A HARDWARE EMULATOR FOR BINARY NEURAL NETWORKS

Marcin Skubiszewski

Digital Equipment Corporation Paris Research Laboratory (DEC PRL)
85, av. Victor Hugo, 92563 Rueil Malmaison Cedex
skubi@prl.dec.com

ABSTRACT

A neural network emulator is described in this paper. The device can implement neural networks of various architectures, provided that the inputs and outputs of neurons are binary.

The device is fully digital. It is mainly composed of 128 serial adders, and can be viewed as a 128-processor specialized SIMD machine. It can emulate neural layers of any size, only depending on the amount of RAM attached (a fully connected Boltzmann machine with 351 neurons has been implemented).

As an example of application, graph bisection using a Boltzmann machine is discussed on the basis of results obtained using the prototype.

1. Basic definitions; the task of the emulator

1.1 A neuron

A neuron is a small computing device with many inputs (called $e_1$) and one output (called $s$). It performs the operation

$$s = f \left( \sum e_1w_1 \right)$$

where $f$ is the threshold function and $w_1$'s are the neuron's weights (real numbers approximated in some arithmetic). The form of the threshold function is fixed for any given type of neural network, but the weights as well as the parameters of the threshold function are different for each neuron.

$$f(x) = \begin{cases} 1 & \text{if } x \geq T \\ 0 & \text{if } x < T \end{cases}$$

Binary threshold implemented

$$f(x) = \frac{1}{1+\exp(-x)}$$

Smooth threshold not implemented

Fig. 1 Usual threshold functions

1.2 Neural layers

A neural layer (fig. 2) is a group of neurons sharing a common set of inputs. A layer is the biggest component of a neural network whose structure does not depend on the network's architecture. So, to be general, the device implements neural layers rather than whole networks. These layers are viewed as atomic instructions by the software programs implementing particular neural networks.

1.3 The formulae of a neural layer

The following notations and formulae result from the remarks above. They describe the neural layer to be implemented by the emulator.

\[ n \quad : \text{number of inputs} \]
\[ m \quad : \text{number of outputs (each output corresponds to one neuron)} \]
\[ e_1 \quad : \text{inputs (}0 < i < n - 1\text{)} \]
\[ s_j \quad : \text{outputs (}0 < j < m - 1\text{)} \]
\[ N_j \quad : \text{neuron computing } s_j \]
\[ T_j \quad : \text{threshold of } N_j \]
\[ w_{1j} \quad : \text{the weight attached to } e_j \text{ in } N_j. \]

The neuron $N_j$ computes $s_j$:

$$s_j = \begin{cases} 1 & \text{if } \sum w_{1j}e_1 \geq T_j \\ 0 & \text{otherwise} \end{cases}$$

Fig. 2 A neural layer
2. The basic processor

The **basic processor** (fig. 3) is intended to implement one neuron at a time. It computes in fixed point two's complement arithmetic, which is also used to represent the weights; let's call b the precision (number of bits) of the arithmetic.

The weights are sent to the processor one by one and bit serially through the wire **weights**. The neuron's binary inputs are sent through the wire **inputs** one by one and in such a manner that during the transmission of the weight $w_{1,j}$, the corresponding input $e_1$ is present on the wire. Consequently, the AND gate delivers, also one by one and bit serially, all the products $e_1w_{1,j}$. These products are added altogether by the serial adder composed of the 1-bit full adder labelled $I$, the **partial sum register** and the carry register.

To compute the neuron's threshold function, we add to the **partial sum register** the fictive input-weight product $e^*w^*_j$ with $e^* = 1$ and $w^*_j = -T_j$;

then, the sign bit of the result (in the **partial sum register**) is the negation of $s_j$ as defined by (2). This bit is sent outside as the processor's output.

3. The row of basic processors

32 processors are assembled to form a **row** (fig. 4). The row can implement a neural layer of up to 32 neurons or part of a larger layer.

During a given computation, each processor implements one neuron. The only individual properties of a neuron $N_j$ are its weights $(w_{1,j})_1$ and threshold $T_j = -w^*_j$. These data are delivered to the corresponding processor by its wire **weights**. This wire is fed by one of the 32 data bits of the **weight RAM**. At the same time, the inputs (common to the whole layer) are delivered to all the processors by the **data RAM** through the wire **inputs**. At the end of the computation, each processor sends to the data RAM, through its **output** wire, the output $s_j$ of the neuron it implements. Thus, the row of processors can implement in one parallel computation a neural layer with any number of inputs and with 32 outputs.

Since a neural layer is nothing more than a group of neurons sharing a common set of inputs, we can split the neurons in a layer into several subsets and consider each subset as being a layer by itself. To implement a layer with more than 32 outputs, it is sufficient to follow this procedure to split it into several smaller layers; then, the outputs of the smaller layers are computed, one layer at a time, by the row of emulators, to be concatenated together.

As can be deduced from these remarks, a neural layer is implemented in $n*b*[m/32]$ cycles.

4. A global view of the emulator

The neural network emulator is implemented on PRL's prototype **programmable active memory** board Perle-0 (see [1]) (fig. 5). The board contains a matrix of 25 programmable gate array chips (Xilinx LCA 3020) and two blocks of fast memory. The programmable gate arrays contain programmable logic functions, registers and routing resources. They can be configured from the host computer to turn Perle-0 into various special purpose digital devices, in this case our neural network emulator.

The number of processors in a row is 32 because the block of RAM used to store the weights has a 32 bit wide data bus, which cannot feed a larger row at full speed. Since available logic resources let us lay out 128 proces-