THE LNEURO-CHIP: A DIGITAL VLSI WITH ON-CHIP LEARNING MECHANISM

J.B. Theeten, M. Duranton, N. Mauduit and J.A. Sirat
Laboratoires d’Electronique Philips
B.P. 15 - 3, avenue Descartes
94451 Liméil-Brevannes Cedex
France

ABSTRACT

Neural network simulations are often limited because of the time required for both the learning and the evaluation phase of the simulation. Our parallel digital LNeuro circuit drastically reduces these times by updating synaptic coefficients related to one neuron in parallel. Contributions of ‘input’ neurons to one output neuron are also computed in parallel.

LNeuro-chips can easily be associated using Transputer microprocessors as controllers. Boards communicate through the reconfigurable links provided by a SuperNode architecture. This allows to simulate large size-networks or structured network architectures like Multi-Layer Perceptrons.

We report demonstrations of a parallel system built with several LNeuro-chips, which include a local learning rule (on a ‘real-time’ application), and the famous Backpropagation algorithm.

INTRODUCTION

LNeuro is in 1.5 μm CMOS digital technology for easy interfacing, known precision, well-defined states and full control of parameters. It is designed to be highly efficient in matrix operations as well as in implementing most learning rules. Special attention was paid to make the device as flexible and cascadable as possible, so that networks of virtually any structure and any size can be built with associations of chips working in parallel.

The description of the architecture of LNeuro and references to other hardware implementations are given in [1] [2] [3] [4].

The core of the chip is the synaptic RAM (fig. 1). It contains 1024 ‘synapses’ (for 16 input and 64 output neurons). LNeuro implement a parallelism of 16 allowing to simulate layers of 32 input and 32 output 8 bit neurons with 16 bit synaptic coefficients (with asynchronous update of neurons). It could also implement layers of 256 binary input neurons and 4 output neurons. The VLSI performs two generic vectorial operations needed in neural computation:

- The scalar product $\sum_{j=1}^{N} W_{ij} V_j$,
- The Hebbian Learning Step which modifies all the synaptic weights $W_{ij}$ converging to a given output neuron i in a single time step according to: $W_{ij}(t + 1) = W_{ij}(t) + \Delta_i V_j$, where $\Delta_i$ is an integer (scalar) ‘increment’ that depends only on the output neuron i. $V_j$ is the state of the input neuron j which may be replaced by some other quantity related to input neuron j in special cases.
Various learning rules are implemented only by modifying the microcode controlling the general learning unit, and by doing several elementary steps. Detailed programming can be found in [2] [3] [4].

**SYSTEM ARCHITECTURE**

Parallel oriented processors (INMOS Transputers, for instance) are well suited to design systems making use of these circuits, as they are fast enough to perform part of the communication protocol required when using several neuromimetic chips.

The frame of this machine is Supernode, a general purpose parallel computer based on INMOS Transputers. The architecture of Supernode is modular, and so well suited to our prototype: it can gather up to 1024 processors and their associated memory. Its parallel architecture allows fast communication between the processors, through four asynchronous links. The communication path between all processors can be dynamically reconfigured with a crossbar interconnection network. Four Transputers take place on every motherboard; each of these controls four LNeuros.

The different blocks of a neurochip can be addressed by a control Transputer as a part of its memory.

**DEMONSTRATION WITH A LOCAL LEARNING RULE**

In many learning rules, modification of \( W_{ij} \) involves only the input neuron state \( V_j \) and the output neuron state \( V_i \). \( V_j \) and \( V_i \) might be replaced by other quantities related to neurons \( j \) and \( i \) respectively, e.g. desired states. We call them local as the procedure needs no information from neurons that are not connected by the considered \( W_{ij} \). The implementation of these rules is rather straightforward in LNeuro: loading the microcode of the rule, determining the right variables \( \Delta_i \) and \( V_j \), and executing several elementary steps.

In particular, neural algorithms for PCA (Principal Component Analysis) that are proved to be more efficient than standard covariance matrix diagonalization. This is due to their local aspect which simplifies computation (see e.g. [6] and references therein). We use such a scheme for an application which require real-time learning (TV-image compression, see [5] [7]).

**DEMONSTRATION WITH THE BACK-PROPAGATION ALGORITHM**

In contrast to the previous learning rule, it is non-local. In fact modification of synaptic weight \( W_{ij} \) requires information located in other units than \( i \) or \( j \). Thus, hardware implementation of this approach is not straightforward.

We already have proposed a special configuration which yields maximum computing speed and minimum inter-chip communication [2] [3] [4]. Fig. 2 displays the arrangement of circuits for implementing the error back-propagation algorithm on a 2-layer Perceptron. Two subsystems perform forward propagation of neural states (3 chips on the left), and back-propagation of errors (1 chip on the right), respectively. It also illustrates cascadability ability of the circuit.

As for the first demonstration, we apply a neural scheme to image compression. The architecture of the net is: \( N \) input neurons, \( n \) hidden neurons, \( N \) output neurons \( (n < N) \). When training e.g. on blocks of pixels extracted from a TV image, the net is able to restitute the whole image with good quality. In the 'standard' case it is related to principal component analysis, but some variants allow to improve the performances (see [5] [7]).