A Neuro-Emulator with Embedded Capabilities for Generalized Learning

Valentine C. Aikens II
Hewlett Packard
Enterprise Storage Division
Roseville, CA 95747-5681

José G. Delgado-Frias
Electrical Eng. Dept.
State Univ. of New York
Binghamton, NY 13902-6000

Gerald G. Pechanek
BOPS, Inc.
6340 Quadrangle, 210
Chapel Hill, NC 27514

Stamatis Vassiliadis
Electrical Eng. Dept.
Delft Univ. of Technology
Delft, The Netherlands

Key words: Neural Networks, Learning Algorithms, Computer Architecture, Parallel Systems, High Performance Computing.

Abstract

Artificial neural networks (ANN) are being used as one of the prime computing tool for an increasing number of applications. This is due in part to the ANN's ability to adapt to changes via learning. The dynamic nature of many applications as well as the computational and storage requirements of current learning algorithms creates a need for high performance neuro-architectures with learning capabilities. In this paper we identify a set of computational, communication and storage requirements for learning in ANNs. These requirements are representative of a wide variety of algorithms for different learning approaches. We propose a novel neuro-emulator that provides the computational ability for the stated requirements. While meeting all the identified requirements the new architecture maintains a high performance during learning. To show the capabilities of the proposed machine we present four diverse learning algorithms and step through the execution of each using the proposed architecture. We include an evaluation of the machine performance as well as a comparison with other architectures. It is shown that with a modest amount of hardware the proposed architecture yields an extremely high number of connections per second.

1 Introduction

The artificial neural network (ANN) paradigm has been used as a computing alternative for a number of applications to which other paradigms may not be well suited. These applications include speech processing [1], adaptive and self-learning control systems [2][3], sequence verification through inferencing [4], visual mapping and localization [5], classification of parts for cellular manufacturing [6], location of visual features [7], path optimization [8][9], and load forecasting resulting in economic power distribution [10].

In the ANN paradigm the input to a neuron is the weighted sum of an input vector. The input vector is comprised of the outputs of the neurons connected to the present neuron. The output of the neuron is the result of applying an activation function, usually non-linear, to the input value. Consequently, there must be a set of mechanisms that compute:

the product of the input vector and the weight matrix, the sum of the products and the activation functions. The outputs can then be used either as the solution or as the inputs for the next network iteration. In the learning process these outputs are used to compute an output error vector.

The purpose of a learning algorithm is to increase the correctness of the network's output as well as to provide network adaptiveness. This is accomplished to a large extent by changing the value of the weights in the network. The amount and method by which the weights are modified varies greatly from one algorithm to another. A number of learning algorithms have been developed to suit the applications and network characteristics [11][12][13][14]. These learning algorithms present different computation and storage requirements. We have identified a set of computation and storage requirements that are common to most learning algorithms. A list of these requirements follows.

- *Competitive learning capabilities.* Competitive approaches are used to determine the neuron with the largest (or smallest) output (commonly referred to as winner-takes-all: WTA). This will require the ability to compare the outputs and determine the winner.

- *Support for multi-layered networks.* Multi-layered networks are sparsely connected; i.e. every node is not connected to every other node. This presents demands on both recall and learning. Assuming the neurons are fully connected between layers, there must be a separate weight matrix for each layer to implement recall. During learning in multi-layered networks the error at the output layer can be calculated directly. The error for neurons in a hidden layer i is derived from the errors in layer $i+1$. This requires both calculation of the hidden layer neuron error and storage for intermediate results.

- *Support for gradient descent.* Gradient descent is the process of traversing a surface in the steepest direction. In neural networks the surface is usually an error surface, and the direction is such that the error is minimized. This type of learning requires the ability to produce the derivative of the error function at the neuron output value.

- *Weight modification and storage.* Learning algorithms usually increase the correctness of the output by changing the value of the weights. To embed learning in an architecture there must be a means of modifying the weights and storing the modified values.

- *Calculation of dot product.* Some algorithms are concerned with matches in the direction of the input and weight vectors rather than matches of magnitude. The distance between two vectors in N space can be determined by means of a dot product of the two vectors for these algorithms.

- *Method of normalizing weight vectors.* The result of the dot product of two vectors is a function of magnitude and the angle between them. The dot products of the weight and input vector of all nodes can be directly compared if the weights are normalized [15]. This requires the ability to normalize the weight vectors.

- *Generation of multiple functions.* Learning algorithms employ a number of functions in the process of calculating new weight values, as well as in determining the winning neuron. These functions include inverse, exponential, square root, threshold, sigmoid,

and the derivatives of some of the activation functions. The required functions must be available within acceptable error limits as well as time and cost constraints.

- *Storage for intermediate results.* Often learning algorithms are comprised of multiple stages or values. To increase performance each value or stage can be evaluated for all neurons before beginning the next logical step. This requires the intermediate results be stored until all calculations are completed, and the final result can be determined.

- *Calculation of neighborhood size.* Some algorithms have been shown to converge faster if the weights are updated as a neighborhood around the winner rather than updating the winning neuron alone. The calculations for the neighborhood size and determination of the nodes within the neighborhood are typically different from those required by the recall operation.

- *Calculation of linear distance between node locations.* There are algorithms which use the linear distance as the criterion for determining the winning neuron. The linear distance between nodes is also an intermediate step in some neighborhood calculations.

- *Vigilance testing.* Clustering algorithms require a criterion for creating a new class. Some algorithms test the degree of similarity of the winning neuron to the input vector for this purpose. This test often requires many calculations and evaluation of the result.

- *Ability to disable and manipulate individual nodes.* The vigilance test utilized by some algorithms is an iterative process. If the vigilance test failed another round of competition must occur without the previous winner(s) taking part. It can possibly become necessary to progressively disable nodes during each iteration.

- *Ability to store weights to symmetric locations in matrix.* Algorithms exist that result in symmetric weight matrices. The ability to store the unique weights to all their symmetric locations reduces the number of calculations required when implementing these algorithms.

As the complexity and problem size of new applications becomes large, there is an increasing need for architectural support for learning algorithms. Artificial neural networks are increasingly being used in applications with inherently dynamic environments. This in turn implies the network weight values need to be updated frequently. The number of calculations required by learning algorithms are usually higher than that required for the recall operation. As a consequence the total number of calculations and the amount of storage required greatly increase. Embedding the capabilities required by learning algorithms in the architecture may not require much additional hardware. Further it appears that learning algorithms, in particular those we have studied, contain parallelism and will exploit the proposed machine organization. If the learning algorithms are implemented in a host computer, the communication overhead will potentially become a bottleneck. This will be due to the high frequency of updating the weights and the massive amount of data required to implement the learning algorithms. Embedding learning capabilities in the neuro-emulator greatly impacts the neuro-emulator's performance. The communication with the host computer is

reduced to the data required at initialization of the network, returning the solution, and the swapping of input values - which can possibly be overlapped with other computations.

This paper has been organized as follows. First we present a description of the Sequential PIplined Neuro-emulator with Learning capabilities (SPIN-L) organization. We show how four learning algorithms can be mapped to the machine organization. These examples are used to illustrate the architecture's functionality; they represent a small sample of the algorithms the emulator is capable of implementing. We then present a performance study of four learning algorithms and a comparison of neuro-emulators that have been suggested in the literature [16]-[24]. Finally we draw a set of conclusions from the study presented here.

2 SPIN-L Description

A novel emulator, called SPIN-L, provides architectural support to fulfill the computational, communication, and storage requirements that are commonly found in a number of ANN learning algorithms. The proposed SPIN-L emulator has a similar structure to a neuro-emulator for the ANN recall process (SPIN) [25] which does not provide hardware support for learning. In the design of SPIN-L we have incorporated a number of novel features to support the learning requirements. In this section we provide a description of SPIN-L with particular emphasis on the novel features.

2.1 Machine Organization

The SPIN-L organization, shown in Figure 1, is comprised of three main sections: the parallel, the reduction, and the multi-function generator. This overall organization is similar to SPIN [25] where these sections are identified as processing elements (containing registers and multipliers), adder tree (a set of bit serial adders arrenged in a tree structure), and function generator (based on a lookup table).

In SPIN-L, the parallel processing cells (PPC) contain four units: a memory unit, an update unit, a register file and a multiplier unit. The reduction section is a binary tree structure (called the processing and communicating tree - PCT) whose processing cells are capable of performing a variety of operations including arithmetic, logic and bi-directional communication. The multi-function generator (M-FG) is capable of evaluating the sigmoid, inverse, inverse squared, exponential, cosine, sine and other non-linear functions. The multi-function generator section also contains storage for intermediate results (IS), as well as hardware to perform single valued operations. The PN-Bus connects both the root of the PCT and the output of the M-FG back to the last PPC. The parallel and reduction sections operate in a bit serial fashion while the multi-function generator is a word parallel implementation. The overall machine design contains N PPCs, the PCT with $\log_2 N$ stages, the multi-function generator and the feedback bus.

The machine is designed to exploit parallelism inherent in the target algorithm and to take advantage of pipelined execution. To calculate a sum of products, a common requirement in ANN algorithms, the PPCs compute N products in parallel. The products are generated bit serially and, the bit-streams as they exit the PPC multipliers, are passed to the PCT which performs the summation. The sum can then be passed to the input of the M-FG for

evaluation of a function or passed back to the last PPC using the PN-Bus. The individual hardware sections are independent of each other allowing them to operate on different sets of data simultaneously. As the first sum of products is being passed back to the last PPC, the second set of products are being summed, and the third are being calculated. This parallel pipelined operation results in a high performance computing engine, while the bit serial implementation provides flexibility of word size, low cost and potentially higher clock rates.

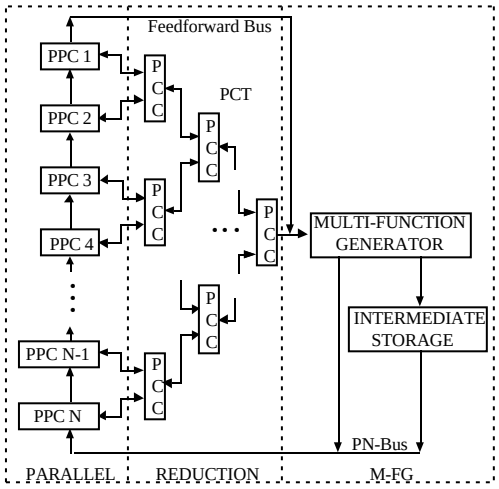


Figure 1: SPIN-L Organizational Overview

2.2 Detailed description of major components

The PPC organization, shown in Figure 2, consists of four units as described above. The memory unit (MEM) is comprised of standard memory and three shift registers. The shift registers provide the bit serial interface between the standard memory and the other hardware units. There is also a buffer register which allows for continued pipelined operation during data transfers to memory. The register file (RF) consists of four shift registers connected to one input and two outputs. The two outputs of the register file can be supplied by any combination of the four shift register values, including one register providing both outputs. The shift registers can receive data from any one of four possible sources, the memory unit, the update unit, the data bus or the previous PPC's register file. The multiplication unit (MLT) is a bit-serial 16-bit two's complement multiplier. It is preceded by a multiplexer which allow its inputs to be selected from a variety of sources. The multiplexer also allows any source to supply both inputs allowing for efficient calculation of squared terms. The update unit (UPD) contains the ability to add, subtract and accumulate. It is possible to bypass both the update and the multiplier cells with no delay.

The PCT is a binary tree with $\log_2 N$ stages. Each node of the tree is called a processing communicating cell and is capable of performing a variety of operations. Because the PCCs operate on bit-streams of data the hardware is compact in size allowing for greater functionality at a reasonable cost. The data paths shown in Figure 1 that connect the individual PCCs are comprised of two uni-directional busses. The PCCs are capable of performing arithmetic,

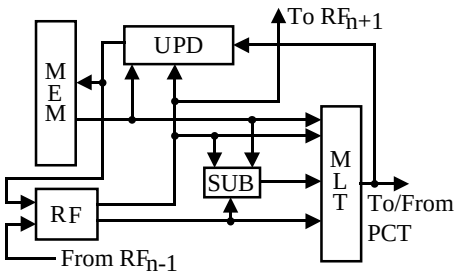


Figure 2: Detailed view of PPC

logic, and communication in the forward direction. They are also capable of communication in the backward direction while forward operations are being performed. Further the inputs can be simultaneously switched with each other and passed back to the previous stage. The logic operations include comparisons allowing the PCT to perform competitive learning with minimum delay. During this operation the path traversed through the tree by the winner is left open for communication of subsequent data.

The M-FG is based on a piecewise linear approximation to non-linear functions. The non-linear function is broken into sections, each section is represented by a linear function. It is possible to evaluate the non-linear function by storing the slope and Y-intercept of each linear function. This reduces the number of entries in the lookup table while providing an accurate representation of many functions. To evaluate a function the required calculations are lookup, a multiplication, and an addition which are pipelined.

The PPCs are able to exploit the inherent parallelism of learning algorithms. The reduction section is able to perform a summation of N values $O(\log_2 N)$ cycles. The Multi-Function Generator provides an accurate approximation to non-linear functions within the constraints of time and cost, allowing an increased number of functions to be supported. The sections of hardware, as well as the stages or units within the sections, can operate in a pipelined fashion. These traits along with the flexibility of the multi-function generator provide a strong core for the execution of learning algorithms on chip. The learning requirements stated in the introduction have been addressed in the design of SPIN-L. Below we present the major elements of the new architecture and the learning requirements which they fulfill.

- *The register files (RFs).* The register files have been enhanced to provide part of the support required by multi-layered networks as well as intermediate result storage. The RFs contain four registers for the storage of the input vector and intermediate results. These registers are connected to two multiplexers which provide the outputs of the RF. Any combination of registers can be provided to the outputs, including one register providing both outputs. One output is also connected to the next RF to implement inter-RF communications.

- *The connections between the weight store and the multipliers.* The outputs of the weight store and the RF are connected to multiplexers which supply the inputs to the multiplier. These multiplexers also control the routing of values to the inputs a subtraction unit prior to the multipliers. The bit serial design allows for a high degree of connectivity with a small number of actual data paths. The multiplier is able to

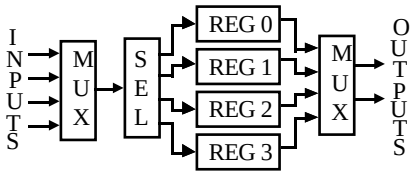


Figure 3: Register File.

receive its inputs from a variety of sources, including one source providing both inputs. This assists in the calculations required for learning in multi-layered networks and neighborhood size determination.

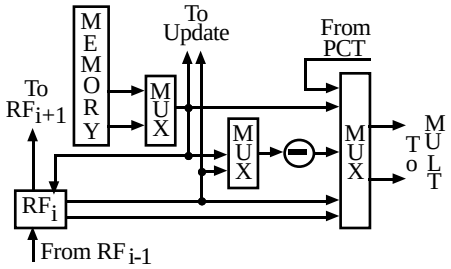


Figure 4: Data Paths to a Multiplier.

- *The Update Unit.* The adders in the Update Units facilitate the weight modification during the learning and normalizing processes. The source of the inputs for these adders can be the output of the multiplier, the RF, or the weight store. The sum is utilized for incremental modifications of the weight vectors. The carry is used to perform the weight updating for algorithms that require the logical AND function, this is a bitwise operation during which the carry is not fed back to the adder.

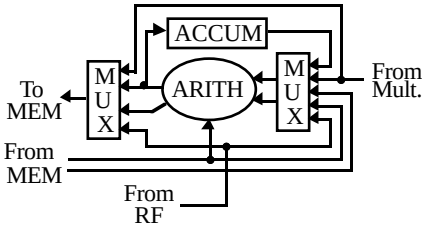


Figure 5: Update Hardware.

- *The processing communicating tree (PCT).* The PCT is a multi-faceted hardware design supporting many computational, communicative and miscellaneous requirements. It provides the necessary arithmetic operations, and also has the capability to perform logical operations required for competitive learning. The tree also provides communication from the sequential end to all the leaves. During competitive learning it maintains the path from the leaf of the winner to the root of the tree. This is used to disable individual nodes as well as for communication purposes.

- *The storage at the sequential end.* The added memory at the sequential end of the architecture will be used to store the learning rate, neighborhood width parameter, and data required for neighborhood calculations, as well as intermediate results. The intermediate results will be stored here to allow for improved performance through continued pipelined operation.

- *The multipliers in the M-FG section.* Many learning algorithms are composed of iterative operations, i.e. sum of products. In their final stages they also often require operations on singular values. These multipliers will be used in this manner during the neighborhood calculations and multi-layered network learning.

2.3 SPIN-L Recall Operation

Most of the learning algorithms today use the network output for recall to determine the amount of adjustment required. As a result the recall and learning operations will be interleaved with learning occurring if the recall operation's output is determined to be incorrect. It is therefore appropriate to examine the recall process before advancing to the learning operation. Figure 6 presents the architecture fully pipelined during recall. We present here an overview of a fully connected network with N neurons during a recall operation.

1. The initial weight vectors are pre-loaded into memory such that the elements of each vector are distributed across the PPC elements. The input vector is stored in the RFs, distributed in the same manner as the weight vectors.

2. SPIN-L begins recall with the multiplication of the input vector, which is stored in the RFs, and the weight matrix. This occurs one weight vector at a time using the multipliers. The input vector elements are multiplied by their corresponding weight vector elements.

3. As the products are produced they are passed to the PCT for the summation process.

4. Its output is used as the input to the function generator which implements the activation function.

5. This value is then passed back on the PN bus to the register file. As the outputs are produced and passed back on the PN-Bus, they are iteratively passed up to the next RF.

These calculations are repeated for all the weight vectors in the described pipelined fashion. As neurons are emulated the updated values are passed back to registers in the register file. The multiple registers in the RFs allow this to occur without overwriting the input vector elements. At the end of the update period, all N neurons have been emulated and the updated-inputs are in the RF registers available for the next update period.

The SPIN-L architecture allows the multiplication of the current weight vector elements to be overlapped with the summation of the products of the previous weight vector elements. Likewise the summation can occur simultaneously with the operation of the function generator. Communications occur on a separate bus allowing it to proceed simultaneously with the

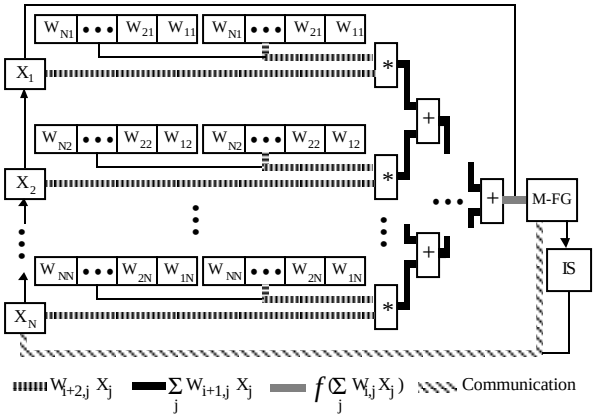


Figure 6: SPIN-L: Pipelined Recall Operation.

operation of the other sections. The outputs of the register file can supply any combination of the register values, including one register supplying both outputs. The bit-serial design allows the units of the PPCs to be highly interconnected using a small number of actual data paths. The update unit is capable of altering the weight vector elements by incrementing or logically ANDing. The PCT is capable of calculating a summation of N terms O(log2N). The storage and multipliers in the M-FG section provide for increased performance by allowing continued pipelined operation. It should be pointed out that SPIN-L and SPIN machines are fully compatible when they perform recall (or feedforward) neurocomputing operations.

In this section we have presented an overview of the proposed machine, and its functionality during recall. We also presented the components that will implement the additional computation and storage requirements imposed by learning algorithms. To show the performance of the proposed machine we now present an evaluation approach for neuro-emulators that focuses on the architectural design, i.e.. it is implementation and technology independent.

3 Performance Evaluation Approach

There have been a number of approaches proposed to measure the performance of a neuro-emulator. Most of them, however, have been technology-oriented; i.e. the assumptions and/or implementations are based on a given technology. We propose and alternative performance measurement based on an architectural perspective where the delays in the critical path are taken into account. The approach for the evaluation is introduced by means of an example.

3.1 Evaluation Method

In order to evaluate the neuro-emulator performance, we have identified the computational requirements of the neural network paradigm as well as hardware assumptions that are common to all the neuro-emulators. The governing equation for the emulation of a fully

connected artificial neural network with N neurons is:

$$Y_i(t+1) = \mathcal{F}\left(\sum_{j=1}^{N} W_{ij}\ Y_j(t)\right) \tag{1}$$

When evaluating the performance of a network of neurons governed by equations in the form of Equation 1, we have defined an update cycle as the time required to compute and communicate all the new neuron values.

Any evaluation of different architectures must be based on a set of realistic assumptions [26][27][28]. It is imperative that the evaluation technique itself does not give any organization an unfair advantage. The evaluation must be carried out at a level where not only the elements of the organizations have an impact on the performance results but also implementation issues do not influence the results. The assumptions that will be used to define this level of evaluation are:

- *Realization of equation 1.* All neuro-emulators use Equation 1 for the purpose of emulating a N-neuron fully connected network. This equation is the most general equation for neural network applications.
- *A non-linear activation function.* The activation function is considered to be a non-linear function [29][13].
- *L-bit precision.* All architectures will use L bits of precision for both the weight and neuron values.
- *L most significant bits required by activation function.* The 2L bits produced by the multiplication will be truncated to the most significant L bits after the sum of products operation.
- *No bit-serial activation function.* The activation function requires that the L bits all be present before beginning its operation. A number of implementations use a lookup table approach that requires all the input bits to be present [30].
- *All inputs must be available to start another cycle.* Due to the recursive nature of equation 1 the $Y_i(t+1)$ neuron emulation can not begin until all N $Y_j(t)$ output values have been calculated.
- *Similar hardware components have the same delay and cost.* Hardware of a given type has the same delay and cost regardless of which architecture it is in, i.e. an adder has the same delay and cost in all architectures.
- *Bit serial operations.* A large number of neurons are usually emulated, this in turn imposes a need for a large amount of hardware. Without loosing the generality of this study, we have assumed bit serial implementation as a consideration for practical implementation.

The focus of this evaluation approach is aimed at establishing the architectural impact on performance. Emulators need to emulate the entire network before another iteration can start; this is due to the recursive nature of Equation 1. We have identified the delay incurred to emulate the entire network once. In this delay calculation, all the conditions such as setup, computation and storage must be taken into account. The delay is incurred along the critical path of the architecture. By determining this critical path from a study of the architecture, the delay along this path during the emulation of the network can be calculated.

To avoid technology and implementation issues we express all delays in terms of functional units rather than in terms of time. If this delay is expressed in terms of time two implementations of the same architecture may show different performances. The delays are represented by the symbol δ, with a subscript indicating the operation associated with the delay. The subscripts used in this study are: A (adder), M (multiplier), AF (activation function), and B (communication). Other delays will be discussed in the organizations for which they are relevant.

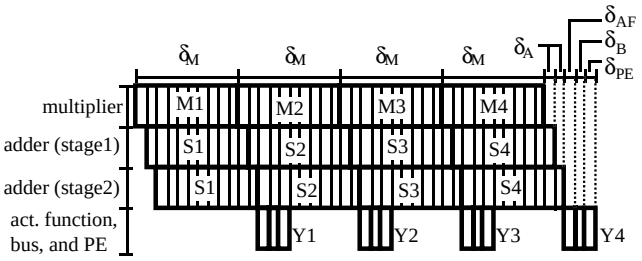


Figure 7: SPIN Delay Diagram

$$N\ \delta_M\ +\ \delta_A\ \log_2 N\ +\ \delta_{AF}\ +\ \delta_B\ +\ \delta_{PE} \tag{2}$$

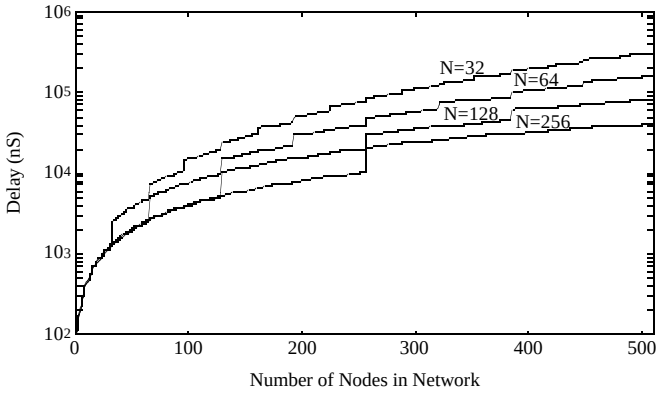
3.2 Virtual Emulation

Virtual emulation is required when the number of neurons(m) of the ANN being emulated is larger than the machine's physical neuron capacity (N). For this virtual emulation mode of operation, N of the m inputs are multiplied by their corresponding weights. These products are added together and this partial sum is stored in an accumulator. The subsequent partial sums are added to the accumulator until all m inputs have been processed. This total sum will then be passed to the activation function resulting in the output value for that neuron. Only N of the m neurons have been emulated, so the process continues iteratively, in the manner described above, with the hardware emulating the next N neurons until all m neurons have been emulated.

$$m\left\lceil\frac{m}{N}\right\rceil\ \delta_M\ +\ \delta_A\ \log_2(N+1)\ +\ \delta_{AF}\ +\ \delta_B\ +\ \delta_{PE}$$

In order to show how the proposed architecture scales as the number of neurons increases, we have plotted four different machine sizes. Figure 8 shows the recall delays for 32-, 64-, 128-, and 256- neuron machines.

Figure 8: Recall Delay for Different Machine Sizes.



When the problem size is smaller than the machine, SPIN-L provides a delay that is dependent on the problem size (in a linear fashion). When the problem size is larger than the physical machine, the size of the machine has a small impact on the delay. At the point when the machine is used in virtual mode, the delay remains within the same order of magnitude.

4 ANN Learning Algorithm Mapping

SPIN-L has been designed as a general purpose neuro-emulator. Thus, in this context the architecture is flexible enough to accommodate a large number of learning algorithms. Without losing generality, we have mapped four learning algorithms onto the proposed neuro-emulator to show the flexibility and potential uses of the structure. The four learning algorithms included in this study are: Self-Organizing Map, Adaptive Resonance Theory 1, Backpropagation, and Hopfield algorithm. We have chosen these four algorithms because they not only are widely used but also present a diverse set of computing requirements. The mapping of the Self-Organizing Map is presented in this section; the mappings of the other three algorithms are contained in the Appendix.

To show the strength of the machine we present the mappings of the four algorithms onto the machine with the operations fully pipelined. The mappings consist of the required operations proceeded by a number and a label. The label naming conventions used in the mappings are presented in Table 1.

In the mappings the number indicates operations that occur simultaneously during pipelined operation, and the label corresponds to a particular section of hardware. Lower case labels indicate units within a main section that are operating in a pipelined fashion. The labels that are bold are the major contributors to the execution delay, units within sections will appear in italics to indicate the same. The mapping and SPIN-L resource utilization for each algorithm is explained below.

4.1 Self-Organizing Map

The self-organizing map (SOM) algorithm [11] is an example of an unsupervised learning algorithm. The SOM is a clustering algorithm which classifies input patterns based on their

Table 1: Mnemonics for Algorithm Mappings.

Sect.	Unit	Description
PPC		Parallel Processing Cell
	mem	Memory Unit Access
	rf	Register File
	mlt	Multiplier
	upd	Update Unit
	sub	Subtraction Unit Before Multiplier
	ffb	Feed Forward Bus
PCT		Processing Communicating Tree
	frw	Operation in Forward Direction
	rev	Operation in Reverse Direction
PNB		PN-Bus
MFG		Multi-Function Generator
	eval	Evaluate Function
	arth	Arithmetic Operation before M-FG
	mlt	Multiplier before M-FG
IS		Intermediate Storage after M-FG
	mem	Intermediate Storage Access
	mlt	Multiplier before IS
	sub	Subtraction unit before IS

similarities. The neurons are placed in an ordered topology which defines the map. Each weight vector represents a point in the feature space described by the map topology. The location of the winning neuron in the map topology indicates which cluster the input pattern belongs to. The neurons in the map take on a spatial ordering, with respect to the clusters, that is preserved as learning progresses [11]. This clustering approach is accomplished by an algorithmic weight updating process. The weights of neurons that are physically near the winning neuron, as described by the map topology, are updated. The amount of change of each weight vector depends on its proximity to the winning neuron as dictated by the neighborhood function. A more detailed description of the algorithm is presented in the literature [11].

4.1.1 SOM Algorithm Brief Description

The ordering of the weights in the map topology, one-, two- or n-dimensional is an input variable. The location of each weight vector in the pre-determined topology will be down loaded from the host at the beginning of execution. The locations will be specified in n-tuples with each element corresponding to a dimension of the map topology. These arrays will be stored at the sequential end of the architecture. Stored with the locations are tags indicating the weight vector they correspond to. The SOM algorithm uses the angle between the weight and input vectors to determine the closest match. If the lengths of the weight vectors are normalized a direct comparison of the dot products can be utilized to determine

the weight vector closest to the input vector using this criterion [15]. To utilize this method the weights will be normalized according to the following formula:

$$\tilde{W}_i \;=\; W_i \,\frac{1}{\sqrt{\sum_j \; W_{ij}^2}} \tag{3}$$

The normalized weights are then used in the calculation of the dot product. Competitive learning using the dot products selects the winning node. During learning the weight vectors are then updated according to the following equation:

$$W(t+1) \;=\; W(t) \;+\; \mathcal{F}(N_i,\; N^*)\;[x(t) \;-\; W(t)] \tag{4}$$

where $\mathcal{F}(N_i, N^*)$ is the neighborhood function. N_i is the node being updated and N* is the winning node. We assume the neighborhood function is Gaussian of the form:

$$\mathcal{F}(N_i,\; N^*) \;=\; \eta(t)\; e^{\left(\frac{-\|r_i \;-\; r^*\|^2}{\sigma(t)^2}\right)} \tag{5}$$

where r represents the weight locations in the map, $\sigma(t)$ is the time variant width parameter and $\eta(t)$ is the time variant learning rate. The form of the neighborhood function shows that as the distance from the winner increases the weight update amount will decrease.

4.1.2 SOM Algorithm Mapping onto SPIN-L

In this section, we present a mapping of the SOM algorithm onto the SPIN-L organization. For this mapping we have manipulated the equations of the SOM algorithm to fully take advantage of SPIN-L's features. These calculations are scheduled to maximize utilization and minimize execution time. The algorithm is implemented as follows: calculate the normalizing terms for the weight vectors, normalize the original weight vectors, calculate the dot product, perform the WTA function, calculate the neighborhood size, and update the weight vectors. Each step is performed for all values before moving on to the next operation, i.e. the normalizing term is calculated for all weight vectors before any weight vector elements are normalized. The mapping, following the conventions described in the introduction of this section, is shown below where idle hardware is not included:

- Normalization

1. **PPC** Each element of the weight vector is passed to both inputs of its respective multiplier.

   PCT The bit-streams of the products, as they are calculated, are passed to the which performs the summation.

   MFG The sum of squared terms is used as the input to the M-FG which evaluates the inverse square root function.

   IST The normalizing terms are stored in the intermediate storage after the M-FG.

2. **PPC**

mlt The weight vector elements are used as one input to the multiplier. The normalizing term is broadcast back through the PCT, and is used as the other input.

upd The bit-streams of the products, as they are calculated, are passed through the update unit and stored in the memory units.

PCT The normalizing term is broadcast from the intermediate storage through the PCT to all leaves.

- Calculation of the Dot Products

3. **PPC** The elements of the input vector are multiplied by the corresponding elements of the weight vector. The inputs are passed from the RFs and the weights from the memory units.

   PCT The bit-streams of the products, as they are calculated, are passed to the PCT which performs the summation.

   PNB The output of the neuron is passed back on the PN-Bus to the last RF. The outputs, as the next output is passed back, are shifted up through the RFs using the inter-RF data path.

- Winner Takes All (WTA) Function

4. **PPC** The dot products are passed around the multipliers to the PCT using the by-pass path with no delay.

   PCT The dot products enter the PCT MSB first. The WTA operation is performed by comparing the input bits. While equal the bit is passed on through the PCT. If unequal the PCC sets the path to the larger input, effectively blocking the smaller.

- Calculation of Neighborhood Size

5. **PNB** The location of the winning node in the map topology is passed to the RFs, each coordinate in a separate contiguous RF.

6. **PNB** The location of each node is passed to the RFs as was done with the winner in a pipelined manner.

7. **PPC**

   sub The coordinates of both will be passed to the subtraction unit, where the non-winner will be subtracted from the winner.

   mlt The bit-streams of the differences, as they are calculated, are passed to both inputs of the multipliers.

   PCT The bit-streams of the squared terms, as they are calculated, are passed to the PCT which performs the summation operation.

   MFG This scaled value is used as the input to the M-FG which evaluates the exponential function.

IS The square of the linear distance, between the winner and non-winner being evaluated, is multiplied by the inverse of the squared neighborhood width parameter.

IS The result of the exponential function evaluation is multiplied by the learning rate and stored in the intermediate storage after the M-FG.

- Weight Update

8. **PPC**

sub The elements of the weight vector are subtracted from the input vector elements using the subtraction unit before the multipliers.

mlt The bit-stream of the difference, as it is calculated, is passed to one input of the multipliers. The other input is the neighborhood function value, which is broadcast back through the PCT.

upd The bit streams of the products, as they are calculated, are passed to the update unit. Here they are added to the original weight values.

PCT The neighborhood function value is broadcast from the intermediate storage back through the PCT to the multipliers.

In this subsection we have shown how the SOM algorithm can be mapped to the SPIN-L architecture. The manipulation of the scheduling of operations allows the machine to maintain a highly pipelined implementation of the SOM algorithm resulting in good performance. This is made possible by fully utilizing all aspects of the hardware present. The parallel portion of the machine performs the required operations a vector at a time rather than performing element by element calculations. Performance for the algorithm implemented as described is reported in Section 5.

5 Evaluation

The evaluation of the proposed architecture is based on functional delays as described in Section 3. To compare this architecture with other neuro-emulators that have been suggested in the literature we consider the use of a benchmark commonly used for neural network evaluation. We will present first an evaluation of SPIN-L based on resource delay in the critical path. This is shown by deriving the delay equation for the ART-1 learning algorithm. The delay equations for the three other learning algorithms are then presented. This is followed by an evaluation based on million connection updates per second (MCUPS) using the NETtalk routine as a benchmark.

5.1 Delay Estimation

The implementation of the ART-1 algorithm using SPIN-L is highly pipelined. This results in the overlapping of many operations. The algorithm suffers the delay of only one of these operations as will be shown in the derivation of the delay equation. The resource delays will be represented by the symbol δ and a subscript indicating the resource. The

multiplier delay will have a **M** subscript the PCC will have an **A**, the function generator an **AF**, the PN bus a **B** and the delay incurred in storing a value will be denoted by the subscript **STORE**. We will assume for the purposes of clarity that the number of nodes present in hardware equals the number required by the algorithm, which is equal to N.

Binary ART Delay Estimation				
Algorithm step		Functional Unit	Operation	User Delay
I.		PPC	X * W_{TD}	$N(\delta_M + \delta_A \log_2 N)$
	1.	PCT	$\sum$ 1a.	
	1.	PNB	PN-Bus results from 1b	
	2.	PPC	W_{TD} bypass multipliers	
II.		PCT	$\sum W_{TD}$	$(NL + \log_2 N)\ \delta_A$
	2.	MFG		
		arth	$\epsilon + \sum W_{TD}$	
		fev	inverse of 2c	
	2.	PNB	PN-Bus normalizing term from 2c	
	3.	PPC	Input vector elements bypass multipliers	
III.		PCT	$\sum$ X	$(NL + \log_2 N)\ \delta_A$
	3.	MFG	$\rho * \sum$ X	
IV.		PPC	(X * W_{TD}) * ($\sum W_{TD}$)	δ_M
	4.	PCT		
		frw	PCT performs WTA function	
		rev	Broadcast scaled input vector to all RFs	
Va.		PPC	($\rho * \sum$ X) - (X * W_{TD})	$L\ \delta_A$
	5.	PCT		
		frw	communication through PCT	$\delta_A \log_2 N$
Vb.		MFG	Vigilance test (compare to 0)	δAF
	6.	PPC		
		pud	X .AND. W_{TD}	$L\ \delta_A$
		pstr	Store new weight vector	δ_{STORE}

We have presented the ART-1 algorithm to show the strength of SPIN-L's pipelined processing and to address the issues of delay prediction in this style of operation. In calculating the delay we were only concerned with getting values to the multipliers or the first stage of the adder tree. This is not to say that the processing stopped there, but rather the next operation at the multiplier would overlap any further operations on those values therefore absorbing the delay.

We now present in Table 2 the delay equations for the learning algorithms studied in this paper, for Backpropagation we assume a two layer network.

Table 2: Delay Equations for Learning Algorithms

ALGORITHM	DELAY EQUATION
Self-Organizing Map	$(5N)\delta_M \ + \ ((3N+1)\log_2 N + 3)\ \delta_A \ + \ (2nL+1)\ \delta_B \ + \ \delta_{STORE}$
Binary ART	$(N+1)\ \delta_A \ + \ ((N+3(\log_2 N))\ \delta_A \ + \ \delta_{AF} \ + \ \delta_{STORE}$
Backpropagation	$(N+1)\ \delta_M \ + \ ((N+3)(1+\log_2 N))\ \delta_A \ + \ \delta_{AF} \ + \ \delta_{STORE}$
Hopfield	$0.5(N^2+N+2)\ \delta_M \ + \ (\log_2 N)\ \delta_A \ + \ \delta_B \ + \ \delta_{STORE}$

5.2 Virtual Neuron Emulation

In processing ANNs, as in other computing paradigms, it is often necessary to provide a mechanism to deal with larger problems than those that can be directly handled by the actual hardware. This results in greater flexibility at a lower cost without restricting the complexity of problems that can be addressed with a machine of set size. The proposed machine, and its predecessor, have been shown to be capable of virtual emulation [36][37]. Below we provide a description of how the SOM algorithm could be implemented in virtual mode. The steps outlined here have been explained in detail in Section 4.1.2.

1. Calculate the normalizing term for 1st (next) N elements of 1st (next) N weight vectors.
2. Multiply normalizing terms by corresponding weight vector elements.
3. Calculate dot product for 1st (next) N weight vectors.
4. Perform WTA.
5. Store winner.

   Repeat steps 1-5 for all weight vectors N at a time.
6. WTA between winners from step 4 in sets of N until only one winner.
7. Pass winner's coordinates in map topology to RFs.
8. Pass coordinates of 1st (next) node in map topology to RFs.
9. Calculate neighborhood function.

   Repeat steps 8-9 for 1st N weight vectors.
10. Multiply neighborhood function by difference between input and weight vectors.
11. Update weight vectors.

    Repeat steps 10-11 for 1st N weight vectors.

    Repeat steps 8-11 for all weight vectors.

The virtual emulation delay equations for the learning algorithms that have been studied in this paper are shown below. In the equations that follow m is the number of neurons required to be emulated and N is the number of neurons the hardware is capable of emulating directly. We now present the delay equation for the self organizing map during virtual emulation. In Equation refeq:virt-som-dely n is the map dimension size.

$$\begin{aligned} & 5N \left\lceil \frac{m}{N} \right\rceil^2 \ \delta_M \ + \ \left[\ \left\lceil \frac{m}{N} \right\rceil \left(\left\{ 3N \left\lceil \frac{m}{N} \right\rceil + 1 \right\} \log_2 N + L + 1 \right) \ + \right. \\ & \left. \left(\left\lceil \frac{m}{N_2} \right\rceil L + \log_2 N + 2 \right) \right] \ \delta_A \ + \ \left(2nL + \left\lceil \frac{m}{N} \right\rceil \right) \ \delta_B \ + \ \delta_{STORE} \end{aligned} \tag{6}$$

Equation refeq:virt-art-dely shows the delay equation for the Adaptive Resonance Theory as it could be implemented under virtual emulation conditions using the SPIN-L architecture.

$$\begin{aligned} & \left\lceil \frac{m}{N} \right\rceil^2 (N+1) \ \delta_M \ + \ \left[\left\{ \left(\left\lceil \frac{m}{N} \right\rceil + 1 \right)^2 + N \left\lceil \frac{m}{N} \right\rceil^2 \right\} \right. \\ & \left. (L + \log_2 N) + \left\lceil \frac{m}{N} \right\rceil^2 \right] \ \delta_A \ + \ \delta_{AF} \ + \ \delta_{STORE} \end{aligned} \tag{7}$$

The Backpropagation algorithm is used in the performance comparison that follows. In this comparison we report performance values for machine sizes that require the algorithm to be executed in virtual mode. For this reason we have expanded it for the sake of clarity and reproducibility of our results. The delay has been split into a delay for recall, calculation of the error for all layers, and the weight update process. In Equations refeq:virt-bp-dely Vm and Vm-1 represent the number of neurons in layer m and m-1, respectively.

$$\delta_{Recall} \ = \ \sum_{M=2}^{n} \left(V_M \left\lceil \frac{V_{M-1}}{N} \right\rceil \ \delta_M \ + \ V_M \left\lceil \frac{V_{M-1}}{N} \right\rceil \log_2 N \ \delta_A \right) \tag{8a}$$

$$\delta_{Error} \ = \ \sum_{M=3}^{n} \ V_M \left\lceil \frac{V_{M-1}}{N} \right\rceil \ \delta M \tag{8b}$$

$$\delta_{Update} \ = \ \sum_{M=2}^{n} \ \left(V_M \left\lceil \frac{V_{M-1}}{N} \right\rceil \ \delta_M \ \right) \ + \ \delta_A \ + \ \delta_{STORE} \tag{8c}$$

The SPIN-L architecture can still take advantage of the symmetry found in the weight matrix of the Hopfield network. In Equation refeq:virt-hop-dely, which indicates the delay of the Hopfield algorithm in virtual emulation mode, n is the number of bits in the input patterns.

$$\left(1 + \left\lceil \frac{m}{N} \right\rceil \ \sum_{K=1}^{n} K \ \right) \ \delta_M \ + \ (1 + \log_2 N) \ \delta_A \ + \ \delta_B \ + \ \delta_{STORE} \tag{9}$$

In this section we have presented the delay equations for SPIN-L's virtual emulation performance. The description of the SOM algorithm shows that the architecture continues to fully utilize the hardware present under the rigorous demands of virtual emulation.

6 COMPARISON

In order to compare the proposed SPIN-L architecture to other machines we have chosen the NETtalk network which is a widely used benchmark. The NETtalk network is a three layer feed forward network. The configuration we use for comparison purposes consists of 203 input neurons, 60 hidden neurons and 26 output neurons. The neurons in the hidden and output layers each have an additional weight to implement the variable threshold. This configuration results in 12,180 connections between the input and hidden layers, 1560 connections between the hidden and output layers and the 86 threshold connections. Overall there are 13,826 variable connections.

It has been suggested that all the resource delays can be expressed in terms of an adder delay. The multiplier delay is $(L + \log_2 N)$ adder delays, the function generator is 2L adder delays, the bus and store delays are equivalent to one adder delay each. Using 10 nsec. as the adder delay, a typical value for current technology, we report the performance in terms of Millions of Connection Updates Per Second (MCUPS).

The available resources do not match the number of neurons for the SPIN-L machine sizes 64 and 128 reported. Thus, the architecture operates in a virtual mode for these machine sizes. Equations 16.A, 16.B and 16.C are used to determine the delay under NETtalk network constraints. The delay value is then used to calculate the number of connection updates that could be performed per second running this benchmark. The performance of SPIN-L and other reported neuro-emulators are shown in Table 3.

Table 3: Comparison of Neuro-emulators implementing NETtalk 203-60-26

NEURO-EMULATOR	NEURONS IN HARDWARE	NETtalk: MCUPS
SPIN-L 256 bit-serial	256	358
SPIN-L 128 bit-serial	128	226
SPIN-L 64 bit-serial	64	131
BP-Systolic	13 K	248
CNAPS	64	160
DAP-610	4 K	160
Sandy / 6	64	135
Sandy / 8	256	135
Cellular Array	4 K	51.5
CM-2	64 K	40
iWarp	10	36
Warp	10	17
CM-1	64 K	13

From Table 3 the following observations can be made. A SPIN-L 256 bit serial architecture provides the highest MCUPS performance. For the NETtalk benchmark this SPIN-L machine size outperforms all the other neuro-emulators even those with a much larger number of processors. A SPIN-L 64 bit-serial performs extremely well using a modest amount of hardware. SPIN-L 64 provides better performance than other much larger systems such as the Cellular Array, CM-2 and CM-1. Machines such as CNAPS and SANDY/6 have a

similar number of physical neurons; however they have parallel word implementations. This in turn imposes a need for much larger hardware; CNAPS and SANDY/6 may utilize over three times the hardware required for SPIN-L. Using distributed weight vectors requires an increase in memory by a factor of N, as well as increased communications to maintain coherency. SPIN-L has an extremely good scalability. It can be observed that increasing the size of the machine provides performance gains. In this example, we did not consider machine sizes above 256 since the machine becomes underutilized. SPIN-L can accommodate this mode of operation easily, however for larger machine sizes the performance will be of the same order as that for the SPIN-L 256.

7 Concluding Remarks

In this paper we have identified the major computational and communication requirements for a large set of ANN learning algorithms. These algorithms span many styles of learning: supervised and unsupervised, multi-layered, spatial, etc. This set of requirements has been used in the design of a novel neuro-emulator, called SPIN-L. This neuro-emulator operates in a pipeline fashion. This in turn along with SPIN-L's parallel, reduction, and multi-function generator structure provide for high performance. We have reported the performance of SPIN-L on four different learning algorithms that represent a wide range of requirements as well as learning approaches. These algorithms are: Self-Organizing Map (which requires competitive learning, and the ability to determine node locations within the map), Adaptive Resonance Theory (requiring vigilance testing, and the ability to iteratively disable nodes in competitive learning), Backpropagation (which utilizes the transpose of the weight matrix during the update process, a separate weight matrix for each layer, and the ability to implement a sparsely connected network), and the Hopfield algorithm (the performance of which can be improved with the ability to store weight values to all symmetric locations, and requires massive communications due to its fully connected nature). We have compared SPIN-L with other machines executing NETtalk (203 inputs, 60 hidden, and 26 output neurons). In this comparison SPIN-L with 64 neurons (directly implemented in hardware) performs extremely well achieving 131 MCUPS. Its similar performance is similar to other machines with a much larger number of processing elements. SPIN-L accommodates a large number of complex functions in the most cost effective way by means of a single multi-function generator. This in turn makes SPIN-L the neuro-architecture with the least amount of hardware among the machines with similar capabilities.

References

[1] Z. Zhao and C.G. Rowden, "Use of Kohonen Self-Organizing Feature Maps for HMM Parameter Smoothing in Speech Recognition," *IEE PROCEEDINGS-F*, Vol. 139, No. 6, pp. 385-390, December 1992.

[2] L. Gordon Kraft and David P. Campagna, "A Comparison Between CMAC Neural Network Control and Two Traditional Adaptive Control Systems," *IEEE Control Systems Magazine*, pp. 36-43, April 1990.

[3] D. H. Nguyen and B. Widrow, "Neural Networks for Self-Learning Control Systems," *IEEE Control Systems Magazine*, pp. 18-23, April 1990.

[4] M. J. Healy, T. P. Caudell, and Scott D. G. Smith, "A Neural Architecture for Pattern Sequence Verification Through Inferencing," *IEEE Transactions on Neural Networks*, Vol. 4, No. 1, pp. 9-20, January 1993.

[5] I. A. Bachelder and A. M. Waxman, "Mobile Robot Visual Mapping and Localization: A View-Based Neurocomputational Architecture that Emulates Hippocampal Place Learning," *Neural Networks*, Vol. 7, No. 6, pp. 1083-1099, 1994.

[6] T. Warren Liao and Lian Jiang Chen, "An Evaluation of ART1 Neural Models for GT Part Family and Machine Cell Forming," *Journal of Manufacturing Systems*, Vol. 12, No. 4 pp. 282-290, 1993.

[7] J. M. Vincent, J. B. Waite and D. J. Myers, "Automatic Location of Visual Features by a System of Multi-Layered Perceptrons," *IEE Proceedings-F*, Vol. 139, No. 6, pp. 405-412, December 1992.

[8] S. Cavalieri, A Di Stefano and O. Mirabella, "Optimal Path Determination in a Graph by Hopfield Neural Network," *Neural Networks*, Vol. 7, No. 2, pp. 397-404, 1994.

[9] J. J. Hopfield and Tank," 'Neural' Computation of Decisions in Optimization Problems," *Biological Cybernetics*, Vol. 52, pp. 141-152, 1985.

[10] J. H. Park, Y. S. Kim, I. K. Eom and K. Y. Lee, "Economic Load Dispatch for Piecewise Quadratic Cost Function Using Hopfield Neural Network," *IEEE Transactions on Power Systems*, Vol. 8, No. 3, pp. 1030-1038, August 1993.

[11] T. Kohonen, "The Self-Organizing Map," *Proceedings of the IEEE*, Vol. 78, No. 9, pp. 1464-1480, September 1990.

[12] G. A. Carpenter and S. Grossberg, "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network," *Computer*, Vol. 21, No. 3, pp. 77-88, March 1988.

[13] D. E. Rumelhart, J. L. McClelland and the PDP group, *Parallel Distributed Processing, vol. 1 : Foundations*. Cambridge, Mass: MIT Press, 1986.

[14] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Nat. Acad. Sci.*, pp. 2554-2558, April 1982.

[15] Howard Anton, *Calculus with Analytic Geometry*, New York: John Wiley & Sons, 1984.

[16] J. Chung, H. Yoon, and S. R. Maeng, "A Systolic Array Exploiting the Inherent Parallelisms of Artificial Neural Networks," *Microprocessing and Microprogramming 33*, pp. 145-159, North-Holland, 1991-1992.

[17] D. W. Hammerstrom, and D. P. Lulich, "Image Processing Using One-Dimensional Processor Arrays," *Proceedings of the IEEE*, pp. 1005-1018, July, 1996.

[18] F. J. Nunez, and J. A. B. Fortes, "Performance of Connectionist Learning Algorithms on 2-D SIMD Processor Arrays," in D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pp. 810-817, Morgan Kaufmann, 1990.

[19] H. Kato, H. Yoshizawa, H. Iciki, and K. Asakawa, "A Parallel Neurocomputer Architecture with Ring Registers," *InfoJapan '90: Information Technology Harmonizing with Society*, Vol. 1, pp. 233-240. Tokyo, Japan: North Holland, October 1990.

[20] B. Faure and G. Mazare, "A VLSI Implementation of Multi-Layered Neural Networks: 2-Performance," in J. G. Delgado-Frias and W. R. Moore, (Eds.), *VLSI for Artificial Intelligence and Neural Networks*, pp. 377-386, Plenum Press, 1991.

[21] X. Zhang, M. Mckenna, J. P. Mesirov, and D. L. Waltz, "An Efficient Implementation of the Back-propagation Algorithm on the Connection Machine CM-2," in D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pp. 801-809, Morgan Kaufmann, 1990.

[22] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Boore, C. Peterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski, and J. Webb, "iWarp: An Integrated Solution to High-Speed Parallel Computing," *Proceedings Supercomputing '88*, pp. 330-339, Orlando, Fl.: IEEE Computer Society Press, November 1988.

[23] D. A. Pomerleau, G. L. Gusciora, D. S. Touretzky, and H. T. Kung, "Neural Network Simulation at Warp Speed: How We Got 17 Million Connections Per Second," *IEEE International Conference on Neural Networks*, Vol. II, pp. 143-150, July 1988.

[24] C. R. Rosenberg and G. Blelloch, "An Implementation of Network Learning on the Connection Machine," in D. Waltz and J. A. Feldman, editor, *Connectionist Models and Their Implications: Readings From Cognitive Science*, pp. 329-340, Ablex Publishing Corp., 1988.

[25] S. Vassiliadis, G. G. Pechanek and J. G. Delgado-Frias, "SPIN: The Sequential Pipelined Neuroemulator," *International Journal of Artificial Intelligence Tools*, Vol. 2, No. 1, pp. 117-132, 1993.

[26] S. Vassiliadis, G. G. Pechanek, and J. G. Delgado-Frias, "SPIN: A Sequential Pipelined Neurocomputer," *Proceedings of the 1991 IEEE Int. Conf. on Tools for Artificial Intelligence*, pp. 74-81, San Jose, Calif., Nov.1991.

[27] G. G. Pechanek, S. Vassiliadis, J. G. Delgado-Frias, "Digital Neural Emulators using Tree Accumulation and Communication Structures," *IEEE Trans. on Neural Networks*, vol. 3, no. 6, pp. 934-950, Nov. 1992.

[28] J. G. Delgado-Frias, S. Vassiliadis, G. G. Pechanek, W. Lin, S. Barber, and H. Ding, "A VLSI Pipelined Neuroemulator," *VLSI for Neural Networks and Artificial Intelligence*, J. G. Delgado-Frias and W. R. Moore (Eds.), pp. 71-80, New York: Plenum, 1994.

[29] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Nat. Acad. Sci*, pp.2554-2558, April 1982.

[30] P. Treleaven, M. Pacheco, and M. Vellasco, "VLSI Architectures for Neural Networks," *IEEE Micro*, vol. 9, no. 6, pp.8-27, Dec. 1989.

[31] G. A. Carpenter, S. Grossberg and D. B. Rosen, "ART 2-A: An Adaptive Resonance Algorithm for Tapid Category Learning and Recognition," *Neural Networks*, Vol. 4, pp. 493-504, 1991.

[32] G. A. Carpenter and S. Grossberg, "ART 3: Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures," *Neural Networks*, Vol. 3 pp. 129-152, 1990.

[33] G. A. Carpenter and S. Grossberg, "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics, and Image Processing*, Vol. 37, pp. 54-115, 1987.

[34] B. Widrow and M. A. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation," *Proceedings of the IEEE*, Vol. 78, no 9, pp. 1415-1441, September 1990.

[35] R. P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE Acoustics, Speech, and Signal Processing Magazine*, pp. 4-22, April 1987.

[36] V. C. Aikens II, J. G. Delgado-Frias, G. G. Pechanek, S. Vassiliadis, "An Evaluation of Performance and Hardware Requirements for Neuro-Emulators," *IBM Technical Report TR51.0831*, IBM Austin TX, March 1994.

[37] V. C. Aikens II, J. G. Delgado-Frias, G. G. Pechanek, S. Vassiliadis, "A Neural Processor with Learning Capabilities," *Technical Report, Department of Electrical Engineering, Delft University*, Delft, The Netherlands, 1997.

A Adaptive Resonance Theory 1

There have been multiple forms of the adaptive resonance theory (ART) algorithm first proposed by Carpenter and Grossberg [12][31][32][33]. As an example of this style of learning we have studied the binary (ART-1) form for this paper. The ART-1 algorithm is an unsupervised clustering algorithm similar to the SOM. The ART-1 algorithm searches for more than just the weight vector closest to the input vector. It uses a vigilance test to insure that the chosen weight vector is similar enough to the input vector that it can be considered the correct classifier. To implement this algorithm we have assumed the F2 layer performs the

WTA function. A more complete description of the algorithm is given by Carpenter and Grossberg [12].

A.1 ART-1 Algorithm Description

Like the Self-Organizing Map, ART-1 can use the dot product to determine the weight vector closest to the input vector. The weights used in this dot product will be the bottom up weights (W_{BU}), which are the top down weights (W_{TD}) normalized. The weights will be normalized using the following equation:

$$\tilde{W}_i \;=\; \frac{W_i}{(\epsilon \;+\; \sum\; W_{ij})} \tag{10}$$

These values are used to form the dot products with the input vector. The dot products then undergo competitive learning to determine the weight vector closest to the input vector. The winning vector will be considered to be close enough to the input vector if it satisfies the vigilance criterion. The vigilance criterion has been stated as [12]:

$$\rho \;=\; \frac{\|V_i\|}{\|X\|} \tag{11}$$

where Vi is the vigilance vector described as:

$$V_i \;=\; X_i \;\bullet\; W \tag{12}$$

We have manipulated the vigilance equation to allow the calculations required to take full advantage of the SPIN-L organization. The actual vigilance equation that will be implemented using the SPIN-L machine will be:

$$0 \;<\; \rho \sum_i\; X_i \;-\; \sum_i\; V_i \tag{13}$$

When a weight vector passes the vigilance test it will be considered the correct classifier. It will then be updated to reflect the inclusion of the present input in the cluster it represents. If the winner fails the vigilance test it is not considered the correct classifier and competitive learning will occur again with the previous winner(s) disabled.

A.2 ART-1 Algorithm Mapping onto SPIN-L

To implement this algorithm efficiently we exploit the pipeline ability of the organization thereby reducing the overall delay by overlapping operations continually. The equations for this algorithm have been broken into steps and scheduled to take advantage of the machine's abilities. The W_{TD} will be down loaded by the host initially. The implementation proceeds as follows: calculate the dot product of the top down weights and the input vector, calculate the normalizing term, calculate the scaled norm of the input vector, perform t6he WTA function, apply the vigilance test, and update the weights. Each step will be performed on all values before proceeding to the next step. The mapping, following the conventions described in the introduction of this section, of the ART algorithm is as follows:

- Dot Product of W_{TD} and X

1. **PPC**

   mlt The input vector and the top down weight vector are passed to the multiplier from the RFs and the memory units respectively.

   rf Previously calculated values are passed up to the next RF using the inter-RF data path.

   PCT The bit-streams of the products, as they are calculated, are passed to the PCT which performs the summation.

   MFG The M-FG is idle (idle hardware will not be shown further).

   PNB The sum-of-products is passed back to RFN using the PN-Bus.

- Normalizing Term

2. **PPC**

   mlt The top down weight vector elements are passed around the multiplier with no delay using the bypass path.

   rf Previously calculated values are passed up to the next RF using the inter-RF data path.

   PCT The PCT performs the summation of the weight vector elements.

   MFG

   arth The time variant value of ϵ is added to the sum of the weight vector elements.

   eval This sum is used as the input to the M-FG which evaluates the inverse function.

   PNB The result of the inverse function is passed back to RF_N using the PN-Bus.

- Norm of Input Vector

3. **PPC** The input vector elements are passed around the multiplier with no delay using the bypass path.

   PCT The PCT performs the summation of the input vector elements.

   MFG The sum of the input vector elements is scaled by the time variant value of ρ.

   IS The result of the multiplication is store in intermediate storage after the M-FG.

- Winner Takes All (WTA) Function

4. **PPC** The dot products calculated in step 1 are multiplied by the normalizing terms calculated in step 2.

   PCT

frw The normalized dot products enter the PCT MSB first. The WTA operation is performed by comparing the input bits. While equal the bit is passed on through the PCT. If unequal the PCC sets the path to the larger input, effectively blocking the smaller. When the winner reaches the root of the tree the only path to a leaf is to the winner's PPC.

rev The normalizing terms calculated in step 3c are broadcast back through the PCT to all RFs.

5. **PPC** The norm of the input vector calculated in step 3 and the dot products calculated in step 1 are passed to the subtraction units, where the dot products are subtracted from the norm of the input.

   PCT The differences are all passed to the PCT, however only the winner has an open path to the root of the tree.

   MFG This value is passed to the M-FG which compares it to zero, performing the vigilance test. Note: If the vigilance test is failed the current winning node is disabled by blocking its path in the first stage of the PCT and the WTA function is performed again. This is repeated until a winning node passes the vigilance test. We continue now assuming the test was passed.

- Weight Update

6. **PPC** The elements of the input vector are passed to the update units where they are added to the top down weight vector elements. The carry out of the adder is equivalent to the logical AND function, the carry out is not propagated back into the next addition.

In this subsection we have shown how the ART-1 algorithm can be mapped to the SPIN-L architecture. It is clear from this description that the mapping fully utilizes the organization of the machine. The advantages of the parallel pipelined nature of the machine are also apparent from the description. Performance for the algorithm implemented as described is reported in Section 4.

B Backpropagation

Backpropagation is one of the most widely used learning algorithms for multi-layered neural networks [34]. Its dependence on intermediate values from other layers presents demands for storage as well as computational hardware. Backpropagation is a gradient descent algorithm, and as such adjusts the weights to minimize an error function. Its strength is the ability to addresses the problem of assigning an error to the neurons in hidden layers. This results in two types of error, an error for neurons in the output layer (L), and an error for neurons in the hidden layers (k). A description of the algorithm in greater detail has been presented[34].

B.1 BP Algorithm Description

Assuming a sigmoid activation function of the form

$$\frac{1}{1 \; + \; e^{-(s/\alpha)}} \tag{14}$$

the error at an output layer neuron is defined as

$$\delta_i^L \;=\; \left(T_i \;-\; Y_i^L\right)\; Y_i^L \;\left(1 \;-\; Y_i^L\right) \tag{15}$$

where T_i is the desired output value for neuron i, and Y_i is the actual output value of the same neuron. The error for a neuron in a hidden layer is then defined in terms of the errors of the neurons in the layer above it as:

$$\delta_i^k \;=\; \left[\; \sum_n \; W_{ni}^{(k+1)} \; \delta_n^{(k+1)} \right] \; Y_i^{(k)} \; \left(1 \;-\; Y_i^{(k)} \right) \tag{16}$$

where n is the number of neurons in the layer k+1, and k is the current layer with the constraint that kL. The weights are adjusted in the direction of greatest descent along the error surface, defined by the Backpropagation algorithm to be

$$W_{ij}^k(t+1) \;=\; W_{ij}k(t) \;+\; \eta(t)\; \delta_i^k \; Y_j^{(k-1)} \tag{17}$$

where $\eta(t)$ is the time varying learning rate. A multi-layered network of perceptrons consisting of three layers is capable of handling any mapping between inputs and outputs, where the complexity of the mapping is limited by the number of nodes in the network [35]. This makes Backpropagation an extremely powerful algorithm while the sparse connections and the values for multiple layers make it a demanding algorithm to implement.

B.2 BP Algorithm Mapping onto SPIN-L

We will now show that even under the demands of a multi-layered network it is possible for the proposed emulator to maintain a high utilization of available hardware resources. This will be shown by examining the execution of the algorithm at the hardware resource level. The execution of the Backpropagation algorithm on the SPIN-L architecture has been previously discussed [28], so we will present here an overview of this process. The equations for the Backpropagation algorithm have not been altered. The calculations for the error equations have been distributed across the machine organization to extract a higher degree of parallelism from the overall algorithm. Without losing generality and for presentation clarity we present a multi layer network with one hidden layer. The mapping, following the conventions described in the introduction of this section, occurs as follows:

- Recall Process

1. **PPC** The input vector elements are multiplied by the weight vector elements using the N parallel multipliers. The inputs are passed from the RF units and the weights are passed from the memory units.

PCT The bit-streams of the products, as they are calculated, are passed to the PCT which performs the summation.

MFG The sum-of-products is used as the input to the M-FG which evaluates the sigmoid function.

PNB The output of the neuron is passed back to RF_N using the PN-Bus.

IS The intermediate storage is idle (idle hardware will not be shown further). This process can be applied to multiple hidden layers as well as the output layer. The outputs of the neurons in the output layer are used immediately to calculate the output layer error, they are not passed back on the PN-Bus.

- Output Layer Error

PNB The PN-Bus is idle.

MFG

sub The output of the neuron is passed to both subtraction units prior to the M-FG. The calculations (T-Y) and (1-Y) are performed here.

mlt The differences are passed to a multiplier which calculates the product in a word parallel manner.

mlt This product is passed to a second multiplier with the original neuron output calculated in 1.MFG.

IS

mem This product is stored in the intermediate storage.

mlt It is also passed to a third multiplier to be scaled by the time variant learning rate.

mem The scaled output layer error is stored in intermediate storage.

- Hidden Layer Error

2. **PPC**

mlt The output layer weight vector elements are passed to the N parallel multipliers along with the output layer error.

upd The bit-streams of the products, as they are calculated, are passed to their update units which perform the accumulation operation.

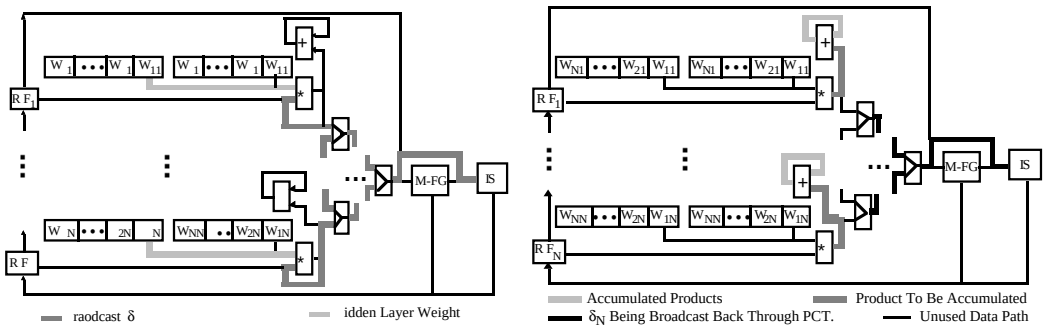
PCT The output layer error vector is broadcast, element by element, from intermediate storage through the PCT to all leaves.

- *Output Layer Weight Update / Final Steps of Hidden Layer Error*

3. **PPC**

mlt The scaled output layer errors are multiplied by the corresponding hidden layer outputs.

upd The bit-streams of the update term, as they are produced, are passed to the update units with the original weight vector elements. The sum of the two terms is the new weight vector elements for the next iteration.



(a) hidden layer error calculation 1 (b) hidden layer error calculation 2

Figure 9: Hidden layer error calculation

mem The new weight vector elements are stored in the memory unit.

ffb The partial hidden layer errors are passed to the sequential end of the machine from the first RF using the forward bus, they are shifted up to the next RF using the inter-RF bus until arriving at the first RF.

PCT The scaled output layer errors are broadcast back through the PCT to all RFs.

IS

mlt The partial hidden layer errors are multiplied by the corresponding outputs.

sub The outputs are subtracted from 1; equivalent to the derivative of the sigmoid activation function.

mlt The two values just calculated in 3d. are multiplied together.

mlt This product is then scaled by the time variant learning rate resulting in the actual hidden layer error.

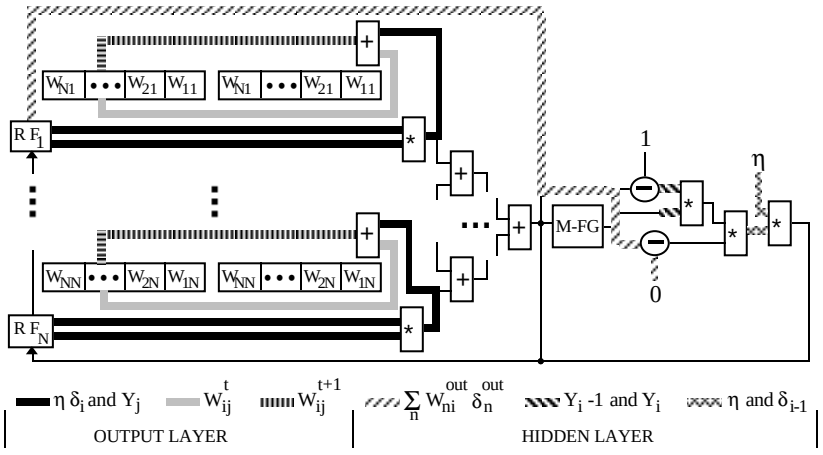


Figure 10: Output Layer Weight Update and Hidden Layer Error Calculations.

- *Hidden Layer Weight Update*

4. **PPC**

mlt The scaled output layer errors are multiplied by the corresponding hidden layer outputs.

upd The bit-streams of the update term, as they are produced, are passed to the update units with the original weight vector elements. The sum of the two terms is the new weight vector elements for the next iteration.

mem The new weight vector elements are stored in the memory unit.

PCT The hidden layer errors are broadcast through the PCT to all RFs.

By storing output values at the sequential end of the architecture this process can be extended to any number of layers. As was stated earlier three layers can create an arbitrary decision surface with the complexity limited only by the number of neurons in the network. We have not limited the architecture to this configuration and thus provide flexibility while maintaining the ability to realize the full potential of the algorithm at a reasonable cost. Performance of the algorithm implemented as described here is reported in Section 4.

C Hopfield Learning Algorithm

The Hopfield learning algorithm is a type of associative memory. The Hopfield model is an interesting implementation due to its highly interconnected structure. Each node of the Hopfield network is connected to every other node. This requires massive communication abilities of any architecture which is used to realize this model.

C.1 Hopfield Algorithm Description

The patterns are stored in the network via learning and are then available for recall. The weights are determined using the following equation:

$$W_{ij} = \frac{1}{N} \sum_{u=1}^{n} X_i^u \, X_j^u \tag{18}$$

where x_i is the i^{th} bit of the u^{th} input pattern [14]. This algorithm for implementing the Hopfield model requires that the inputs be ± 1. The resulting weight matrix will be square, K x K with K being the number of bits in each input pattern. This matrix will be symmetrical with respect to the diagonal i = j. The actual number of calculations resulting in unique values will be equal to:

$$\frac{N^2}{2} + \frac{N}{2} = \sum_{K=1}^{N} K \tag{19}$$

C.2 Hopfield Algorithm Mapping onto SPIN-L

The Hopfield algorithm is comprised of storing the patterns, by calculating the weight values, and recall. We will now show how these two functions, as they have been described above,

can be implemented using the proposed architecture. The mapping is presented using the conventions described in the introduction of this section.

- Calculation of weights.

1. **PPC** The N multiplications are carried out in parallel.

2. *PCT* The products will then be passed to the PCT where the summation will be carried out.

- Scaling of the weights.

3. *MFG* The weights are scaled using the number of neurons in the network. This multiplication occurs at the end of the PCT.

4. *PNB* This value, representing one of the weights, is passed back to the register file using the PNbus.

- Storing of weights.

5. **PPC** The hardware allows the tag to be matched as X,Y and Y,X. This results in the weights being stored in the symmetrical positions of the weight matrix.

- Recall.

6. **PPC**

7. *mlt* The input pattern is multiplied by the weight vectors.

8. *rf* The outputs, as the next output is passed back, are shifted up through the RFs using the inter-RF data path.

9. *PCT* The bit-stream of the products, as they are produced, are passed to the PCT which performs the summation operation.

10. *MFG* The M-FG is idle (idle hardware will not be shown further).

11. *PNB* The output of the neuron is passed back on the PN-Bus to the last RF.

- Winner Take All

12. **PPC** The outputs calculated in step 3. are passed around the multipliers using the bypass path.

13. *PCT* The outputs enter the PCT MSB first. The WTA operation is performed by comparing the input bits. While equal the bit is passed on through the PCT. If unequal the PCC sets the path to the larger input, effectively blocking the smaller.

14. *PNB* The winner is passed back on the PN-Bus as the network output.

The recall performance of the Hopfield model has been shown to degrade if an attempt is made to store more than N patterns, where N is the number of weights [14]. Thus we have assumed that there will be N patterns and they will be available at the beginning of execution. The Hopfield algorithm has been shown to be efficiently implemented in spite of its demanding communication requirements. This is achieved using a single bus by evaluating one neuron output at a time, removing any bus contention, with the operations highly pipelined. The performance of the Hopfield algorithm is reported in Section 4.