Multilayer Perceptrons on Splash 2

Nalini K. Ratha
Exploratory Computer Vision Group
Thomas J. Watson Research Center
Yorktown Heights, NY 10598

Anil K. Jain
Department of Computer Science
Michigan State University
East Lansing, MI 48824

Abstract

Multilayer perceptrons (MLPs) are one of the most popular neural network models for solving pattern classification and image classification problems. Because of their ability to learn complex decision boundaries, MLPs are used in many practical computer vision applications involving classification (or supervised segmentation). Once the connection weights in a MLP have been learnt, the network can be used repeatedly for classification of new input patterns. Several special-purpose architectures have been described in the literature for neural networks as they are slow on a conventional uniprocessor. In this paper, we describe mapping of MLPs onto Splash 2 – a "custom computing machine". The main features of the proposed mapping are: (i) the number of nodes in a layer is not fixed; (ii) the number of layers in the network is not fixed; (iii) it is based on a set of reprogrammable FPGAs and a programmable crossbar; and (iv) it has a significant speedup over a uniprocessor. The mapping has been used for implementing a 3-layer MLP for page segmentation application with an appreciable speedup of approximately 150 over a SPARCstation 20 for one million pattern vectors with 20 features per pattern.

1 Introduction

Artificial neural networks (ANNs) attempt to mimic biological neural networks. One of the main features of biological neural networks is the massively parallel interconnections among the neurons. Computational model of a biological neuron involves simple operations such as inner product computation and thresholding. A taxonomy of ANNs is given in [5]. Amongst these different kinds of networks, the simplest model is that of a perceptron shown in Figure 1 (a). Given a d-dimensional input feature vector $X = (x_1, x_2, \ldots, x_d)$, the output y is used to determine the category of the input. A perceptron can be trained to learn a linear decision boundary in a multidimensional feature space. Computationally, a perceptron performs innerproduct of its input vector X and the weight vector W. The output of the innerproduct stage is subjected to a non-linearity, typically a $tanh(x)$ type of function, producing the desired output. A multilayer perceptron (MLP) consists of several layers of perceptrons as shown in Figure 1(b). The nodes in the i^{th} layer are connected to all the nodes in the $(i+1)^{th}$ layer through suitable weights. There is no interconnection among the nodes of a layer. By convention, the first layer is the input layer and the last layer is the output layer. The input pattern vector is presented to the network and the output is observed on the output layer nodes. Training of a MLP involves two stages: (i) feedforward stage: the training patterns with known class labels are presented at the input layer starting with a randomly initialized weight matrix and computing the output at the output node. (ii) weight update stage: the weights are updated in a backward fashion starting with the output layer. The weights are changed proportional to the error between the desired output and actual output. Typically, the backpropagation algorithm is used for this purpose. The two steps are repeated until the network converges. Once the weights have been learnt, a MLP is supposed to have generalization capability, i.e., it can near correctly classify unknown patterns. For this reason, MLPs are commonly used in many pattern recognition applications. At most three layers are needed to implement an arbitrary complex decision boundary [5].

MLPs can be implemented using digital and analog techniques. Computationally, a node in a MLP performs innerproduct of the input vector with the weight vector at the node and applies a non-linear function to the output. In this sense, the computational power needed at a node is not very demanding. However, the number of interconnections is very large. For an n–node MLP, $O(n^2)$ interconnections are needed which makes mapping a MLP onto a parallel processor a real challenge. On a uniprocessor, the whole operation proceeds sequentially one node at a time. Hence, there is no complex communication involved. However, for a high performance implementation, efficient communication capability must be supported.

In a typical pattern recognition and computer vision application, the number of input nodes (equivalently, the number of features) is usually large ($>$ 100 is not uncommon). The classification process involving complex decision boundaries demands a large number of hidden nodes. Yet another constraint is the desired speed of the computer vision system. Often, the entire vision system, especially in industrial applications, needs to operate in "real-time" (frame rates). Hence, high input/output bandwidth is desired along with fast classification (recall) speeds. The network training can be carried out off-line. Therefore, we focus on the recall phase. For a practical vision system,

0-8186-7987-5/97 $10.00

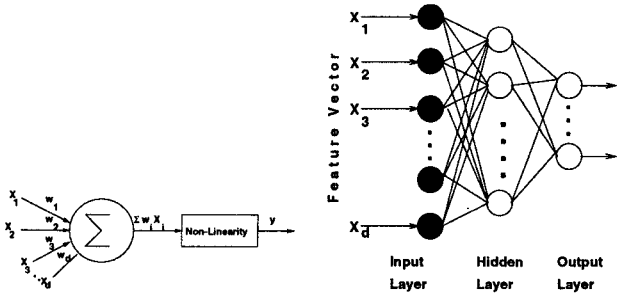

Figure 1: Perceptron. (a) a schematic of a perceptron; (b) a multilayer perceptron.

the total number of input patterns can also be very high as the number of input patterns are linked to the large number of pixels in the input image. For example, to process a $1,024 \times 1,024$ image in "real-time" requires that 30 frames be processed per second resulting in the number of input patterns equal to 30 million per second. A real-time neural network classifier is then expected to perform billions of operations per second. Note that the connection weights in general are floating point numbers demanding floating point multiplications and additions. On a general-purpose uniprocessor, throughputs of this kind are difficult to achieve even with today's most powerful processors. Hence, several parallel and special-purpose architectures have been designed and fabricated to implement artificial neural networks [11]. It has been observed that even general-purpose parallel processors fail to provide high performance for ANNs. Hence, many special neurocomputers have been built using either commercially available special-purpose VLSIs (DSPs, floating-point processors) or special-purpose VLSIs, possibly, using analog, digital or hybrid design techniques. A special-purpose VLSI architecture provides the best performance. But, with a dynamically changing architecture (the number of nodes and layers) from application to application, it is expensive to design a VLSI architecture for individual applications. Typically, architectures with a fixed number of nodes and fixed number of layers are fabricated.

Many special-purpose implementations of neural networks have been described in the literature. A survey of parallel architectures for neural networks is given in [11]. A multi-layer perceptron implementation has been described in [7] using GAPP - a systolic array processor chip. The architecture of SNAP-64, a 1-dimensional ring of parallel floating point processors, is described in [6]. Using Xilinx XC 3090 FPGAs, Cox *et al.* [3] describe the implementation of GANGLION. A single board caters to a fixed neural architecture of 12 input nodes, 14 hidden nodes and 4 output nodes. Using the CLBs, 8×8 multipliers have been built. A lookup table is used for the activation function. Bortos *et al.* [1] describe a smaller network implementation using less powerful Xilinx XC 3042 FPGAs. Their system supports 5 input nodes, 4 hidden nodes and 2 output nodes. A neuron is based on two XC 3042s and the nonlinearity is based on an 8K EPROM-based lookup table. Several other implementations have been surveyed in [7]. There have been many other well-known neural network chips and architectures e.g., ANNA [10], CNAPS [7].

In this paper we take a different approach to mapping MLPs onto a custom computing machine. There are several advantages of a custom computing approach over a VLSI implementation. In Section 2, we describe some of the specific advantages. Section 3 contains a brief description of architecture and programming environment of Splash 2 — one of the earliest and most successful custom computing machines. In Section 4, we describe our mapping of MLP onto Splash 2. In section 5, the performance of the mapping is evaluated with respect to an image segmentation algorithm. Section 6 provides conclusions and future work.

2 Custom Computing Machines

In contrast to a general-purpose processor, the application specific integrated circuits (ASICs) are used for a specific application. For a given application, ASICs provide higher performance compared to a general-purpose uniprocessor. If a user can customize the architecture and instructions needed for a given application, i.e., program at a gate-level, then a high performance can be achieved. This is the basis of a *custom computing machine (CCM).* Using a CCM, a designer can tune and match the architectural requirements of the problem. There are other advantages of using a CCM. An ASIC is fast but is costly, non-reconfigurable and needs more time to implement. A CCM can overcome these limitations easily. CCMs use field programmable gate arrays (FPGAs) as compute elements. Since FPGAs are off-the-shelf components, they are relatively cheap. By virtue of the reconfigurability of the FPGAs, they are easily reprogrammed. As CCMs do not need to be fabricated with every new application, they are often employed for fast prototyping and save a considerable amount of time in design and implementation of algorithms.

3 Splash 2 — Architecture and Programming Flow

Splash 2 is one of the leading FPGA-based custom computing machine designed and developed by the Supercomputing Research Center [2]. The Splash 2 system consists of an array of Xilinx 4010 FPGAs, improving on the design of the Splash 1 which was based on Xilinx 3090s [2]. Each Splash 2 processing board has 16 Xilinx 4010s as PEs ($X_1 - X_{16}$) in addition to a seventeenth Xilinx 4010 (X_0) which controls the data flow into the processor board. Each PE has 512 KB of memory. The PEs are connected through a crossbar that is programmed by X_0. There is a 36-bit linear data path (SIMD Bus) running through all the PEs. The PEs can read data from their respective memory. A broadcast path also exists by suitably programming X_0. The Splash 2 system supports several models of computation, including PEs executing single instruction on multiple data (SIMD mode) and PEs executing multiple instructions on multiple data

(MIMD mode). It can also execute the same or different instructions on single data by receiving data through the global broadcast bus. Individual memory available with each PE makes it convenient to store temporary results and tables. The crossbar cane be used to set arbitrary communication paths between PEs. To program a Splash 2, we need to program each of the PEs (X_1- X_{16}), the crossbar, and the host interface. More details and the schematic of Splash system and a PE is shown in [9].

4 Mapping a MLP onto Splash 2

In implementing a neural network classifier on Splash 2, a perceptron implementation has been used as a building block. Hence, the design of a perceptron on Splash 2 is described first. A perceptron consists of two stages, namely (i) an inner product computation, and (ii) a non-linear function applied to the output of the previous stage as shown in Figure 1. In our case, the perceptron is assumed to have 20 inputs which uses a non-linear function (a sigmoid function) to produce a real-valued output. We have used $tanh(\beta x)$ with $\beta = 0.25$ as the non-linearity in our implementation. For our mapping, two physical PEs serve as a neuron. The first PE handles the inner product phase and the second PE handles the non-linearity stage and writes result to the external memory operations. As the connection weights are fixed (we assume that the perceptron has been trained), an efficient way of handling the multiplication is to employ a lookup table. Since a large external memory is available at every PE, the lookup table can be stored. A pattern vector component is presented at every clock cycle. The PE looks up the multiplication table to obtain the weighted product and the sum is computed using an accumulator. Thus, after all the components of a pattern vector have been examined, we have computed the inner product. The non-linearity is again stored as a lookup table in the second PE. On receiving the inner product result from the first PE, the second PE uses the result as the address to the non-linearity lookup table and produces the output. Thus, the output of a neuron is obtained. The output is written back to the external memory of the second PE, starting from a prespecified location. After sending all the pattern vectors, the host can read back the memory contents. A layer in the neural net is simply a collection of neurons working synchronously on the input. On Splash 2, this can be easily achieved by broadcasting the input to as many physical PEs as desired. The output of the neuron is written into a specified segment of external memory and read back by the host at the end.

For every layer in the MLP, this exercise is repeated until the output layer is reached. Note that for every layer, there is a different lookup table. Thus, we have been able to implement a MLP on Splash 2 utilizing the available hardware resources including the crossbar for broadcast purposes. A wavefront of computation proceeds one layer at a time. The schematic of the mapping for a single layer is shown in Figure 2.

We have implemented a lookup table-based multiplication scheme utilizing the external memory since

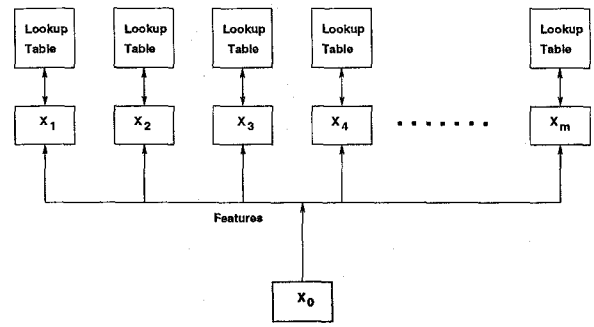

Figure 2: Mapping a single layer of a MLP onto Splash 2. The i^{th} PE computes $\sum w_{ij}x_i$ and $(i+1)^{th}$ PE computes $f()$ for odd i.

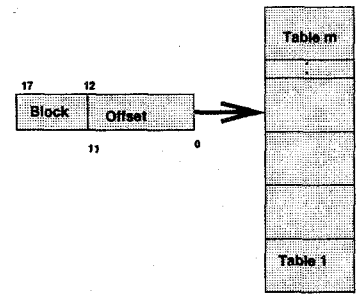

Figure 3: Lookup table address computation.

a full fledged floating-point multiplier is not possible on a Xilinx 4010-based PE. In our implementation of a neuron, the first PE in the PE pair is devoted to the inner product computations. There are m multiplications to be performed per node corresponding to the m-dimensional weight vector. In general, the weights are real numbers. The lookup table is divided into m segments. A counter is incremented at every clock which forms the higher order (block) address for the lookup table. The pattern vector component forms the lower order address bits. Splash 2 has an eighteen bit-wide address bus for the external memory. We have allocated high-order 6-bits for the block address and low order 12-bits are allocated for the offset address within a block (see Figure 3). Note that the offset can also be negative corresponding to a negative input to the lookup table. The numbers have been represented by a 12-bit 2's complement representation. Hence, the resolution of this representation is effectively eleven bits. Within the PE, the accumulator is 16-bit wide. After accumulation, the accumulator result is scaled down to 12-bits.

The non-linearity used in a neuron is the hyperbolic function $tanh(0.25x)$. For efficiency considerations, a lookup table-based approach has been taken to compute the value of this function on Splash 2. The address is 12-bits long as obtained from the accumulator. Note that the input can be either a positive or a negative number. Hence the non-linearity lookup table has a 11-bit resolution. Though the memory is

16-bit wide, effective 12-bit 2's complement numbers only are stored.

5 Performance Evaluation

Let m denote the number of features (no. of input layer nodes), K be the number of patterns to be classified and l be the number of layers in the network. In our implementation, $m = 20$, $l = 2$ and K is the total number of pixels in the input image (e.g., 1,024 $\times$ 1,024 = 1 M pixels). We now analyze the requirements for mapping a MLP onto Splash 2 in terms of number of PEs required and the number of clock cycles required to complete a classification process. The number of PEs needed is equal to twice the number of nodes in each layer. Number of clock cycles needed $= m * K * l$. For the given values of m, K and l, the no. of clock cycles $= 20 * 2 * 10^6 = 40$ million. With a clock rate of 22 MHz, time taken for 40 million clock ticks = 1.81 secs.

In situations where the number of PEs required is larger than the available PEs, either more processor boards need to be added to the Splash 2 system or the PEs need to be time shared. Note that the neuron outputs are produced independent of other neurons and the algorithm waits till all the computations in each layer are completed.

A MLP has a communication complexity of $O(n^2)$, where n is number of nodes. As n grows, it will be difficult to get good timing performance from a single-processor system. With a large number of processor boards, the single input data bus of 36-bits can cater to multiple input patterns. Note that in a multi-board system, all the boards receive the same input. This parallelism can give rise to more data streaming into the system, thus reducing the number of clock cycles by a linear factor. For a 12-bit input, the scale down factor is 3.

The performance of the mapping on Splash 2 can be compared with a host implementation for different sizes of the neural network. For this comparison, we look at only a single layer and represent the network size by the number of nodes in that layer. Multi-layered networks are considered to be linearly scalable in our architecture. The performance measure is the processing time as measured by the number of clock cycles for Splash 2 with a 22 Mhz clock and the elapsed time on the workstation measure using 'clock' function. The sequential time and the Splash time have been plotted in Figure 4. Note that the time has been plotted on a log scale to accommodate the large scale difference in time.

For our network with 20 input nodes, implemented on a 2-board system, we achieve 176 million connections per second (MCPS) per layer by running the Splash clock at 22 MHz. In general, for a b-processor board system, a total speed of $176b$ MCPS is achievable. Thus, a 6-board system can deliver more than a billion connections per second. This is comparable to the performance of many high-end VLSI-based systems such as Synapse and CNAPS which perform in the range of 5 GCPS [11].

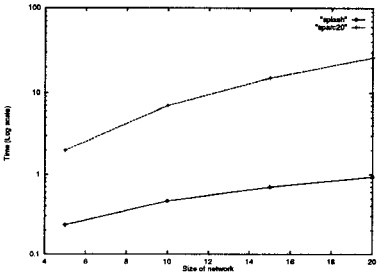

Figure 4: Speed comparison of neural network on Splash and Sparc 20.

6 Page Layout Segmentation using MLP

We now describe the implementation of image segmentation algorithm based on a MLP. The process of spatial partitioning of an image into mutually exclusive connected image regions is known as image segmentation. In an automated document image understanding system, page layout segmentation plays an important role for segmenting text, graphics and background areas. Such a segmentation allows us to apply character recognition algorithms to only text regions. Jain and Karu [4] have proposed an algorithm to learn texture discrimination masks needed for image segmentation. The performance of this approach for page layout segmentation has been demonstrated by Zhong et al.[12]. The schematic diagram of their segmentation algorithm is shown in Figure 5(a) (M = 7 in our implementation).

The page segmentation algorithm by Zhong et al.[12] has three stages of computation, namely, (i) feature extraction, (ii) classification, and, (iii) post-processing. The feature extraction stage is based on a set of twenty masks obtained by the learning paradigm proposed in [4]. The second stage is a multistage feedforward neural network with 20 input nodes, 20 hidden nodes and three output nodes. The connection weights and other parameters of the neural network have been learned for document images using the training approach described in [4].

The input to the algorithm is the gray level scanned image of the document and the output is the labeled image, where each pixel is assigned one of the three classes. A sample input image and the segmentation result produced by this algorithm are shown in Figure 5. The input gray level image is shown in Figure 5(a). The three-level segmentation results obtained by the sequential algorithm is shown in Figure 5(b) where the background is shown by black pixels, the text areas are shown in gray pixels and the graphics areas are shown in white. Figure 5(c) shows the results after the postprocessing stage which places bounding boxes around every segmented area. The text areas are enclosed by black boxes and graphics areas are enclosed by white boxes.

The mapping of filtering stage is discussed in [8]. We focus on the neural network-based classification stage. The classifier has 20 input nodes, 20 hidden

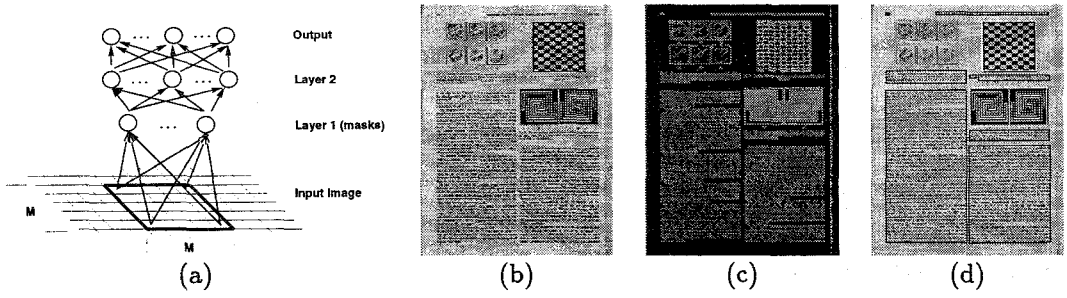

Figure 5: Page layout segmentation. (a) schematic of the algorithm; (b) Input gray-level image; (c) Result of the segmentation algorithm; (d) Result after postprocessing.

nodes and 3 output nodes. Using the mapping scheme described in Section 4, the classifier has been mapped onto Splash 2. The classification stage takes approximately 295 seconds on a SPARCstation 20 and using Splash 2 at 22 MHz, it is estimated to take 1.81 seconds. The segmentation output is the same whether it runs on Splash 2 or a workstation host.

7 Conclusions

In this paper a novel scheme of mapping MLPs on a custom computing machine has been presented. The scheme is scalable in terms of the number of nodes and the number of layers in the MLP and provides near-ASIC level speed. The reconfigurability of CCMs has been exploited to map several layers of a MLP onto the same hardware. An important attribute of the CCMs allows us to combine various sub-stages of an algorithm on the same hardware system by just changing the control bit stream. This property is useful in designing real-time complex vision systems. The performance gains achieved using this mapping have been demonstrated on a network-based image segmentation algorithm.

References

[1] N. M. Bortos and M. Abdul-Aziz. Hardware implementation of an artificial neural network. In *Proc. IEEE Intl. Joint Conference on Neural Networks*, pages 1252–1257, Nagoya, Japan, October 1993.

[2] D. A. Buell, J. M. Arnold, and W. J. Kleinfelder, editors. *Splash 2: FPGAs for Custom Computing Machines.* IEEE Computer Society Press, Los Alamitos, 1996.

[3] C. E. Cox and E. Blanz. GANGLION–a fast field-programmable gate array implementation of a connectionist classifier. *IEEE Journal of Solid-State Circuits*, 27(3):288–299, March 1992.

[4] A. K. Jain and K. Karu. Learning texture discrimination masks. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(2):195–205, February 1996.

[5] A. K. Jain, J. Mao, and K. M. Mohiuddin. Artificial neural networks: A tutorial. *IEEE Computer*, 29(3):56–63, March 1996.

[6] R. W. Means. High speed parallel hardware performance issues for neural network applications. In *Proc. IEEE Intl. Joint Conference on Neural Networks*, pages 10–16, Orlando, Florida, June 1994.

[7] T. Nordstrom and B. Svensson. Using and designing massively parallel computers for artificial neural networks. *Journal of Parallel and Distributed Computing*, 14:260–285, 1992.

[8] N. K. Ratha and A. K. Jain. High performance custom computing for image segmentation. In *High Performance Computing Conference, New Delhi*, pages 67–72, December, 1995.

[9] N. K. Ratha and A. K. Jain. FPGA-based computing in computer vision. In *Proc. of IEEE Workshop on Computer Architecture for Machine Perception, Boston, Massachusetts*, October, 1997. Accepted for publication.

[10] E. Sackinger and H.-P. Graf. A board system for high-speed image analysis and neural networks. *IEEE Trans. on Neural Networks*, 7(1):214–221, January 1996.

[11] N. B. Serbedzija. Simulating artificial neural networks on parallel architecture. *IEEE Computer*, 29(3):56–63, March 1996.

[12] Y. Zhong, K. Karu, and A. K. Jain. Locating text in complex color images. *Pattern Recognition*, 28(10):1523–1536, October 1995.