NM6403 Software Development Kit

Application Report

Neural network performance evaluation test

Written by Mikhail Zabaluev

JOINT-STOCK COMPANY
RESEARCH CENTRE

Contents

Introduction

This document covers the software implementation of a neural network on the NeuroMatrix® NM6403 processor [1]. The program is provided as an example. One of its goals is to show the effective approach to implement the neural networks functionality on NM6403 processor; the other is to measure the processor's performance in the tasks of such class. The sample neural net [4], the two-layered perceptron that is processed in forward-propagation scheme, has the following parameters:

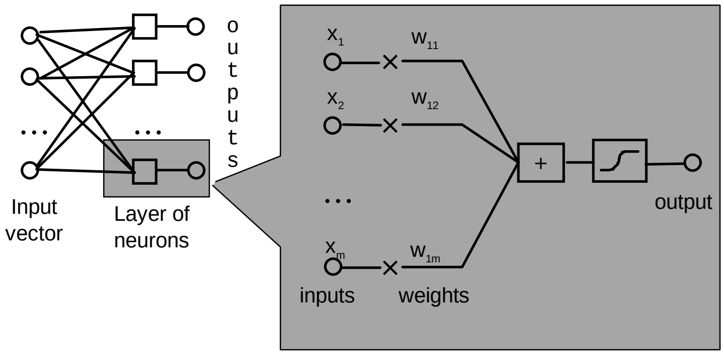
- the net has got 12 inputs, one hidden layer of certain (about hundreds) number of neurons and the output layer of 12 neurons;
- input and output values of the neurons are represented as 64-bit signed integers;
- weight coefficients of the neurons are represented as 16-bit signed integers, packed in fours into 64-bit words;
- the output characteristic of a neuron (that is, the dependence of the output value from a weighted sum of the inputs) is defined as an integer function with a non-linear (sigmoid) real prototype, whose output values lay in range from -32767 to 32767. At the argument values less than -32768 or greater than 32767, the function takes the margin values, accordingly -32767 or 32767.

The computational performance of the program is the subject of measurement. All the time-critical procedures are written in assembly language and have been hand-optimized for maximum processing speed.

Brief description of neural network functioning

A multilayered perceptron-like neural net consists of several layers of uniform *neurons* (Fig. 1). A neuron can be represented as a numeric (scalar) function of a vector of numeric input values. The neuron output is calculated as follows: the input values are multiplied by corresponding *weight coefficients* and these products are added together. The obtained sum is used as an argument for certain function, called *threshold function* or *activation function*, to obtain the neuron's output value. As a rule, this function is chosen non-descending and bound in output value to a certain range. In order to achieve better trainability when using gradient methods, the function (or its real-value prototype in case of discrete values) is often chosen to be smooth and non-linear. The neurons of the same layer of perceptron share one vector of inputs. Tuning of weight coefficients called training changes the network output behavior.

Fig. 1. The scheme of a neural network layer



Calculation of weighted sums

The vector-processing unit of the NeuroMatrix®NM6403 processor [1] is designed to execute a weighted summation (in fact, the vector-by-matrix product), which is a common operation for matrix calculations. In our example, the configuration of the Active Matrix of NM6403 is one column and four rows filled with 64-bit words of input data:

```
rep 4 wfifo = [ ar4++ ], ftw;
...
wtw;
```

The corresponding chunks of weights matrix of four 16-bit values packed into 64-bit word are multiplied by these values with respect to sign. The four products are summed to a 64-bit value that is added to the already accumulated sum (Fig.2). These four multiplications and the summation of five values are performed during one step of vector operation which, being multi-step, calculates several (up to 32) such partial sums for several neurons due to the fact that the input vector is the same for every neuron in the hidden layer:

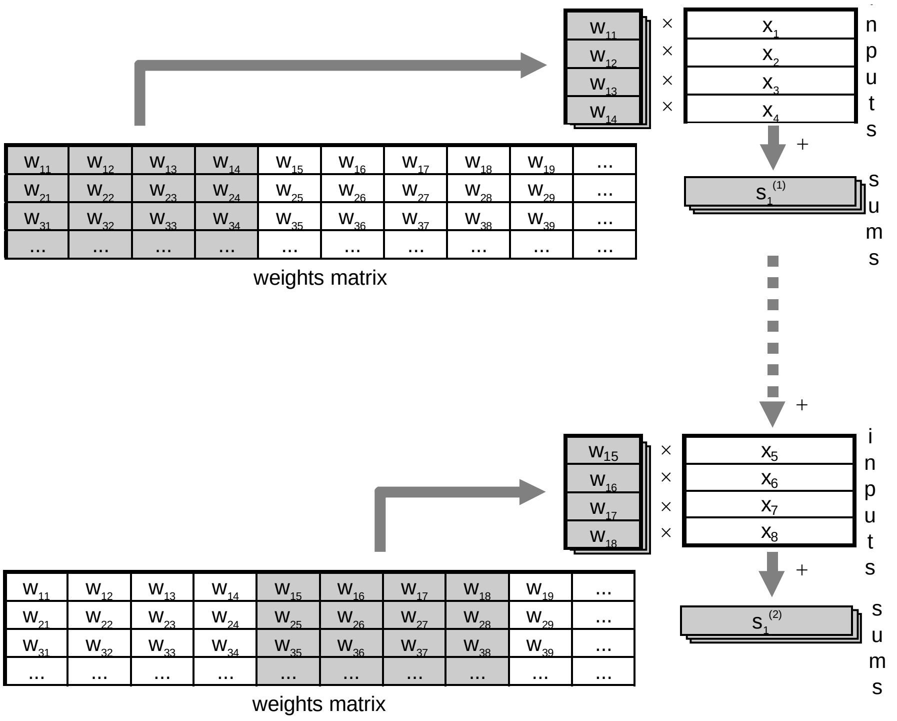
```
rep 32 data = [ ar0 += gr0 ] with vsum ,data, 0;
```

(for the first four input values),

```
rep 32 data = [ ar0 += gr0 ] with vsum ,data, afifo;
```

(for the forthcoming input values, adding accumulated sums left by the previous such operation in the results queue).

Fig.2. The weighted sums calculation scheme



In order to avoid overheads for saving and restoring of partial sums from memory, the sums of all the neurons processed by one such operation are added to the result of the next iteration and do not leave the vector device. Thus, consequently changing fours of input values in the multiplier matrix[1], complete weighted sums are calculated for, say for certainty, 32 neurons. During the next iteration, the sums of 32 more neurons are calculated, and so on until the neurons of the whole layer are processed.

Calculation of threshold function

One would note that the threshold function becomes nearly constant for the arguments with large magnitude. Therefore we can select a range of integer argument values and retrieve function values within this range from a table. The arguments that are out of the range deliver the marginal function values. The hardware saturation function that is implemented in NeuroMatrix® NM6403 is suitable to truncate values to the range:

```
rep 32 with activate afifo + 0;
```

The control register `f1cr` is initialized such that saturation function replaces the values that are less than -32768 or greater than 32767 with -32768 or 32767 accordingly, and leaves the values from -32768 to 32767 unchanged. After hardware saturation is applied we use table of 65536 numbers to

---

[1] Matrix loading is an expensive operation but it has to be performed in order to avoid more serious expenses for the saving/restoring of partial sums; plus, it mostly executes in parallel with weighted summation.

retrieve function values corresponding to all the possible arguments. The table is filled with sigmoid function values converted to integers:

$$f = 32767 \cdot \frac{1-e^t}{1+e^t}, t = -\frac{x}{2048}$$

, where x runs the range from -32768 to 32767.

The evaluation of the neural network is implemented in three procedures, which have been implemented in the assembly language [2, 3], with C-callable interface. They are called by the main C++ program routine. The C++ prototypes of these calls along with the descriptions of their work are listed below.

```
void SumLayer ( size_t nInput, size_t nNeurons,
                const long *input,
                const unsigned long *weights,
                long *output );
```

This procedure calculates a vector of weighted sums for a neuron layer of arbitrary dimension. The accumulated sums are then truncated to the range -32768…32767 by the hardware saturation function. The `nNeurons` parameter gives the number of neurons in the layer, `nInput` gives the number of layer's inputs. The `input` pointer refers to an array of 64-bit input values, `weights` points to the beginning of a matrix of 16-bit weights packed by fours into 64-bit words, `output` points to an array of 64-bit weighted sums calculated by this procedure. Due to the multi-step manner of the vector instructions execution, there exist some alignment requirements for sizes of arrays in memory. These requirements are met with padding by zero elements if necessary. The size of the input array should be at least `nInput` elements and be a multiple of four 64-bit words. Each row of the 16-bit weight values (every such row contains weights of one neuron) takes the whole number of 64-bit words; the number of these rows, as long as the size of the output array, shall be at least `nNeurons` and be a multiple of 32.

```
void SumLayer12 ( size_t nInput,
                  const long *input,
                  const unsigned long *weights,
                  long *output );
```

This procedure is similar to `SumLayer` except that the number of neurons equals 12, so the parameter `nNeurons` is omitted. This performance-twisted layer-specific version allows to get rid of the external loop that was used to iterate packs of 32 neurons. Twelve-repetition operations are used to calculate weighted sums. Accordingly, both the number of weight matrix rows and the number of outputs are 12.

```
void ApplyNeuroFunc ( size_t nInput,
                      const long *input,
                      long *output );
```

This procedure applies the threshold function to calculated weighted sums of a neuron layer. The weighted sums, as mentioned above, are in the range from -2^{15} to $2^{15}-1$. The table of 65536 values is used to calculate the function result.

Listed below is the `main()` routine which procedures that implement the neural net of particular configuration, along with used constants and declarations.

```
    // number of neurons in the hidden layer
static const size_t nHidden = 1024;


extern "C" {
    long input[];
    long hidden[];
    long output[];
    unsigned long weights1[];
    unsigned long weights2[];
}


int main ()
{

        // initialize data

    Init();

        // get the starting clock value

    clock_t t1 = clock();

        // evaluate the sums of the first layer

    SumLayer( 12, nHidden, input, weights1, hidden );

        // evaluate the outputs of the first layer

    ApplyNeuroFunc( nHidden, hidden, hidden );

        // evaluate the sums of the second layer

    SumLayer12( nHidden, hidden, weights2, output );

        // evaluate the outputs of the second layer

    ApplyNeuroFunc( 12, output, output );

        // get the final clock value

    clock_t t2 = clock();

        // return elapsed processor clocks

    return t2 - t1;
}
```

The net consists of two layers. The first one called hidden has 12 inputs and the neurons number defined by `nHidden`. The outputs of the hidden layer are the inputs of the second, or output, layer, that consists of 12 neurons. The number of cycles spent to calculate signal propagation through two layers is measured using `clock()` function.

The main routine is contained in main.cpp file. Assembly sources of functions described above are in layers.asm, their C++ prototypes are in layers.h. The nfunctab.asm file contains the threshold function table. The data.asm file contains declarations of data buffers that are used in the program.

Table 1 contains timing results returned by the program for two different `nHidden` values. The program was run on NM6403 processor of dual-processor NM1 board at 40 MHz.

Table 1. Time of signal propagation through the net for various sizes of the hidden layer.

Number of neurons in hidden layer (nHidden)	Equivalent number of element multiplications	Number of cycles spent	Elapsed time in milliseconds
512	12288	13150	0.33 msec
1024	24576	25800	0.65 msec

Literature

1. RC Module. NeuroMatrix® NM6403. Architectural overview. http://www.module.ru/files/archover.pdf
2. RC Module. NeuroMatrix® NM6403 SDK. Assembly language overview (preliminary version).
3. RC Module. NeuroMatrix® NM6403 SDK. Programmer's guide (preliminary version).
4. Fausett, L. Fundamentals of Neural Networks: Architectures, Algorithms, and Applications. Englewood Cliffs, NJ: Prentice Hall, 1994, ISBN 0-13-334186-0.

Research Centre Module
Box: 166, Moscow, 125190, Russia
Tel: +7 (095) 152-9335
Fax: +7 (095) 152-4661
E-Mail: postmast@module.ru
WWW: http://www.module.ru

Printed in Russia Release date: 1999 March, 29