Neural Networks implemented on fixed point Digital Signal Processors

Adrian Spilca
Institute for Applied Research
University of Applied Sciences FH Ravensburg-Weingarten
Doggenriedstrasse 40-42, 88250 Weingarten, Germany
Phone: +49-751-501739, Fax: +49-751-48523
Email: aspilca@ars-sun1.ars.fh-weingarten.de

ABSTRACT: This paper describes an implementation of a Neural Network (NN) on Digital Signal Processors (DSPs). The NN, Multilayer Perceptron type, was trained off-line for pattern recognition, as part of a general purpose voice interface. Only the feed-forward propagation is implemented on the DSP. Time consuming for one layer is measured and compared with those resulted from implementations of the same NN on a general purpose PC-class computer. The DSPs used were Texas Instruments fixed point 2nd and 5th generation. C and assembly language implementation are explored.

KEYWORDS: Digital Signal Processor, Neural Network, forward propagation time, vocal commands.

INTRODUCTION

NEURAL NETWORKS

Neural Networks are adaptive tools more and more used in our days. Their major feature is the capability of learning. Because usually they are software implemented they have the disadvantage of being slow. Neuro-hardware also exists but it's still too expensive to be used for common applications. Implementing Neural Networks on Digital Signal Processors allow a faster response time and designing embedded systems for control, recognition or other applications.

It is well known that the *total input* of a unit ***i*** (sigma unit) is defined as being the following weighted sum:

$$i_i = \sum_{j=1}^{N_i} w_{ij} \cdot a_j + q_i \qquad (1)$$

where,

a_j - the state of activation (output) of the ***j***-th unit in the precedent layer (N_i units that have output connected to input of ***i*** unit);

w_{ij} - the weight of connection between units ***j*** and ***i***;

q_i - the bias or offset of unit ***i***.

The output of a unit, also known as activation, is obtained by applying, generally, some sort of threshold function to the total input. In this work it is used a *bipolar sigmoid with temperature*:

$$bipsig(x) = \frac{2}{1+e^{-\frac{x}{T}}} - 1 \qquad (2)$$

Due to the fixed point DSP the NN is implemented on, the activation function was tabled.

The NN implements a classifier trained off-line in MATLAB, using as inputs real data taken from microphone and real-time processed, like in Waibel (1989), in order to feed the NN.

DIGITAL SIGNAL PROCESSORS

Digital Signal Processors are microprocessor chips which was design especially to support numeric intensive procedures like signal processing algorithms. One of their basic feature is their ultrafast hardware multiplier and ability in accumulating products, and such products appear very often in mathematical support of neural networks (see *(1)*). They are fast and compact and this makes them very suitable for embedded applications.

Texas Instruments (TI) and other manufacturers provide powerful VLSI chips capable of implementing a variety of complex digital signal processing systems. The DSPs explored in this paper are all members of TI's fixed point TMS320 family, '5x, '54x and the new DSP microcontroller '24x.

THE FINAL GOAL

The final goal is to provide a voice interface being capable to recognise some spoken commands and possible to be connected in a simple way to any electronic device. The recognition task is not very hard, due to the small dictionary, isolated words, but the use of a single chip for audio signal processing and recogniser (NN) allows the manufacture of a *hand-held voice interface*. Also, the processor still has time to handle simple tasks like DC motor control.

This interface will be used in our lab to add intelligence to wheel-chairs for disabled people, for controlling the wheel-chair by voice or for assisting the disable person in some emergency cases.

IMPLEMENTATION

THE NN ARCHITECTURE

The Neural Network implemented was a *classifier* used to recognise 4 spoken commands. Experiences was done on the NN architecture presented in figure 1.

This structure was used to recognise 4 words : "sus", "jos", "stinga", "dreapta" (Romanian "up", "down", "left", "right"). The training process took only a few minutes on a Pentium machine, in MATLAB, with 100 training samples, 25 for each class, one male speaker. Recognition rate was 100% with a certainty of about 90%, on a 100 samples test file, different from those in the training file but taken from the same speaker. The training process is not described in this paper, due to the classical architecture used and existence of good quality MATLAB tools for this purpose. Improvements in this area are beyond the main goal.

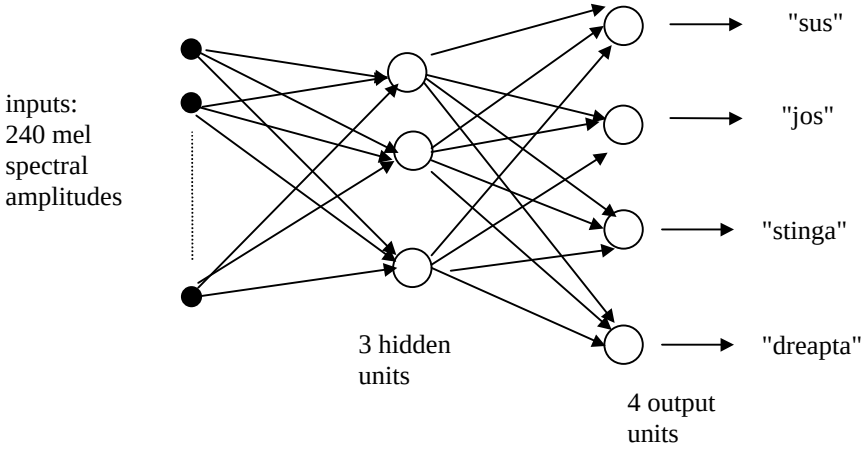

Figure 1. The classifier.

The input in the NN consists in 240 mel-scale coefficients obtained after the processing of a sample of voice in the following manner. 20 Hanning windows of 128 samples of signal in time domain was used to compute the Fourier transform. The spectrum obtained in this way was mel processed to get only 12 normalised mel coefficients for each window. In this way, some sort of frequency domain compression allows a reasonable number of input units for the NN. The process is described in detail in a previous paper, Spilcã (1996).

Three hidden units were enough for this task. Later work, with more than one speaker and many recognised commands showed that three units are no longer enough, neither the scarce data base used for training. The initial structure of the NN was not modified though, due to the fact that for the proposed goal – time measurements, this is irrelevant. There are anyway some provisions for increasing this number, taking into account the main problem - the limited amount of DSP memory.

The DSP NN outputs exactly the same results as the PC implementation.

THE PROGRAMS

High level - C

The relation *(1)* can be seen also as matrix multiplication. *Feed-forward* imply weights multiplied by previous layer activation. Remember, the weights were obtained after the training process.

The feed-forward C sequence to compute the activation of the *hidden layer*, considered for time interval measurements, is presented below. Note the fixed point implementation (only integers used), the floating point one is considered to be too straight to be presented here (simple two "for" instances in order to multiply weights by inputs and accumulate (see *(1))*, then the "exp" math function applied to get the sigmoid, see *(2*)).

```
for (i = 0; i < nh; i++)         // for each hidden unit
{
       templ = th [i];           // long (32 bits) value initialised with the bias
       templ <<= SHL ;           // shift left to meet Q10 fixed point requirements
       for (j = 0; j < ni; j++)  // consider each input
              templ += (long) wi[i][j] * ts[j];       // 32 bits multiply and accumulate
       temp = templ >> SCAL;     // back to 16 bits resolution
       temp += NSIG2;            // scaling for zero to be in the middle of bipsig table
       if (temp > NSIG) temp = NSIG1;   // adjust high to the biggest table index
       if (temp < 0)  temp = 0;         // adjust low
       ah [i] = bipsig [temp];          // read the activation from the table
}
```

where:

th - bias vector
wi - weights of connections between the input layer and hidden layer
ts - test vector (input to network)
ah - activation of hidden layer
ni - number of inputs
nh - number of hidden units
constants
SHL = 10, Q10 fixed point system; SCAL = 13, related to Q10 and bipsig (bipolar sigmoid) scale
NSIG = 1024, the bipsig number of elements; NSIG2 = 512, NSIG1 = 1023, related to NSIG

The sigmoid function was tabled in 1024 points - that means 1K from the precious DSP memory, but the time consumed to calculate the exact value of the activation using *(2)*, is unacceptably large. 10 bit resolution for the activation function was considered enough. This table handling causes some problems, like *saturation*. The *temp* value (index in the sigmoid table - *bipsig*) is prevented to overflow or underflow the table limits, so, the activation function output is saturated above to +1 and below to -1. Other activation functions, like *hyperbolic tangent* could also be used, or *linear* activation for *function approximation* tasks, you have only to generate a new table.

A floating point sequence was implemented also, for time comparison (see *Results* paragraph).

Low level – assembly language

The same algorithm was implemented in assembly language for three of the most used fixed point DSPs, namely ´5x, ´54x and ´24x. It follows closely the one presented above in C, only the most important issues will be emphasized.

```
´5x sequence first:

       LAR    AR4, #th            ; load bias address in index register AR4
       LAR    AR5, #wi            ; load weights address in index register AR5
       MAR    *, AR4              ; make AR4 the current index register
       LACL   #nh_1               ; load accumulator with constant
       SAMM   BRCR                ; store nh-1 in BRCR register for repeat block
       RPTB   lay1                ; repeat the next block BRCR+1 times for layer 1
              LACC *+, SHL, AR5   ; initialize the accumulator with bias
              RPT  #ni_1          ; repeat next instruction ni times
                     MAC ts, *+   ; multiply and accumulate wi x ts
              APAC                ; accumulate last product
.......................           ; scaling and preparing pointer on bipsig table
lay1:         TBLR   *+, AR4      ; table read output of activation function
```

The central attention should be placed on the RPT and MAC instructions that implement the accumulation of partial products wi * ts. This is also the main component of the consumed time for this sequence. '5x requires only ni+2 cycles for this matrix multiplication instead of 2*ni (both RPT and MAC takes 1 cycles to execute). Postincrement of index (*+ for current ARx) is also welcome and used.

´54x sequence next:

```
       blockrepeat(lay1)                ; repeat next block, nh times
              A = *AR4+ << SHL          ; initialize the accumulator with th
              repeat(#ni_1)             ; repeat next instruction, ni times
                     macp( *AR5+, ts, A) ; multiply and accumulate
.......................
```

The code for ´54x is straightforward after the above explained ´5x, even easier to understand due to the algebraic style. The initialization phase was skipped. *macp*, again the main issue, specifies that the test vector should be in program memory, for faster execution of this repeatable instruction.

For ´24x the sequence looks very much alike the ´5x one, the only difference consists in the absence of RPTB – repeat block instruction, another AR was used in a classic mode to implement the external loop. This slows down a little, but not significantly.

TIME INTERVAL MEASUREMENT RESULTS

The following table shows the processing time needed for 1 layer (the hidden one, 240 inputs, 3 hidden units). Anyway, this is the main time component of the feed-forward through all the NN.

Implementation:	*Time consumed [microsec]*
Floating point in C on PC (133 MHz)	900
Fixed point in C on PC (133 MHz)	414
Fixed point in C on PC (350 MHz)	65
Fixed point in ASM on DSP- ´5x	45
Fixed point in ASM on DSP- ´54x	37
Fixed point in ASM on DSP- ´24x	79
Fixed point in C on DSP- ´24x	2515

For DSP the time interval measurements were done using the hardware timer inside the DSPs (´5x and ´24x - precision 50 ns, ´54x - precision 25 ns). For PC, the time interval was measured using C functions. DSP Starter Kits (DSK) with ´C50 and ´C542 and Evaluation Module (EVM) for ´F240 was used.

First, you may notice that the DSP implementations are faster (except the ´24x one, but as you will see later , this latency is caused by subjective causes). This happens even considering the difference in clock period for PCs (Pentium processors) in one case significant – 350 MHz compared to 40 MHz for ´54x or even 20 MHz for the other two DSPs. It’s known that the C compilers for x86 processors are very well optimized, so the assembly language implementation on PC would not bring a significant change.

It’s not the same about the C compilers for DSPs, or other embedded systems processors. Compare the huge amount of time needed in the case of C implementation on DSP with the assembly language implementation on the same DSP. This happens due to the fact that the C compiler doesn’t use the complex MAC instruction, not even configured for the most powerful optimisation level, but simple MPY and ADD. Of course, this work is concerning only with TI’s DSPs and TI’s C compiler, so the observations applied only to this tools.

The explanation of the big difference for the two 20 MIPS DSPs (´5x and ´24x) is the use of the EVM for ´24x that loads the program into the external DSP memory. Even if there are no wait-states inserted, the execution of the repeat MAC takes 2n+2 cycles instead of n+2 that would take if the program would reside in internal DSP memory. Actually, it is enough to place the test vector *ts* (and *ah* for output layer) in internal memory. In this case the time consumed for hidden layer propagation is close to the one measured for ´5x. The problem is that ´24x provides only 256 words of internal program RAM (block B0) and this is close to the limit with this poor NN architecture (240 *ts* words and 3 *ah*)

CONCLUSIONS

The NN implementation achieved the same recognition accuracy for both PC and DSP implementations, while the DSP NN performs faster (more than 10 times faster, if you consider similar clock conditions, or integration technology – Pentium II is the ultimate state-of-the-art Intel product, there are TI fixed point DSPs, from the 6th generation, 1200 MIPS, not explored in this paper).

Take into account also the size of a DSK board (12 cm x 7 cm, under 1 cm height) and the one of a Pentium II with the cooling installation, about the same surface, much ticker, for only the processor. The power consumed and the price makes the DSP implementation attractive for embedded control.

Another conclusion, once more revealed, is that a task like forward propagation through a Neural Network implemented on a Digital Signal Processor should be done in assembly language to fully exploit the features of the processor.

ACKNOWLEDGMENTS

This work was done under a bilateral Romanian-German research contract between University “Politehnica” of Timisoara and University of Applied Sciences FH Ravensburg-Weingarten, “Implementation of common artificial intelligence applications on mobile robots”.

REFERENCES

Kröse B., van der Smagt P., “An introduction to Neural Networks”, University of Amsterdam, 1994.

Spilcã A., “Neural Networks running on Digital Signal Processors”, 10th International Conference on Control Systems and Computer Science, CSCS 10, Bucharest, 1995.

Spilcã A., Pragay A., “Partially-connected topology Neural Network for Speech Recognition”, ConTi'96 – International Conference on Technical Informatics, Timisoara, 1996.

Texas Instruments, “TMS320C50 User's Guide”, 1993.

Waibel A., Hanazawa T., Hinton G., Shikano K., Lang J., “ Phoneme Recognition Using Time-Delay Neural Networks”, IEEE Transaction on Acoustics, Speech and Signal Processing, 1989.