storage location.
- Set Counter_X = number of inputs.
- Set Counter Y = number of outputs.
- Call ARRAY_MUL.

Step 8: Accumulate the multiplied values to get E_k.

- Set DPTR_1 to the address of the first multiplied value from Step 7.
- Set Counter X = number of inputs.
- Set Counter_Y = number of outputs.
- Call ARRAY_ACC to get E_k.

Step 9: Add D_k with E_k to get F_k

- Set DPTR_1 to the address of the first D_k, D_1.
- Set DPTR_2 to the address of the first E_k, E_1.
- Set Counter_X = the number of outputs.
- Set Counter_Y = 1.
- Call ARRAY_ADD to get F_k.

Step 10: Compare F_k with threshold values to

- Set DPTR_1 to the address of F_1.
- Set DPTR_2 to the address of the threshold value.
- Call FLOATING_COMP.
- Check Carry flag.
- If Carry = 1, result = positive.
- If Carry = 0, result = negative.

After these 10 steps, the output, positive or negative, will be obtained.

Verification

An assembly language code, based on the above algorithm, is written and implemented in a 8051 embedded system. A proven HMLP software implemented in a [6], is used as a simulation software to verify the calculations given by the 8051 embedded system. The test is based on these specifications:

Number of data: 100 sets
Number of input nodes: 4
Number of hidden nodes: 1
Number of output nodes: 1

It is found out that the 8051 embedded system is 100% accurate with respect to the proven software [6].

Conclusion

The implementation of HMLP in assembly language is thoroughly discussed in this research. When a 8051 embedded system, implemented using this assembly

language, is tested against a proven HMLP simulation system based on certain specifications, it is found to be as accurate as the proven system. More testing should be done in the future to further verify the implementation of HMLP using the assembly language.

References

[1] Tan, K.L.; Mashor, M.Y.; Mat-Isa, N.A.; Ali, A.N.; and Othman, N.H. 2003. Design of a Neural Network Based Cervical Cancer Diagnosis System: A Microcontroller Approach. Proc. Of the 3rd International Conference on Advances in Strategic Technologies. Vol II, pp: 725-729.

[2] Mashor, M. Y. 2000. Hybrid Multilayered Perceptron Networks, International Journal of Systems Science, Vol 31, No 6, pp: 771-785.

[3] Mashor, M. Y. 1999. Nonlinear System Identification using HMLP Networks. Journal- Institution of Engineers, Malaysia, Vol 6. No 2. pp: 33-39.

[4] Aisnota. 2002. BCD Floating Point Math Package. Citing Internet source URL http://www.code.archive.aisnota.com.

[5] Cambridge University. 2001. Maclaurin Series. Citing Internet source URL http://thesaurus.maths.org/dictionary/map/word/808.

[6] Mat-Isa, N.A. 2002. Sistem Diagnosis Awal Pangkal Rahim Berasaskan Rangkaian Neural. PhD Disertation.