EFFICIENT DIGITAL PRE-TRAINED NEURAL NETWORK IMPLEMENTATION

Tamás Szabó, Gábor Horváth

Technical University of Budapest, Dept. of Measurement and Information Systems,
H-1521. Budapest, Műegyetem rkp. 9, Bldg. R. I./113. Hungary,
Tel.: +36 1 463 2057, Fax.: +36 1 463 4112,
E-mails: [szabo, horvath]@mmt.bme.hu

ABSTRACT

In many applications, like nonlinear filtering in embedded measuring systems the most significant advantages of neural networks come mainly from their parallel architectures. In these cases neural networks can be applied only if they are really implemented in parallel hardware form ensuring rather high-speed operation. The difficulties of parallel digital hardware implementation mostly come from the high complexity of the parallel many-multiplier structure and the need for implementing the nonlinear squashing functions for all neurons. Efficient realization of both basic elements may help to implement rather complex neural networks with many neurons. This paper suggests a new bit-serial full parallel neural network implementation method for pre-trained networks. The method makes possible significant hardware cost savings. The proposed approach - which is based on the results of a previously suggested method for efficient implementation of digital filters - uses bit-serial distributed arithmetic. The efficient implementation of a matrix-vector multiplier is based on an optimization algorithm which utilizes the advantages of CSD (Canonical Signed Digit) encoding and bit-level pattern coincidences. The resulting architecture performs full-precision computation and allows high-speed pipe-line operation. The proposed approach seems to be a promising one for FPGA and ASIC realization of pre-trained neural networks and can be integrated into automatic neural network design environments.

1. INTRODUCTION AND MOTIVATIONS

Neural networks can provide efficient solution to many signal processing problems, e.g. nonlinear adaptive filtering, principal and independent component analysis, clustering, etc. One of the main obstacles of the spread of neuro-applications is the lack of suitable implementation medium. A very wide scale of different neuro-implementations can be found; we can find devices from analog neurochips through large parallel general purpose computers - like Warp or Connection Machine - to tricky optical devices. However, there are only few examples for compact implementations in which the given neural network operation is performed by a single or only some chips. These compact solutions make possible that NNs can be embedded into instruments.

All these different (analog, digital or hybrid) implementations have their own advantages and disadvantages. Analog hardware provides high operating speed, but carries some problems that are hard to overcome (e.g. precision or stability issues). Software implementations usually have not high enough operating speed and/or run on large hardware. Dedicated digital hardware can be plausible solution, because it has arbitrary precision and dynamic range, provides high noise immunity, further they are relatively easy to design. Mature CAD systems support its design and fast prototyping (for instance on programmable devices like FPGA or ASIC). Unfortunately, one of the main drawbacks of the digital neuro-implementations is the large hardware cost (chip surface) required by multipliers.

As it is well known, most of the NNs have the following basic structure in the recall phase:

- the whole NN can be divided into similar layers composed of neurons,
- a neuron consists of a linear combiner followed by a non-linear activation function.

This work was supported by the Hungarian Fund for Scientific Research (OTKA) under contract T021003

Thus, a layer in a pre-trained NNs can be considered as a general function (activation function $f(\cdot)$) of a matrix-vector product, in which the coefficient matrix is constant (it is determined during training). For the ith layer:

$$\mathbf{y}^i = f(\mathbf{W}^i \mathbf{x}^i) \tag{1}$$

One of the main difficulties of NN realization is to implement these matrix-vector multiplications efficiently. In this paper firstly we present such methods which help to synthesize hardware saver matrix-vector multipliers. We focus on such cases where constant matrices are used, that is we are dealing with the efficient implementation of pre-trained (fixed) networks. The efficient implementation of matrix-vector multipliers is based on an optimization algorithm which utilizes the advantages of CSD (Canonical Signed Digit) encoding and bit-level pattern coincidence. The resulting architecture performs full-precision computation and allows high-speed pipe-line operation. The details are discussed in section 3.

2. NEURAL NETWORK IMPLEMENTATION

One of the most essential property of NNs is their inherent parallelism. Mainly this property is responsible for their high computational power and speed. The computation of a given layer can be performed by neuron-parallel or neuron-serial ways; the computation of the synapses can be also done in serial (so called Multiply-ACCummulate structure) or parallel manner (weight parallelism). To exploit the advantages of NNs the full parallel implementation is the most favorable one.

Besides multipliers, another bottleneck of the hardware realizations of NNs is the high number of connections, because the implementation of wires consumes large chip surface. Bit-serial communication and computation will help not only to reduce the number of basic arithmetic elements, but to keep the number of connections tractable. This approach decreases the number of wires and arithmetic elements about one order of magnitude (depending on the precision of the computation) but unfortunately, decreases the operation speed, too. Nevertheless, as it can be seen later, the operation speed of a structurally full-parallel NN implementation approach balances this speed fall.

Our goals can be summarized as it follows: given a matrix-vector multiplier which operates in such a way, that the elements of the input vector $\mathbf{x}^i$ are presented and processed in parallel manner and thus all elements of the output vector are also available simultaneously. The results of the matrix-vector products are passed through the piecewise linear activation functions. The proposed algorithm provides efficient method to synthesize the matrix-vector multipliers and squashing functions to save as many hardware as possible.

3. THE PROPOSED MATRIX-VECTOR MULTIPLIER SYNTHESIS METHOD

The idea of efficient implementation of a matrix-vector multiplier is based on the *"do not implement unless it is necessary"* rule. This means the followings: given some - for instance binary - representation of the element of a coefficient matrix,

- if there are zero bits in this representation, the corresponding parts of a serial multiplier are needless to implement,
- if there are similar bit patterns, it is enough to implement the pattern and the corresponding part of the multiplier only once.

Such an optimization method for vector-vector (inner product) multiplication was originally presented in [1]. In this solution the constant coefficients are not stored explicitly, but they are "built" into the structure. This kind of approach is "much more distributed" than those ones in the literature of Distributed Arithmetic (DA). (The idea of the DA was originally arisen in the field of digital filter implementation as it is reviewed in [4].) A two-step version of this approach was introduced in [2]. That method first reduces independently the elementary vector-vector products of a matrix-vector product and, as a second step, it makes some more reduction in the resulting graphs by merging their identical parts. This two-step algorithm often results in local optima. This paper deals with the generalization of the original method. The proposed algorithm looks for a single-step optimal reduction in the whole matrix-vector product.

3.1. Analysis of the matrix-vector multiplication operation

Assume that an NN has N inputs and M outputs ($\mathbf{W}$ is an M by N matrix). The $\mathbf{y} = \mathbf{W}\mathbf{x}$ matrix-vector product can be written as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,N} \\ w_{2,1} & w_{2,2} & \dots & \\ \vdots & \vdots & \vdots & \vdots \\ w_{M,1} & w_{M,2} & \dots & w_{M,N} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \tag{2}$$

The *i,j*th coefficient $w_{i,j}$ can be rewritten in binary representation $\begin{bmatrix} h^i_{j,1}2^0 & h^i_{j,2}2^{-1} & \dots & h^i_{j,Bc-1}2^{-B_c-1} \end{bmatrix}$ where $h^i_{j,k} \in \{0,1\}$. From this it follows that for the *i*th output $y_i = \sum_{l=1}^{N} w_{i,l}x_l$ can be formed as :

$$y_i = \mathbf{x}^{\mathbf{T}}\mathbf{H}^i \tag{3}$$

where $\mathbf{H}^i =$

$$\begin{bmatrix} h^i_{1,1}2^0 & h^i_{1,2}2^{-1} & \cdots & h^i_{1,B_c-1}2^{-B_c-1} \\ h^i_{2,1}2^0 & h^i_{2,2}2^{-1} & \cdots & h^i_{2,B_c-1}2^{-B_c-1} \\ \vdots & \vdots & & \vdots \\ h^i_{N,1}2^0 & h^i_{N,2}2^{-1} & \cdots & h^i_{N,B_c-1}2^{-B_c-1} \end{bmatrix} \tag{4}$$

In this manner the *i*th output y_i (corresponding to the *i*th row of $\mathbf{W}$) can be considered as a special vector-matrix product, in which the elements of the j-th row of
$\mathbf{H}^i_j = \begin{bmatrix} h^i_{j,1}2^0 & h^i_{j,2}2^{-1} & \dots & h^i_{j,Bc-1}2^{-B_c-1} \end{bmatrix}$
are the binary representation of the corresponding $w_{i,j}$. The obvious implementation of this product is the application of N serial-parallel multipliers followed by an adder tree. On closer examination such a multiplier is composed of one-bit adders and bit-delay elements (D flip-flops). Moreover, it can be found that its complexity depends on the number of non-zero elements of $\mathbf{H}^i$.

At this point significant hardware cost reduction can be achieved if the well known Canonical Signed Digit (CSD) encoding is used (see [3]). In CSD codes instead of the $\{0,1\}$ values of the binary representations there will be three possible values: $h^i_{j,k} \in \{0,+1,-1\}$. In this case the maximum number of non-zero elements in a CSD encoded number is $B_c/2$ (where B_c is the word length). Moreover, its statistical average is $B_c/3$. As it can be seen easily the usage of ± 1 values in hardware realization does not mean any extra cost because the complexity of a bit-serial adder and a substractor is the same.

The second idea is to reduce the occurrence of identical bit patterns in the CSD representation of the $w_{i,j}$-s. This reduction can be achieved if the distributive property (Eq. 5) of the multiplications is exploited. That is:

$$\begin{aligned} (p_{i,\alpha} + p_{i,\beta})x_n &= p_{i,\alpha}x_n + p_{i,\beta}x_n \\ p_{j,\gamma}x_n + 2^D p_{k,\gamma}x_m &= p_{j,\gamma}(x_n + 2^D x_m) \end{aligned} \tag{5}$$

where $p_{i,\alpha}, p_{i,\beta}, p_{j,\gamma}, p_{k,\gamma}$ are sub-polynomials of the CSD representation of p_i, p_j, p_k coefficients respectively, such that $p_i = p_{i,\alpha} + p_{i,\beta}$, $p_{j,\gamma} \subseteq p_j, p_{k,\gamma} \subseteq p_k$, and $p_{j,\gamma} = 2^D p_{k,\gamma}$. If in a $p_i x_n$ partial product p_i can be divided to the sum of its partial polynomials $p_i = p_{i,\alpha} + p_{i,\beta}$, than the product can be computed using these partial polynomials as it is shown by the first line in Eq. 5. The second line states that the identical partial product groups can be merged.

3.2. The optimization algorithm

For better understanding of the algorithm a simple demonstrative example will be presented. Let us examine a part of a matrix-vector product. Let the *i*th row of the matrix ($N = 4$) $\mathbf{w}_i = 2^{-7}[-112, 78, 89, 47]$ and $B_c = 8$, thus

$$
\mathbf{H}^i = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{-1} & 0 \\ 1 & 0 & \mathbf{-1} & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{6}
$$

As it can be seen the second and the third lines of $\mathbf{H}^i$ contain the same $\{1\,0\,1\,0\,0\,-1\}$ pattern, disregarding from their different positions and signs. If a unit-delay is used in order to shift the second row of $\mathbf{H}^i$ (this means multiplication by 2), a new composite signal $x_5 = 2x_2 - x_3$ can be generated by a pre-adder, which will be multiplied by the common bit pattern. This means that the coinciding patterns can be removed from the second and the third rows:

$$
\begin{aligned} y_i = {} & [x_1, x_3, x_4, x_5]* \\ & \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix} \end{aligned} \tag{7}
$$

The modified $\mathbf{H}^i$ contains the newly introduced x_5 but x_2 and the second row of $\mathbf{H}^i$ can be dropped out because x_5 fully absorbed it. Certainly, x_2 can be dropped out only-if the corresponding rows are empty in all of the $\mathbf{H}^i$-s at least after the current iteration.

As it is discussed later, in order to achieve high operation speed pipe-line bit-serial arithmetic have to be used. This is why the digits in the new row of $\mathbf{H}^i$ are shifted right by one position to compensate the inherent one-bit-delay of the pipe-line serial operator.

The question is: how to decide which rows should be merged. To help this decision statistical matrices are generated in every iteration step. The statistical matrices $\mathbf{S}^i n(l, m)$ contain the largest number of non-zero elements of coinciding identical (contradicting) bit patterns from the l-th and the n-position right-shifted m-th rows of the actual $\mathbf{H}^i$ matrix. The diagonals of $\mathbf{S}^i n$ are cleared to ignore self coincidences.

$$
\begin{aligned} \mathbf{S}^i 0 &= \begin{bmatrix} 0 & 1 & -1 & -1 \\ 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}; \\ \mathbf{S}^i 1 &= \begin{bmatrix} 0 & 0 & -1 & 0 \\ -1 & 0 & -1 & 0 \\ -1 & -3 & 0 & -1 \\ -1 & 1 & 2 & 0 \end{bmatrix}; \\ \mathbf{S}^i 2 &= \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} \end{aligned} \tag{8}
$$

For instance, let us see $\mathbf{S}1(3, 2)$ in Eq. 8. The second $(m = 2)$ and the one-position $(n = 1)$ right-shifted third $(l = 3)$ row of the initial $\mathbf{H^i}$ (Eq. 7) must be considered in this case. Eq. 9 shows the largest coinciding bit pattern of these rows (with one-position shift).

$$
\begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 & -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{9}
$$

As it is seen there are three contradicting non zero bits in these patterns. Consequently $\mathbf{S}1(3, 2) = -3$ as it is in Eq 8. The positive elements in $\mathbf{S}n(l, m)$ mean coinciding, while the negative elements mean contradicting bits in the corresponding patterns.

The cost of an iteration step is $c_{step}(n,m) = 1 + \Delta D_m * C$, where ΔD_m is the delay difference between the necessary and the existing delay of the x_m signal. C is a technology dependent factor, namely the cost ratio of the delay and the arithmetic operations (D flip-flop and a pipe-line serial adder/substractor). This parameter controls the algorithm. The available maximum gain is determined by the $\mathbf{S}^i n$ statistical matrices and the cost of the iteration step $c_{step}(n,m)$. If only this (the *i*th) row of $\mathbf{W}$ is considered than the maximum gain is $\max_{n,l,m}(\mid \mathbf{S}^i n(l,m) \mid - c_{step}(n,m))$.

To compute the maximum gain for the whole matrix-vector multiplication and not only for the mentioned $\mathbf{H}^i$, the corresponding statistical matrices $\mathbf{S}^i n$, $i = 1...M$ (where M is the number of elements in the output vector $\mathbf{y}$) must be summed. Thus, the global maximum gain is $\max_{n,l,m}(\mid \sum_{i=1}^{M} \mathbf{S}^i n(l,m) \mid - c_{step}(n,m))$.

The necessary number n_{max} of statistic matrices $\mathbf{S}^i n$ above which the available gains are surely smaller than the maximum gain is a parameter of the algorithm which dynamically changes during iterations. It is a finite number because the number of non-zero digits in CSD representation is finite. Its computation is detailed in [1].

The algorithm goes to the direction in every step where the gain is maximal. This means that it moves toward the local optima of the iteration step. This property is similar to that of the well known steepest descent method. Although there is no guarantee that the algorithm will find the global optimum after the iterations, it results in a significant hardware cost reduction as it can be seen later. However, finding the global optimum is an NP-complete problem.

This iteration process continues until the global maximum gain is not smaller than one. During the execution of the algorithm the non-zero digits of $\mathbf{H}^i$ are continuously eliminated and the arithmetic functions are executed in the pre-addition phases. This procedure ”builds the coefficients into the structure”. The convergence of the algorithm is guaranteed in finite number of step, because the maximum number of non-zero bits in CSD representation is $B_c/2$ thus, the maximum number of non-zero digits in $\mathbf{H}$ is $N * B_c/2$.

3.3. Concluding remarks and some realization issues

The distributed realization can be applied in all areas of digital signal processing where matrix-vector multipliers are used. Such an application field is the implementation of the various pre-trained NNs.

The proposed method significantly reduces the hardware cost of the implementation of matrix-vector multipliers. It provides full precision even in the computation of the sum of the partial products. Its operating speed can be kept relatively high because:

- the architecture of the resulting distributed matrix-vector multiplier allows pipe-line operation,
- although the reduction destroys the regularity of the structure, the average fan-out is decreased in average in the reduced structure, allowing higher clock-rate.

This approach is very suitable for FPGA and ASIC implementations. The proposed algorithm was implemented in C and an example implementation was designed on Xilinx XC4000 FPGA family. In this environment the Configurable Logic Blocks (CLBs) consist of two combination units and two flip-flops. A bit-serial arithmetic unit (an adder, a substractor) can be realized by a single CLB with the same cost, while the implementation of the pipe-line flip-flops does not require any extra cost. The cost ratio between the implementation of an adder and a delay element is $C = 0.5$. Experiences show that although the necessary routing has lost its regularity this did not make any significant difficulties in FPGA routing. The average wire length decreases which allows higher achievable bit-rate for both FPGA and ASIC technologies.

N	**M**	C_{TRI}	C_{RED}	**R %**
10	5	222	107	48.2
10	10	445	224	50.3
20	10	895	383	42.8
20	20	1790	677	37.8
40	10	1795	664	37.0

Table 1: Reduction example, $B_c = 8$

N	M	C_{TRI}	C_{RED}	R %
10	5	322	165	51.2
10	10	645	317	49.1
20	10	1295	534	41.2
20	20	2590	989	38.2
40	10	2595	981	37.8

Table 2: Reduction example, $B_c = 12$

N	M	C_{TRI}	C_{RED}	R %
10	5	422	208	49.2
10	10	845	411	48.6
20	10	1695	708	41.7
20	20	3390	1283	37.8
40	10	3395	1274	37.5

Table 3: Reduction example, $B_c = 16$

4. REPRESENTATIVE EXAMPLES

Using the approach discussed significant power reduction and hardware saving can be achieved as it will be illustrated by some representative examples. For this example we assume that the weight distribution of a trained neural network is approximately Gaussian. (see [5]). This is why the elements of the sample coefficient matrices were generated randomly in $[-(2^{Bc-1}), 2^{Bc-1}]$ with zero mean and ($\sigma_2 = \frac{2^{Bc-1}}{3}$). (Approximate Gaussian distribution.)

The following tables (Table 1, 2, 3) contains some representative results for coefficient word length $B_c = 8, 12, 16$, respectively. The number of inputs N are $10, 20, 40$ while the number of outputs M are $5, 10, 20$. The following notations are also used:

- the hardware cost of the trivial realization can be approximated by $C_{TRI} = \frac{M*N*(B_c+1)}{2}$ (This is because the average number of non-zero elements is $\frac{B_c}{2}$ in the binary representation.)
- the "C_{RED}" column contains the hardware requirement of the reduced matrix-vector multiplier
- the "R %" column shows hardware requirements in percent compared to trivial representation ($\frac{C_{RED}}{C_{TRI}}$)

Concluding these examples it can be said that the reduction ratio increases as the dimensions (N and M) and the word length (B_c) increase. This can be easily explained because the more bits in the matrix-vector product the more to reduce. However, it is not definitely true because at different word lengths the algorithm can execute different reduction steps, but it can be considered as a trend.

To summarize the results, this modified reduction method results in 8-10% saving compared to the algorithm presented in [2].

The number of arithmetic unit required by the multipliers in the given examples (C_{RED} in table 1, 2, 3) can be compared to data in table 4 which contains some details about Xilinx FPGAs. Experiences show that the routing of this distributed implementation uses mainly simple and local connections, thus the routing will not be a bottleneck. Moreover, the control of the bit-serial arithmetic is simple, too. These allow chip utilization of higher than 90%.

For instance a 20-input 10-output fixed NN with 16-bit coefficients can be implemented on a Xilinx XC4020 FPGA. Considering the timing data of the XC4000E family this system will provide its outputs in every $\frac{(B_c+B_x)}{50MHz} \simeq 0.5\mu secs$ (assuming that the word lengths of the input data are $B_x = 8$ and the clock rate is 50MHz).

Type	CLBs	Arithm. units
XC4020	784	784
XC4028	1024	1024
XC4044	1600	1600
XC4062	2034	2034
XC4085	3136	3136
XC40125	4608	4608
XC6216	4096	1365
XC6264	16384	5460

Table 4: Size of Xilinx FPGAs in CLB and the number of implementable arithmetic units

5. CONCLUSION

The proposed new synthesis method can significantly reduce the hardware requirement of the bit-serial matrix-vector multipliers. This distributed arithmetic approach provides high operating speed by the full parallel computation of the whole layer. The optimization algorithm can be integrated into a neural network design environment in which the desired NN can be trained by a simulator. After training the reduction algorithm can run through the resulting coefficient matrices from which output the net-list of the matrix-vector multiplier circuit can be generated.

6. REFERENCES

[1] B. Fehér, ”Efficient Synthesis of Distributed Vector Multipliers”, Microprocessing and Microprogramming, Vol. 38, pp. 345-350, 1993

[2] T. Szabó, B. Fehér, G. Horváth, ”Neural Network Implementation Using Distributed Arithmetic”, accepted for KES’98

[3] R.I. Hartley, K.K. Parhi, ”Digit-serial computation”, Kluwer Academic Publishers, 1995

[4] S.A. White, ”Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review”, IEEE ASSP Magazine, pp. 4-19, July 1989

[5] I. Bellido, E. Fiesler, ”Do Backpropagation Trained Neural Networks have Normal Weight Distributions?”, ICANN’93

[6] M. Glesner, W. Pöchmüller, ”Neurocomputers, An overview of neural networks in VLSI”, Chapman & Hall, 1994