Design of Low-cost, Real-time Simulation Systems for Large Neural Networks

by

Mark R. James

A thesis submitted to fulfil the requirements of the degree of

Master of Science, at the University of Sydney.

JANUARY, 1992

ABSTRACT

Systems with large amounts of computing power and storage are required to simulate very large neural networks capable of tackling complex control problems and real-time emulation of the human sensory, language and reasoning systems. General-purpose parallel computers do not have communications, processor and memory architectures optimized for neural computation and so can not perform such simulations at reasonable cost. This thesis analyses several software and hardware strategies to make feasible the simulation of large, brain-like neural networks in real-time, and presents a particular multicomputer design able to implement these strategies. An important design goal is that the system must not sacrifice computational flexibility for speed, as new information about the workings of the brain and new artificial neural network architectures and learning algorithms are continually emerging.

The main contributions of the thesis are:

— an analysis of the important features of biological neural networks that need to be simulated,

— a review of hardware and software approaches to neural networks, and an evaluation of their abilities to simulate brain-like networks,

— the development of techniques for efficient simulation of brain-like neural networks, and

— the description of a multicomputer that is able to simulate large, brain-like neural networks in real-time and at low cost.

Part of this work will be published in the Journal of Parallel and Distributed Computing, Special Issue on Neural Computing on Massively Parallel Processors, March 1992.

ADDRESS FOR CORRESPONDENCE:

Mark James	Tel	: +61-2-692-4276
Basser Department of Computer Science		: +61-2-692-3423
Madsen Building, F09	Fax	: +61-2-692-3838
The University of Sydney NSW 2006	Email	: mrj@cs.su.oz.au
AUSTRALIA		

ACKNOWLEDGEMENTS

I would like to sincerely thank my research supervisor Doan Hoang for his patience, advice and support. I would also like to thank my parents for financial support, Louie Esteban for his help and friendship, and Anne Bolding, librarian of Burkitt Library, The University of Sydney for her friendly assistance.

The preliminary stages of this research were supported by an Australian Government Commonwealth Postgraduate Scholarship, and grants from the Research Foundation for Information Technology at The University of Sydney.

DECLARATION

The work in this thesis is my own, except where otherwise referenced
or widely known facts are stated.

TABLE OF CONTENTS

Chapter 1

Introduction

Electronic computers have come into widespread use in the past thirty years due to their superior speed on tasks of data manipulation and simple control. However there is also a need for systems that complement electronic computers in areas where they are not well suited or where they are not able to match human performance. These areas include fast pattern recognition, human-computer interaction, language processing, and complex reasoning. Such systems could, if a community desires,

— relieve people from certain boring, stressful, remote or dangerous tasks,

— be used to assist people (especially the disabled) with their work and leisure while allowing privacy and enhanced independence, and

— aid productivity and creativity by providing more natural and intelligent computer interfaces.

Researchers have attempted to program computers to mimic human capabilities but have run into a number of problems.

First there is the bottleneck of acquiring and preparing data or knowledge for entry into a system. Rule based systems require extensive interrogation of experts and the integration of that knowledge into a rule database. Expert knowledge, as articulated, is usually incomplete, inaccurate and contradictory, necessitating extensive system refinement and testing. With probabilistic expert systems, experts must estimate the strength with which each antecedent of a proposition contributes to the "firing" of a rule and the strength with which the consequent is subsequently asserted. Probability estimation errors can compound and result in an incorrect output. Humans, on the other hand, are able to learn both by rule and example, through both teaching and self-discovery. A large number of examples can be internally synthesised into rules, either consciously or subconsciously.

Second, classification systems employing machine learning algorithms are notoriously brittle to noisy, highly variable, unexpected, or unseen input. Many machine learning algorithms have difficulties with highly structured input data. Hand-chosen input features and recognition structures are usually sub-optimal. Decision tree methods do not fully capture feature space and classification space adjacencies, which is necessary for good generalization performance.

Third, as the systems are scaled to handle larger and more complex problems, performance falls rapidly while the required number of rules explodes or learning algorithms get more complex.

The problem of accurate machine recognition of speech is a good illustration of these points. The creation of a large vocabulary robust system that transcribes conversational speech has been much harder than was expected when research began in the early 1960s. The problem is the enormous variability of the speech signal. People can easily recognise speech independent of variations in the transmission channel, pitch, accent, talking rate, rhythm, emotional tone, and stress, despite large changes in the speech waveform. Compounding this, at fast speech rates, the sounds of speech begin to overlap each other and pauses between words are eliminated. This variability can only be handled by a structured system that recognises progressively larger chunks of input. However, it is not clear what form the chunks should take, and so recognition units are hand-chosen based on incomplete experimental knowledge of the speech code. A mechanism of combining chunks to form larger chunks, that is tolerant of input variation, is also difficult to find. The result is that even the best current systems are not suitable for use as robust, general-purpose speech interfaces. The complexity and variability of the speech signal mean that a high-performance speech recognition system will most likely have to self-discover relevant input features, and self-construct sequence-recognizing state-machines from vast amounts of sample speech.

In order to overcome these problems when designing systems to match human capabilities, it was natural that the means by which the brain performs its tasks was investigated. Although many details of brain function remain shrouded, enough is known to extract some of the principles by which the brain achieves its speed and power. The creation of Artificial Neural Network (ANN) systems were pioneered by cybernetics researchers in the 1950s, yet widespread research did not begin till the mid 1980s following disappointment with progress in mainstream Artificial Intelligence research, and the discovery (and rediscovery) of powerful ANN architectures.

ANN models are inspired by the way large numbers of simple neurons, connected together in the brain, can collectively produce complex behaviour. A network can store information in the strength of connections between pairs of neurons and in the level of activation of each neuron. It learns to respond appropriately to its input by adjusting the connection strengths according to the current network state. Algorithmically simple, robust systems that can learn incrementally, discover relevant structures in the data through self-organization and which can integrate enormous amounts of information to solve a problem can be created with ANN techniques. Simple neural networks are beginning to outperform established methods.

Systems with large amounts of computing power and storage are, however, required to simulate very large neural networks capable of tackling complex control problems and real-time emulation of the human sensory, language and reasoning systems. Recent progress in the understanding of brain function, the development of powerful artificial neural architectures and advances in VLSI technology have made it feasible to construct low-cost and compact parallel computers for simulating large ANNs capable of attempting these complex tasks.

This thesis is concerned with the design of such systems. It proposes a set of new techniques for efficiently simulating sparsely connected ANNs in which only a small proportion neurons active at any one time — properties that are shared by the brain and some ANN models. A multicomputer design that is able to apply these techniques to simulate large networks with real-time (human-speed) response and learning rates (as well as efficiently simulate all common ANN architectures) is then described. The use of low-cost and high-speed processor and memory technology, and the simplicity of the inter-processor communications interface, allow very large networks to be simulated at modest cost. An important design goal is that the system must not sacrifice computational flexibility for speed as new information about the workings of the brain and new ANN architectures and learning algorithms are continually emerging.

The main contributions of the thesis are:

— an analysis of the important features of biological neural networks that need to be simulated,

— a review of hardware and software approaches to neural networks, and an evaluation of their abilities to simulate brain-like networks,

— the development of techniques for efficient simulation of brain-like neural networks, and

— the description of a multicomputer that is able to simulate large, brain-like neural networks in real-time and at low cost.

Outline of the Thesis

Chapter 2 describes the computational structures that we are interested in simulating; namely, biological neurons and their artificial models (both their input/output behaviour and adaptive capabilities), natural and artificial neural network architectures, and self-contained artificial sensory systems.

Chapter 3 reviews previous approaches to neural network simulation. Implementation technologies and system architectures for efficient simulation neural networks with various forms of connectivity are discussed. Hardware architectures are evaluated on their ability to meet the design goal of low-cost and real-time simulation of large neural networks. It concludes that existing systems are inadequate.

Chapter 4 introduces techniques for simulating very large, brain-like neural networks with the minimum possible computation and memory requirements. Methods of network update are presented that allow the influence of one neuron on others, through its outgoing connections, to be calculated only when the neuron is active. This saves a significant amount of computation when the neural network is only sparsely activated, as is the case in competitive networks. The simulation techniques presented also provide:

— the ability to move the synaptic parameters of dormant neurons to secondary storage,

— the means to efficiently simulate networks that are updated asynchronously or where the necessary neuron latency or input integration time-step is less than the minimum time between changes in the output of a neuron,

— efficient implementation of brain-like weight update rules, and

— the reduction in inter-processor communications requirements in parallel implementations.

Methods for storing network parameters compactly in memory are then discussed.

Chapter 5 outlines the design of a multicomputer for low-cost, real-time simulation of large ANNs. The application-specific design results in a cost-performance benchmark far ahead of general-purpose multiprocessors. The design is able to efficiently implement the techniques of Chapter 4. Signal processing and analog interface cards are described that can be integrated to make the system suitable for research on speech recognition and understanding.

Chapter 6 details how various ANN architectures would be simulated on the system.

Finally, Chapter 7 summarises the conclusions of the thesis and considers the future of ANNs in pattern recognition and artificial intelligence.

Chapter 2

Structures to be Simulated

To design a system capable of emulating the neural networks of the human brain (among others), it is vital to discover the important aspects of the brain that need to be modelled. For efficient simulation, structural and functional details that do not contribute to the power of the model must be factored out, and hardware support for important information processing mechanisms should be provided.

The contributions of this chapter include:

— an analysis of the important features of biological neurons that need to be simulated, including a clarification of Holler's measure of required network update rates for real-time simulation,

— a review of neural network learning rules, both biological and artificial,

— consideration as to the required accuracy and dynamic range of computations and storage in neural network simulations,

— a detailed discussion of likely functional structures in the cerebral cortex of the brain,

— definitions of important aspects of artificial neural network simulations, and

— justification of the power of integrated artificial sensory systems.

2.1 The Neuron

A Neural network is a massively parallel computational system. Each neuron performs a relatively simple input to output transformation yet, when connected together in large networks, extremely powerful computational structures can result. This section reviews the various input/output and adaptive capabilities of biological neurons and their artificial models.

2.1.1 Models of Neuron Behaviour

The major features of a biological neuron are shown in Fig. 1.

Neurons can, and almost always do, have many inputs, but have only one output. Axon pulses arriving at a synapse trigger (within 0.5 *ms*) a reduction in the membrane resistance of the area of dendrite on the other side of the synapse. During the few milliseconds that the resistance remains low, current, in the form of ions, is able to move in or out of the cell under osmotic pressure. The current flow imposes a voltage change in the adjacent dendrite that is quickly propagated to the cell body. Here, inputs of all active synapses are integrated to produce a change in voltage at the head of the axon. The period of integration (usually about 10-20 *ms*) is determined by the capacitance and resistance of the cell's membrane, which is a function of the type, shape and size of the neuron. The time taken for the summed voltage change to decay allows for integration of temporally spaced inputs.

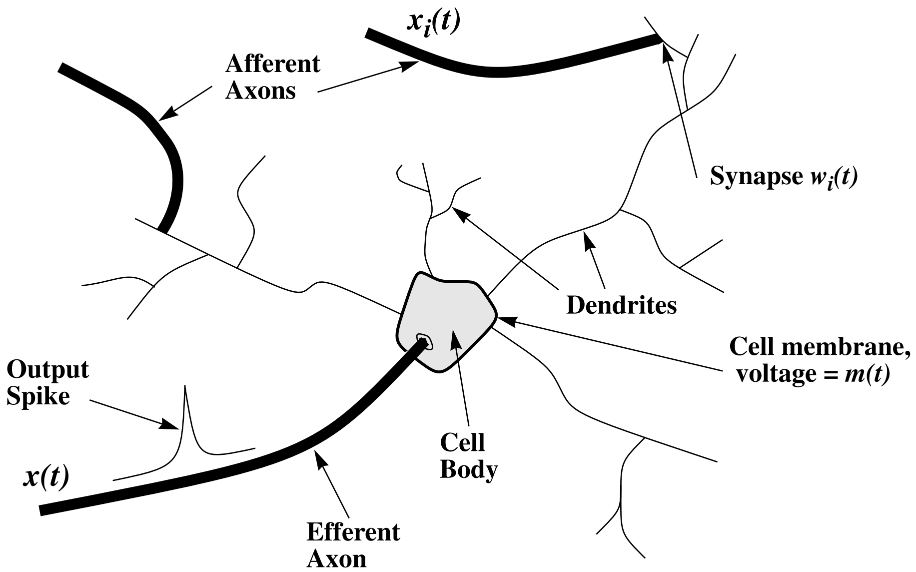


FIGURE 1. Main features of a biological neuron.

If the voltage at the head of the axon becomes greater than a threshold value (Θ), the cell will fire, sending an output pulse down the axon. The pulse travels down the axon at speeds between 0.1 and 120 metres per second. The propagation speed is a function of the diameter of the axon (wider is faster) and whether the axon is sheathed with a waxy material that greatly reduces propagation losses and increases propagation speed. Axons can therefore be used as delay lines in the nervous system (Jones and Van der Poel 1990).

The time it takes for a neuron to fire following synchronous active input of sufficient strength (or the time-to-fire following a single active input for a neuron that is near threshold) is called the **latency** of the neuron. This determines the speed with which an input can pass through a sequence of neurons. The latency of brain-neurons is about 2 *ms* and is dependent on the strength of the input.

The firing of the cell resets the cell voltage at least part of the way towards (and, for some neurons, well below) the resting membrane voltage, from which the inputs can integrate the cell back to threshold again, sending another pulse down the axon. The cycle of reset-integrate repeats over the duration of application of the input, producing a firing rate which is a function of the current total input strength during that period. This enables us to view a neuron's output as either temporally spaced individual spikes or an analog value representing a firing rate, and a neuron as a hybrid analog-digital or purely analog device. The firing rate of most brain neurons is limited to approximately 100-200 *Hz*. For example, in the auditory periphery, the spike representation is vital for computation of the pitch and direction of a sound based on the precise temporal coincidence of inputs. The pulse-rate modulation output coding method also means that it takes time for an analog output to be precisely signalled and integrated, which is likely to be a factor in the categorical perception of short auditory stimuli. Probabilistic synaptic transmission and thermal noise mean that the membrane voltage can fluctuate stochastically, which can lead to a non-zero output firing rate even when no input is applied.

The axon can divide many times and contact the dendrites of many other cells, forming synapses. Thus the output pulse forms part of the input of the cells it contacts. Receptor and effector cells link the nervous system with the environment by forming synapses with peripheral neurons. Dendrites are usually short, branching close to the neuron body. Axons on the other hand can arborize over a must greater extent, seeking out dendrites of distant neurons.

Many active inputs are usually required to bring the cell to the firing threshold. Ables (1982) has calculated that synchronous pre-synaptic firing is 10 times more efficient at activating a neuron than integration of asynchronous inputs. This forms the basis of a theory of selective attention and coherent feature binding through synchronized oscillations of neuron firing (see Barinaga, 1990). The phase of neuron firing determines if two neurons interact or not.

Neurons can also receive other non-synaptic inputs that affect the way the cell operates. For example, changing the firing threshold or membrane resistance. Such inputs are called neuromodulators (Foote and Morrison, 1987).

Each input pulse does not affect a neuron equally. First, an input can be excitatory or inhibitory, depending on whether it adds to or subtracts from the integrated voltage, and second, the strength and time course of the effect of an input on a neuron depends on its location and the type and current state of the corresponding synapse. For example, synapses furthest from the cell body cause smaller voltage changes at the cell body than more proximal synapses, yet the voltage change lasts for a longer time. Continuous application of such an input can have the effect of a firing threshold change. The same should be true of a class of inhibitory inputs[1] that invoke long-lasting current flows. Altogether, the measure of transmissibility of a neuron's output across a synapse can be encapsulated in a single positive or negative synaptic weight parameter, *w*. More generally, the synaptic weight can take

1. Type $GABA_B$.

the form an impulse response function, $h(t)$ which can describe the initial peak and decay of current through a synapse in response to an input spike, or the temporally smeared (due to dendritic capacitance) manifestation of this current at the cell body.

In summary, a neuron can be modelled as a leaky integrator of its synaptic input:

$$\tau \frac{dm(t)}{dt} = -m(t) + input(t) \\ input(t) = \sum_i w_i \cdot x_i(t) = \sum_i (h_i \star x_i)(t) \quad (1)$$

Spiking neuron model : if $m(t) > \Theta$ then $m(t) \leftarrow m(t) - RESET$, and output a spike.
Average firing rate model : output = $f(m(t) - \Theta)$.

$m(t)$ (≥ 0) is the membrane voltage at the cell body with time-constant τ and $x_i(t)$ (≥ 0) is the instantaneous or average firing rate of the axonal input at time t, at the synapse with weight w_i or $h_i(\cdot)$. $\star$ denotes convolution. f is the nonlinear relation between average membrane voltage and firing rate for the particular neuron type.

Many ANN architectures use a more abstract model of a neuron where time is discretised, synaptic inputs change much more slowly than the membrane time constant, and the membrane voltage is fully reset at the beginning of each step.The input/output behaviour of a neuron can therefore be simulated as follows:

$$x(t+1) = f\left(\sum_i w_i \cdot x_i(t) - \Theta\right) \quad (2)$$

where $0 \leq x(t) \leq 1$ is the output strength of the neuron at time t.

Time discretisation is of course required for digital simulation of neuron models. Digital simulation of the model of Equation 1 requires that the simulation time-step be a small enough fraction of the membrane and synaptic time-constants that the input can be accurately integrated. The time-step should also be a whole fraction of the required neuron latency, and also small enough that spike arrival times are sufficiently precise (if the spiking neuron model is used). This usually means that a neuron's state needs to be updated at a much faster rate than the maximum rate at which the neuron's output changes (the maximum firing rate) — say every 1 *ms* against every 10 *ms*. The neuron model of Equation 2, by doing away with the spike representation and input integration, requires only that the update rate match the maximum desired rate of output signalling and be less than the maximum tolerable neuron latency.

Real-time simulation of a network of neurons requires that simulation time equal actual time. Therefore, to match the speed of biological neurons, each synapse must be updated at a rate which is at most equal to the maximum rate at which outputs can change. That is, in the worst case of all outputs changing at their maximum rate, the simulation must run at approximately 100 *CUPSPW* (Connection updates per weight per second) (Holler, 1991). This does not mean that the simulation time-step can be 10 *ms*. Neuron outputs (and membrane voltages) will need to be updated at shorter intervals for low neuron latency (and accurate input integration and spike timing). Because there are many more synapses than neurons, at simulation time-steps that are not too small, synaptic update dominates the simulation computation. Therefore the *CUPSPW* value is a good measure of the computational load. The time-step can be shortened to reduce latency without incurring excessive computation if the simulation restricts synaptic calculations each time-step to those synaptic inputs (neuron outputs) that have changed since the last update, rather than calculate the effect of every synapse at each time-step. Chapter 4 will discuss methods of efficiently simulating neurons where the simulation update rate is greater than the maximum rate at which an output can change.

It should be noted that the discussion of neuron models presented above aims only to highlight the general features of a neuron to be simulated. Real biological neurons have voltage-sensitive ion channels in synapses and dendrites, and some abstract neural models combine their inputs in ways other than a weighted sum (Kohonen, 1988; Mel and Koch, 1990), or have internal neuron state variables. Core simulation computations remain additions, subtractions, multiplications, and exponential decays.

2.1.2 Weight Modification Rules

A neuron can adapt to its input by modifying its synaptic weights. For most current ANN architectures and, it appears, biological neurons, changes in the weights are dependent on the degree of correlation between the level of activity of the incoming axon and the activity of the post-synaptic neuron or dendritic branch over short periods of time — the so-called Hebbian class of learning laws. It is the local nature of Hebbian laws which allows discussion of weight modification in the context of single neurons rather than complete neural networks. Locality is sometimes broken by modulating the learning with a global reinforcement signal. Thus a general form of Hebbian learning is:

$$\Delta w(t) = F\left[PRE(\cdot), POST(\cdot), t\right] \cdot R(t) - d \tag{3}$$

where $PRE(\cdot)$ and $POST(\cdot)$ are the time courses of pre-synaptic and post-synaptic activity and F is an arbitrary functional. $R(t)$ is the reinforcement signal and d is a weight decay term.

Four common instances of Hebbian laws will be discussed.

Simple Hebbian Synapses

Hebb's original idea was a synaptic weight increase following correlations between the pre-synaptic and post-synaptic firing strengths:

$$\Delta w_{ij}(t) = k \cdot x_j(t) \cdot x_i(t) \tag{4}$$

with k a constant and $x_j(t)$ and $x_i(t)$ the pre-synaptic and post-synaptic activities respectively.

The problem with a straight application of this learning rule is that weights can only increase, eventually leading to saturation of activation and a loss of important weight relativities. The learning law can be made usable in competitive networks (only the most responsive neurons to an input get to change their synaptic weights), or by adding a weight decay term (Chernjovsky and Moody, 1990).

An obvious deficiency of Equation 4 is that there is no differentiation between the order of pre- and post- synaptic firing. Such differentiation may be necessary for learning and recall of input strings.

The Delta Rule

This learning rule originated in the psychological literature as a model of conditioning, and is used in many ANN architectures. A synaptic weight is changed in proportion to both the pre-synaptic activity and the difference between an “expected” post-synaptic activity and the actual post-synaptic activity:

$$\Delta w_{ij}(t) = k \cdot x_j(t-1) \cdot [x_i^{\text{EXPECTED}}(t) - x_i(t)] . \tag{5}$$

In so-called “supervised” neural networks, which learn mappings between an input space and an output space, the expected output is taken to be the desired neuron output for the current input. Weight modification aims to reduce the neuron output error by distributing blame for the error among neuron inputs, according to their activation. The neuron error is usually calculated non-locally, requiring a more complicated neuron model that can accept special error inputs.

One alternative for the expected output is $min(x_i(t–1)+n, 1)$ where n is a small constant (Tesauro, 1990). That is, a post-synaptic activation increase, paired with pre-synaptic activity, leads to a weight increase, while a decrease in post-synaptic activation paired with pre-synaptic activity leads to a weight decrease. This rule is able to explain all forms of conditioned response.

Another alternative for the expected output is the average value of $x_i(t)$. This was used by Sutton and Barto (1981) and Bear *et.al.* (1987), and will be discussed in the context of biological learning laws.

Note that if $x_j(t–1) = 0$, no weight modification takes place. Pre-synaptic activity is the trigger for learning which is important for the efficient simulation method to be discussed in Chapter 4.

Competitive Learning

With competitive learning (Kohonen, 1988), significant post-synaptic activity is the trigger for learning. Only those neurons that win a competition for activation through mutually excitatory and inhibitory connections are able to modify their weights. The difference between competitive learning and simple Hebbian learning is that the direction of the weight change is dependent on the relative sizes of the current weight and the pre-synaptic activity:

$$\Delta w_{ij}(t) = k \cdot [x_j(t) - w_{ij}(t)] \cdot x_i(t). \tag{6}$$

The set of weights on a neuron move to match an exemplar of the set of input vectors that cause the neuron to win the competition. Topological maps of the input space can emerge. However, the memory of an input can get washed away if an input is silent for too long. Also, the rule also does not allow for sharpening of responses by weakening synapses on weakly activated neurons.

Biological Synapses

Vindicating Hebb, synaptic weights have been observed to change in the brain as a function of the correlation between pre- and post-synaptic activity at a synapse. The time scales over which the activity is sampled appears to be very short and the order of pre- and post-synaptic activity is important (Hashemzadeh-Gargari *et al.*, 1991). Agents that enhance synchronization of neural activity are known aid memory formation.

Weights have been observed to increase up to a factor of three with simultaneous pre-synaptic activity and post-synaptic voltage increase, and decrease with either low post-synaptic voltage coupled with pre-synaptic activity or with high post-synaptic voltage and an absence of pre-synaptic activity (see Tsumoto, 1990, for a review).

These forms of weight modification have been observed as potentiation or depression of synaptic strength that is expressed in fractions of a second to minutes after stimulation, and lasting for periods of at least a month. Long term memory storage requires permanent modification which is hypothesised to be triggered by the presence of chemical modulators. Short-term fatigue of continually activated synapses is well-known and short-term synaptic facilitation has been observed (Hirsch and Crepel 1990; Schultz and Johnston, 1991).

In animal neocortex (the part of the brain that controls higher functions), weight change has been hypothesised (Bear *et al.*, 1987) and been observed (Artola *et al.* 1990) to be dependent on the post-synaptic voltage level when the pre-synaptic axon is active, as shown in Fig 2.

The membrane voltage at each synapse will be due to combination of the current from the synapse itself and current that has spread from other synapses and the cell body, and so will depend on the average recent level of cell activation.

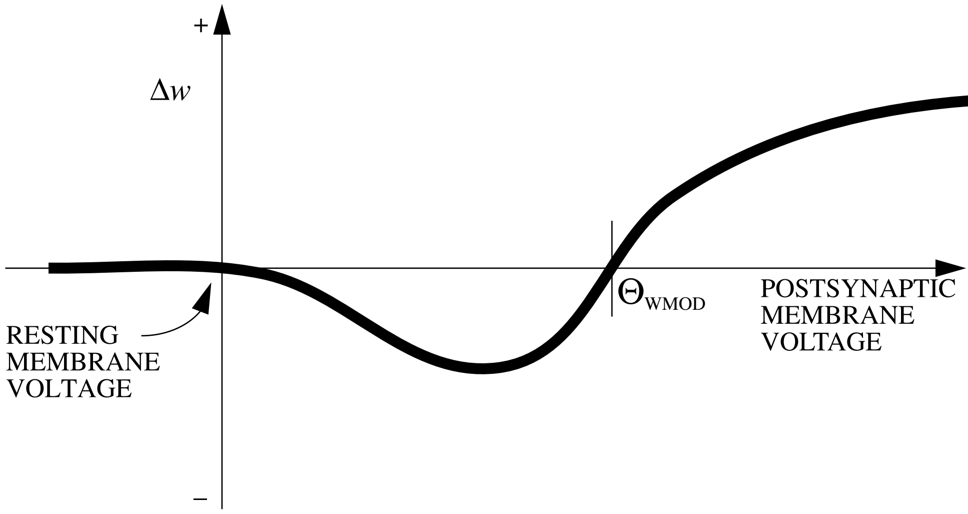


FIGURE 2. The BCM weight modification law.

For acceptable theoretical network behaviour, the threshold (Θ_{WMOD}) that determines whether the weight increase or decreases must not be fixed, but vary according to the long term average of post-synaptic voltage (Bienenstock, Cooper and Munro, 1982). This could possibly be implemented by an automatic gain control (AGC) loop mediated by a subset of cells in layer 5 of the neocortex (James and Hoang, submitted).

The depression of synaptic weights at low post-synaptic voltages allows sharpening of the response to active afferents by punishing active synapses on those neurons that are weakest in a competitive battle for activation. The absence of weight modification at very low voltages protects memories from being erased on neurons that receive very little input.

Heterosynaptic depression is the name given to the reduction of the ability of an input to drive a neuron after paired high post-synaptic voltage and a lack of pre-synaptic activity. It has been observed as a reduction of field potentials in the hippocampus and visual cortex. It is not known how much this is due to actual synaptic weight reduction and how much to increased inhibition of the post-synaptic neuron. Increased inhibition could be the result of the same AGC mechanism discussed above. Evidence that weight modification in the neocortex does require pre-synaptic activity comes from experiments on the visual cortex where the degree of synaptic depression was greatly reduced when spontaneous pre-synaptic activity was blocked (Chapman *et al.*, 1986).

It is thought that only excitatory synapses formed with spine-like post-synaptic dendritic projections are capable of other than short-term weight change. Spines host neurotransmitter receptors known to be involved in synaptic weight modification, and have a shape that allows chemical compartmentalization. Inhibitory synaptic weights are thought to be fixed. The strength of an inhibitory projection can still be learnt in two ways. One, through the depression of excitatory synapses in a joint excitatory and inhibitory projection, or two, potentiation of synapses on a neuron that goes on to excite an inhibitory neuron in the projection area. There is no reason why artificial systems need follow this restriction.

2.1.3 Required precision, accuracy and dynamic range of computations and parameter storage/communication for simulating neuron behaviour

What accuracy, precision and dynamic range are required for computation and parameter storage/communication in simulations of neuron behaviour? Faster and more compact simulation hardware may be possible if the range and tolerance of computation and storage is restricted, yet the power of the neurons to learn and respond appropriately should not be compromised. The adaptive nature of neurons often allows for a relaxation of accuracy requirements if the computation and storage are of sufficient precision. This is important for simulations on analog hardware.

Neuron Outputs (Synaptic Inputs)

Many ANN architectures allow inputs to the network to code for a continuous stimulus attribute such as length or temperature. Any limitation on the precision or dynamic range of the representation of such inputs obviously limits the accuracy with which the network can use the attribute in any classification. Outputs of neurons within the network should have enough precision and dynamic range to accurately code a "level of belief" in the presence of the micro-feature coded by the neuron. It is possible to reduce the required precision and dynamic range of network inputs to that of neuron outputs, while at the same time maintaining an accurate representation of input space, if they are re-coded or automatic gain control feedback loops are employed. Possible input codes include coarse coding (Hinton *et al.*, 1986), range codes, or binary codes, where an external input value is represented by the activities of many axons. Therefore, there can often be a tradeoff between numbers of input axons, or active input axons, and the necessary computational and storage precision. Neuron output strengths can also code for the order of an input sequence (Grossberg, 1985), necessitating at least three or four bits of output precision in these types of networks.

Spiking neurons have outputs of one bit precision only, unless the spike shape is significant. For these neurons, accurate output representation requires accurate representations of both spike timing and the membrane voltage of the neurons post-synaptic to the output.

Synaptic Weights

In ANN models, the precision and dynamic range of the synaptic weights determine the accuracy with which each neuron codes a partition of input space. If the precision of the weights is too low to encode a salient partition, more than one neuron will be required to do the same job and the network size will be larger, and its generalization performance poorer, than the optimum. For ANNs that learn via gradient descent on error surfaces, weight precisions need to be higher during the learning phase than the performance phase to ensure convergence. Competitive learning models require the weights to be of the same order of precision as the external inputs so that the average strengths of the input vector can be encapsulated in the weight vector.

ANN models that aim to emulate the cerebral cortex certainly need weights with greater than one bit precision. Models that have automatic gain control neurons that regulate the effective output threshold of other neurons or synaptic inputs that are strong enough to veto all excitatory inputs to a particular neuron require weights with large dynamic ranges. Specifically, a modulating input with full power of veto over n synapses, each with k bit weights, will require a weight of $k+log_2 n$ bits. The graded perceptual weighting of various auditory input features that contribute to a speech percept which vary between individuals — known as trading relations (Pisoni and Luce, 1987) — demonstrate that the synaptic weight producing the phenomenon must be greater than 1 bit. In fact, there is evidence (Mason *et al.*, 1991) that the range of effective synaptic weights of biological neurons is at least six bits.

Synaptic Input Accumulation

For a neuron with n synapses, synaptic weights of w bits of precision, and synaptic inputs of i bits of precision, synaptic inputs must be accumulated to $log_2 n+w+i$ bits for no loss of precision. The required precision of input accumulation can often be less than this for two reasons,

1. the saturation of output firing rates above a certain input level allows the maximum representable input level to be restricted to this level, and
2. many synapses will have small synaptic outputs that can be ignored; synapses with small weights can be removed from the input calculation until learning makes the weight large enough.

In summary, the necessary and sufficient precision with which neuron outputs, synaptic weights, and accumulated synaptic inputs need be represented is dependent on the neuron model and the complexity of the input. With too few bits in the representation and the network can not function effectively or efficiently; with too many bits and the rate of simulation is slowed down and storage requirements go up for little or no performance gain. The necessary storage and computational precision my be able to be reduced if non-uniform quantization is employed. For example, logarithmic quantization of synaptic weights. Determination of the optimal precision for a given application, input coding scheme and network architecture, will require some experimentation.

The beauty of digital simulation is that precisions higher and lower than the precision of the ALU can be emulated in software. Processor support for bit-field extraction should be provided to handle lower precision calculations. For one-bit neural outputs and weights, an instruction that performs a bit-wise logical AND, and then computes the number of 1's in the result is useful (Hammerstrom, 1990). As a compromise, the precision of the ALU should be large enough so that most neural models can be simulated without resort to software precision enhancement, but not of so high a precision that the instruction rate is held back by the speed of the ALU and multiplier or that extraction of lower precision numbers is always required. The neural network simulation system to be described in Chapter 5 uses a processor with a 12×12 bit multiplier and 29-bit accumulator. Precision extension should be unnecessary for networks other than those of the error minimization type mentioned above.

2.2 Neural Networks

This section outlines how neurons fit together in networks, both in the brain and in artificial neural architectures.

2.2.1 Natural Neural Networks

The anatomy (structure) and physiology (functional mechanisms) of the human brain are worthy of investigation when designing artificial neural systems. The cerebral cortex of the brain, which processes sensory inputs, determines motor responses and produces the higher faculties of language and reasoning, is a good place to look for powerful architectures for general purpose pattern recognition. The neural networks of the brain stem and thalamus, which preprocess sensory stimuli for input to the cortex, have evolved to extract perceptually important sensory features that cortex, with its generic architecture, can not. For example, the sounds of language have likely evolved to take advantage of (or co-evolved with) brain stem acoustic pattern detectors. The cortex alone would not be able to keep up with fast speech. Some important features of speech perception such as rate-dependent processing (Gordon, 1988), contrast effects (Johnson, 1990) and interpretation of the auditory "scene" (Bregman, 1990) are likely to depend on particular mechanisms of cortical processing. ANN architectures which lack these mechanisms are likely to require larger networks or perform more poorly than those that include them. Even the uniquely biological phenomena of sleep may give clues to the regulation of learning in ANNs.

The cerebral cortex, contains some 10^{11} neurons. The neurons are arranged in layers in a convoluted sheet. Each neuron forms synapses with up to 10^5 other neurons — a total of approximately 10^{15} synapses when the average number of synapses per neuron is considered. There are dozens of neuron types, each with its own synaptic and integrative properties, and each form synapses with a particular set of other neuron types, over a particular spatial extent. The rough connectivity is genetically programmed and is honed through activity dependent pruning of connections during development.

While it seems daunting to consider simulating a network of such size, there are many regularities in the connectivity between neurons which can be exploited to reduce both the amount of computation and storage required, making it feasible to simulate a functionally useful portion.

(i) There is evidence that neurons in the cerebral cortex form tight anatomical and functional groupings of 100-200 cells, called minicolumns (Mountcastle, 1978) (but see Swindale, 1990). The cells reside in a 30-50 μm cylinder, perpendicular to the cortical surface, passing through all cortical layers. It is thought that all cells in a minicolumn are derived from a single cell during development.

A large number of the connections to a particular neuron in a minicolumn come from other cells in the minicolumn. This is especially true in the left (dominant) hemisphere of the cerebral cortex which is believed to subserve practised tasks — especially tasks that require processing and discrimination of temporal input streams, such as speech processing. The columns in the right hemisphere seem to be looser with more inter-column connections, suggesting more of an associative memory structure (Seldon, 1985). This structure is particularly suited to tasks such as face recognition.

Minicolumns can act as an integrative units for input, reducing the effect of cell death, probabilistic synaptic transmission or firing fatigue on function. Positive and negative feedback within the minicolumn can lead to hysteresis and oscillations with various time constants which could subserve short-term-memory, automatic gain control (Hagenzieker *et al.* 1990) and selective attention (Barinaga, 1990; Sporns *et al.*, 1991) functions. The "magnocellular" areas of the cerebral cortex are likely locations of short term memory storage, as the long input integration times of the large cell sizes in these areas can best latch intra-column positive feedback.

Minicolumns send excitatory and inhibitory links to other minicolumns, both close-by and more distant, leading to cooperation and competition for activation and a limit on total local activity. A subset of the inter-minicolumn links take the form of **lateral-inhibition**, with local excitatory links, and more distant inhibitory links.

Groups of minicolumns are likely to self-organize to respond to selected simultaneous input from overlapping projections to their location. The projections to a minicolumn come from other minicolumns in the region, other areas of cortex, and from cortex surrounding the hippocampus which is likely to be involved in conscious recall of the information that the minicolumn stores. Often the self-organized structure consists of groups of minicolumns, 500 μm in diameter, with common input — a "macrocolumn" — flanked or surrounded by macrocolumns receiving, or responding to, very different input. This is likely a consequence of lateral inhibition. For example, the functional orientation columns of visual cortex correspond to the anatomical minicolumns. Macrocolumns that respond best to oriented bars of light at a particular point of visual space, presented to the left eye, are surrounded by macrocolumns maximally sensitive to right eye stimulation and macrocolumns not sensitive to orientation.

The modular nature of a minicolumn means that each layer in a minicolumn, or a minicolumn itself, can possibly be treated as a complex neuron (perhaps one with multiple input and output types). Also, multiple synapses of the same type between pairs of minicolumns can be simulated as a single synapse with stronger initial weight and speed of weight change. The result is a reduction, by at least an order of magnitude, in the number of units and synapses that need to be simulated.

(ii) On a larger scale, neurons are grouped into modules. A given module has a common genetically programmed neural architecture which is optimized for its function. Although a large number of connections form between neurons in the same module, as discussed above, a module also communicates with a specific set of 10-30 other modules (Mountcastle, 1978). A significant fraction of these connections are to adjacent and nearby modules, but there are long projections as well. This pattern of connections is illustrated in Fig. 3, which shows the current state of knowledge of the communications between modules in the monkey visual cortex. Thus, much neural communication is local, and the more distant communication is to a small fixed set of targets.

The pattern of module-to-module projections has a random component yet is far from totally random. A topographic order is often imposed on projections, either through axon contact guidance during development or later activity-dependent connection pruning. Also, projections of a single axon between modules usually terminate in one or more dense patches, each approximately the size of a macrocolumn, spread across a module.

All these regularities in the connectivity can be exploited to

1. reduce the total bandwidth and increase the predictability of inter-processor communication loads in a parallel simulation system, and
2. reduce the number of bits required to encode the network connectivity.

Sparse connectivity and competition for activation can lead to sparse, distributed activation within a given module. Such representations aid the generalization abilities of the network and are required for reliable associative memory storage (Hecht-Nielson, 1990; Kohonen, 1988). Another consequence is that, at any given time, one need only calculate the axonal influence of the small number of firing neurons when simulating such neural architectures. This will be exploited in the methods of Chapter 4.

The brain can recognise complex objects in a couple of hundred milliseconds, even though the stimulus must be processed through chains of many modules. Thus, in any real-

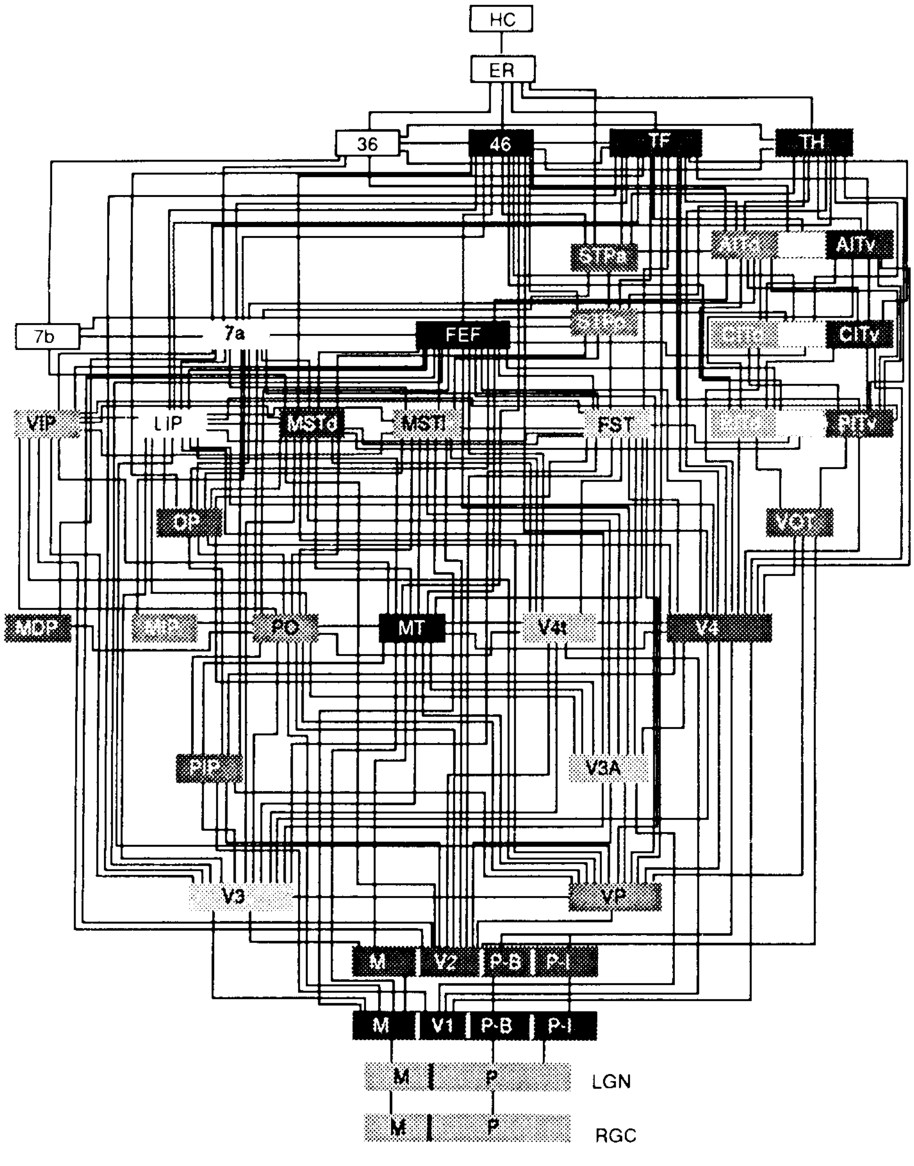


FIGURE 3.

Pattern of connectivity between modules in the monkey visual cortex.

From van Essen *et al.*, 1991.

time simulation of a cortex-like ANN architecture, it is vital that the model neurons match the latency of brain neurons.

2.2.2 Artificial Neural Networks

While the brain is a powerful neural computational system, one need not slavishly follow its exact architecture and functionality when designing artificial neural systems. Biological neural nets are constrained by the need to self-construct, and by the living chemical substrate they have to work with.

A large variety of Artificial Neural Network models have been proposed, and applied in many applications. These models are usually defined by the input/output behaviour of the local computing units (“neurons”), the method of network update, the pattern of connectivity between neurons via weighted links, and a set of learning laws to govern the ways the weights are modified. A simulator for neural network research should be able to efficiently simulate a wide variety network models with differences in each of these defining characteristics. Models of neurons and synaptic learning laws have already been discussed in previous sections. The other two attributes will be discussed here.

The neurons in a network can be updated synchronously, asynchronously or in continuous-time. Continuous-time updates can only be executed on analog hardware, especially neuron models with analog outputs. Synchronous update refers to simulations where time is discretised and the outputs of either all or a subset of neurons in the network are simultaneously updated each time-step, based on the neuron outputs of the previous time-step. Most ANN models use this update method. Following the neuron model of Equation 2, update of the network can be seen to centre on the calculation of a matrix-vector product:

$$\underset{\sim}{x}(t+1) = f(W\underset{\sim}{x}(t) - \underset{\sim}{\Theta}) \tag{7}$$

where W is a matrix of synaptic weights. Many ANN architectures specify a split-synchronous update method where only one layer of neurons is updated at every time step.

Synchronous update of neurons with analog outputs may not be suitable where neurons are connected in cycles, especially if all outputs are allowed to change at each time-step. With asynchronous update, update equations such as Equation 2 are calculated for only one randomly selected neuron at each time-step, or the simulation is driven by the timing of network inputs. Combinations of synchronous and asynchronous update are possible.

The type of connectivity between neurons falls into two classes:

1. A neural network architecture will be called *completely connected* if, at each time-step during the execution of that architecture, any axon that forms a synapse with a neuron that is scheduled to be updated at that time-step also forms a synapse with every other neuron that is scheduled to be updated at the same simulation time-step. This definition includes architectures where every neuron is connected to every other (including possibly itself) and also architectures such as that shown in Fig. 4 if the second and third layers are updated synchronously at alternate simulation steps. The first layer of neurons work simply as a fan-out units, distributing each input to all neurons in the next layer. Other connection topologies add connections between neurons in a layer or feedback connections to previous layers. By the definition given, any network updated asynchronously (only one neuron updated per time-step) will be completely connected. Efficient methods are available for the simulation of completely connected networks. These will be summarised in Section 3.3.1.

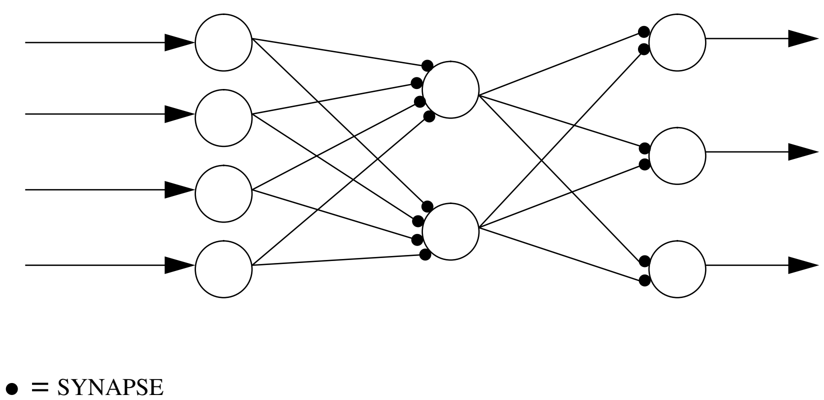


FIGURE 4.

A three layer feedforward neural network with complete connectivity between layers.

2. A neural network architecture will be called *sparsely connected* if it is not completely connected. This includes various brain-like connection schemes including columinar, patchy, and lateral-inhibition forms.

The *effective connectivity* of a network will be defined as the connections between neurons that have high enough synaptic weight to have a useful effect on the target neuron. This definition removes connections that are present but not worth including in the synaptic computations until the weight becomes larger through learning. Thus even completely connected networks can have sparse effective connectivity. The computational efficiency of a neural network simulation can be improved if the effects of non-effective synapses do not have to be calculated.

ANN architectures can also have a constructive or destructive learning algorithm that adds or deletes connections as learning proceeds.

The most popular ANN algorithm, by far, has been Backpropagation (Rumelhart *et al.*, 1986). Backpropagation is a layered, feedforward ANN model that learns using the delta rule to minimise output errors of all input vectors in the data it is trained on. Its popularity is based on its simplicity, analysability and ability to learn arbitrary input-output mappings. Its failings include a need to present each training vector many times until the output errors converge to zero, the tendency to get stuck in local error minima, and an incomplete input model at zero output error. Many simulation systems have been designed to efficiently simulate backpropagation networks. There is a need for an efficient simulator of networks with brain-like connectivity, learning rules, and distributions of neural activity.

2.3 Sensory Systems

To create a self-contained systems for simulating human sensory processing capabilities, modules that simulate input receptor and output effector functions must be integrated with the modules that simulate the neural net. Our interest lies in speech processing, so cards for signal processing and audio I/O have been designed for inclusion in the neural network simulation system to be described in Chapter 5.

Much of the power of human information processing stems from the ability to code the sensory and internal contexts of a event. Incomplete contextual coding manifests itself as “noisy” input features that can be eliminated by restricting the size of the input model. However, it is not usually clear to what size the model should be restricted, leading to a number of heuristics. Real-time simulation of large neural networks with a range of artificial sensors should be able to encode enough context to duplicate the power of its biological counterpart.

Chapter 3

Hardware Architectures and Algorithms for Neural Network Simulation

This chapter reviews current hardware architectures and algorithms for neural network simulation. The chapter explains why previous approaches are not cost-effective for use in large-scale real-time artificial neural systems, providing the motivation for the development of a special-purpose digital simulation system able to do the job.

Also, in Section 3.3.2, a method for reducing off-chip bandwidths in neural network simulations is outlined.

3.1 Definitions and Simulation Targets

Artificial Neural Network architectures and models of biological networks can be executed on analog or digital hardware. The analog pedigree of ANN neurons makes the use of analog computation more attractive than it has been for other algorithms. The hardware that executes a particular neural network architecture is known as an **implementation** of that architecture if each local computational unit in the architecture (synapses or neurons) is realized as a dedicated storage and processing unit (digital or analog) with the specified input/output behaviour, and if each axonal branch has a dedicated communications path. Any other realization of the architecture is known as a **simulation** of that architecture. For example, digital hardware can only approximate the input/output behaviour of analog (continuous-time) neuron models, and so digital hardware could only simulate, rather than implement, this architecture. The word simulation will be used for both implementation and simulation unless indicated otherwise.

Fewer processor or communications channels are required if the computation of more than one synapse or neuron is time-multiplexed on a single processing unit, or if a single communications link carries the output of more than one axon. This simulation technique is known as **virtualization.** A virtual simulation will run more slowly than a full implementation, yet can be more cost-effective if the slower speed is sufficient.

A neural network simulation system will be designed to meet a set of performance targets including:

— the range of network architectures (especially the forms of connectivity between neurons) to be efficiently and effectively simulated,

— the maximum number of neurons and synapses to be simulated, and

— the required input throughput.

The input throughput can be designed for real-time processing and learning (either at human or sensor speed) or be designed for a maximum speed of off-line data processing.

3.2 Implementation Technologies: Analog (Electronic and Optical) and Digital

The optimal implementation technology for a given performance target will depend on the relative cost and capabilities of computation, storage, communication and control in various technologies. A simulation system may employ any combination of technologies to implement each of these four simulation functions. The interfaces required between various technologies work against hybrid systems, yet many hybrid simulation systems have been constructed.

Computation

Electronic analog implementations of artificial neural networks are attractive for their ability to simulate in continuous-time and directly mimic aspects of the behaviour of their biological equivalents such as input integration, membrane voltage decay, resistive synapses, current summing, noise generation, and nonlinear response properties. Multipliers for synaptic calculations can be implemented using Ohm's law or using compact Gilbert multipliers. Adders can be implemented using Kirchoff's current law and I-V converters. Cheap CMOS technology can implement neurons with I/O bandwidths in *MHz* that draw very little power.

The compactness and low noise generation of computational components in analog VLSI makes feasible the integration of computation and storage to create fully implemented systems. The size of adaptable synapses implemented in analog VLSI can, however, require a chip area many times that required for synaptic weight storage — although some compact and elegant structures have been designed (Cauwenberghs *et al.*, 1991). Also, the power requirements of a separate computing unit for each synapse can begin to add up. Any move to virtualization to reduce the number of synaptic update units separates computation and storage, necessitating communication of the weight to the update unit, and requires the introduction of digital control sequencing. If the required network throughput allows a high level of virtualization, the size of parallel digital multipliers and the requirement for sequential summing become less of a burden, and the algorithmic flexibility of digital simulation can be taken advantage of. Analog VLSI is therefore best for simulating pre-trained single-architecture networks at high throughput rates.

The accuracy of analog multipliers and adders is sufficient for most neural architectures and, as noted in Section 2.1.3, the neuron can often adapt to correct any inaccuracies. There is also a clever technique for creating $n \times m$ bit synapses from m, $n \times 1$ bit synapses (Graf *et al.*, 1991).

Analog hardware can also implement relaxation networks that quickly solve combinatorial optimization problems, efficiently implement competition between neurons via current limited buses, and interface directly to sensors without an intervening digitization stage, allowing the creation of one-chip systems with the use of analog VLSI.

Optical technology can implement multiplication with spatial light modulators and adders by superposition. Greater than one-quadrant multiplication is, however, very difficult. Throughput is very fast, but the technology is currently too immature to simulate many ANN architectures.

Storage

The large memory requirements of digital computers has meant that technology for volatile and non-volatile digital storage has been a focus for the semiconductor industry. Very high density and low-cost storage is available. Digital storage of network parameters also allows compression algorithms to be used to reduce storage requirements. Use of volatile storage necessitates regular backups of the parameters to secondary storage. New high-density, non-volatile and re-writeable memory technologies such as FRAM appear promising for use in neural network simulators.

Synaptic weight storage on analog VLSI chips can be via fixed resistors, floating gates, or charged capacitors, achieving precisions of up to ten bits. Higher precisions are possible with digital weight storage and accompanying digital to analog converters. By storing an analog value on the floating gate rather than a digital value, storage densities that rival non-volatile digital densities are possible. Capacitor-based storage is volatile, and must have much greater capacitance values (and hence areas) than digital storage capacitors so that the stored analog value can be accurately retained and refreshed.

Optical storage in arrays of spatial light modulators, holographic plates or volume holograms has potentially the highest density. Re-writeable storage is, however, a problem.

Communications

Analog communication makes better use of wire bandwidths than digital communications. This outweighed in fully digital systems by more difficult output switching, and the need for data converters at either end of a link. Optical connections can be in free space, overlap with each other, have high fan-out and can come directly from input objects in vision applications. They therefore have great potential if they can be applied cheaply and precisely.

Control

Digital control is required in virtualized simulations to sequence data fetches and computations, and to switch outputs. The simplicity and repetitive nature of calculations required for neural network simulation limit the required complexity of the control.

3.3 Hardware Architectures and Algorithms for Neural Network Simulation

Efficient hardware architectures and software algorithms for neural network simulation are dependent on the system performance targets described in Section 3.1. Most important will be the range of architectures and forms of connectivity of the networks to be simulated, whether the networks will be pre-trained or must be adaptive, and the required system throughput. Overspecifying a simulation target can lead to a much more costly implementation. Specifying high simulation flexibility is often very costly, as it rules out analog implementations that are dedicated to a single network architecture.

Even for a given performance target there will be tradeoffs between:

— the amount of storage duplication, inter-processor communication and input pipelining,

— the amount of virtualization and processor cost, processor speed, inter-processor communications bandwidth, communications latency and communications overhead, and

— storage density and off-chip I/O bandwidth.

Technology parameters will determine the optimal system form.

Hardware and software for efficient simulation is heavily dependent on the type of connectivity between neurons in the networks to be simulated. Architectures for the simulation of completely and sparsely connected networks with therefore be considered separately.

3.3.1 Simulation of completely connected networks

By the definition of Section 2.2.2, a neural network is completely connected if, at each simulation time-step, each axon synapsing with a neuron being updated at that time-step synapses with all neurons being updated at that time-step. This definition leads to some regular architectures for communication and computation.

The Mesh/Torus

A mesh is the natural layout for an analog VLSI implementation (Fig. 5).

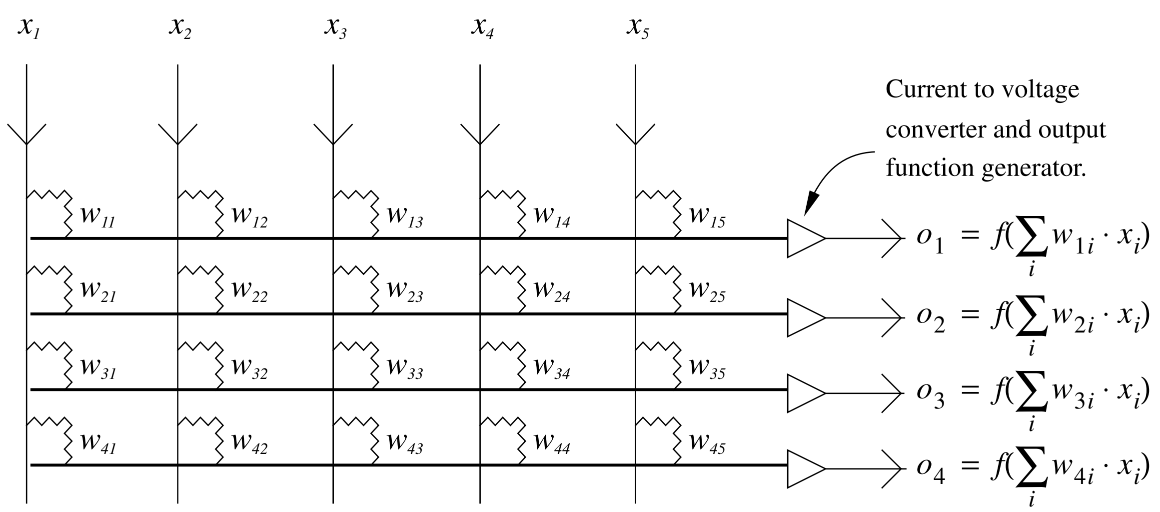


FIGURE 5.

Mesh layout for implementations of completely connected networks in analog VLSI.

Each neuron has a dedicated summing bus for its synaptic inputs. Outputs can be routed back as inputs. Layered completely connected networks can be implemented by abutting chains of blocks, each rotated 90 degrees from the last, possibly crossing chip boundaries.

Digital implementations can not make use of a summing bus and so, as a large multi-input adder is impractical, must accumulate synaptic inputs systolically (Fig. 6). Inputs are propagated vertically, while the synaptic inputs of each neuron is accumulated horizontally. The inputs must be staggered so that they are calculated at the right time, meaning that, for the example of Fig. 6, it will take 9 clock cycles for o_4 to appear. The inputs can, however, be pipelined.

The systolic mesh is ideal for simulation of Backpropagation networks because, in the backward pass, output errors can be passed horizontally and accumulated vertically. Bi-directional links or extra links to form a torus must be used to make this possible.

It is not necessary to have one processor for each synapse. The simulation can be virtualized by mapping rectangular portions of the weight matrix to each processor, and slices of

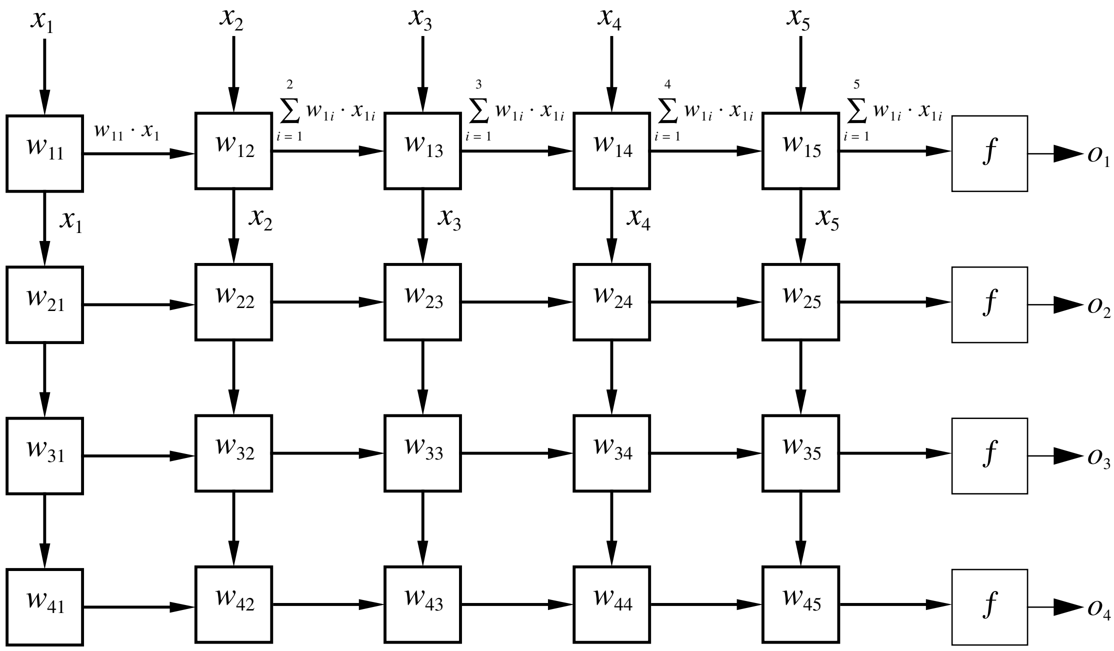


FIGURE 6.

2-dimensional systolic architecture for simulation of completely connected networks.

the input and output vectors to each column and row. This makes a given processor array able to simulate networks of various sizes, given sufficient memory at each processor.

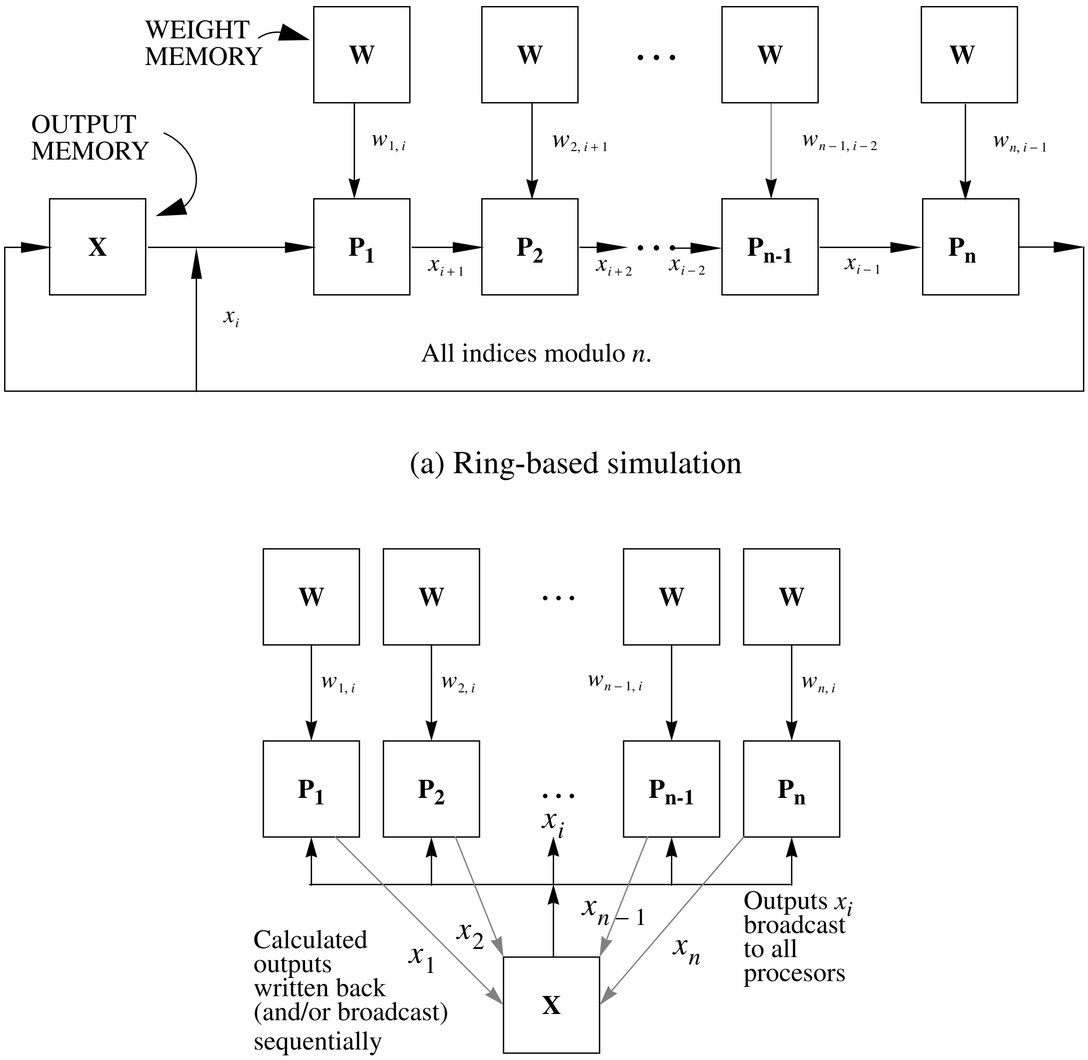
The advantage of the mesh is its ability to achieve maximum parallelism and hence maximum throughput. Its disadvantages are its high inter-processor communications requirements, the need for low communications latencies even at high levels of virtualization, and the low processor utilization unless the inputs can be pipelined.

The Ring

The amount of virtualization can be increased by collapsing the horizontal axis of the mesh, leaving a one dimensional processor pipeline. As the inputs pass through the chain, each processor must compute the full update of all neurons that are mapped to it. Synaptic accumulation is now internal to each processor. A ring is formed by joining the end processors by a communications link to enable the calculated outputs to be recirculated as synaptic inputs to the same or another set of neurons, or to participate in a Hebbian weight update cycle (Fig. 7a). There remains the pipeline delay for the inputs to visit each processor.

The ring is also suitable for Backpropagation simulation for two reasons. First, the ring can be used in the backward pass to efficiently accumulate output errors. Second, it suitable for the “case-based” simulation method in which the whole network is mapped on to each processor, and each processor works with a different input vector. The ring is used to systolically sum the weight changes calculated for each input vector. If a large number of inputs are processed between weight accumulation, the inter-processor communications load can be greatly reduced at the cost of duplicated storage of each weight.

Hypercube type connectivity can emulate a ring by numbering the processors in grey-code sequence.



(a) Ring-based simulation

(b) Bus-based simulation

FIGURE 7.

Simulation of completeley connected networks on ring and bus architectures.

The Bus

The pipeline delay of the ring can be eliminated by connecting the processors to a bus that can broadcast synaptic inputs, that is either external inputs or calculated outputs, to all processors (Fig. 7b). The total communications bandwidth is therefore greatly reduced.

However, a bus is not efficient at communicating different data between pairs of processors, as is required to accumulate output errors in the backward pass of a Backpropagation simulation. The situation can be salvaged if each processor stores the synapses formed with the axons as well as the dendrites of all neurons mapped to it. That is, for every row (neuron) of the weight matrix mapped to a processor, the processor also stores the corresponding column. Thus, for double the storage cost, only output errors need to be communicated rather than many partial error sums. The bus is also suitable for “case-based” Backpropagation simulation, as the weight changes can be broadcast to each processor for summing.

Crosspoint Switch

A crosspoint switch can directly simulate a bus and, in a small number of steps, simulate an ring or mesh. Its main advantage is to eliminate the pipeline delays of the mesh an ring by broadcasting inputs, while providing the communications topology required for efficient simulations of the backward pass in Backpropagation.

3.3.2 Reducing off-chip bandwidths in highly virtualized simulations of completely connected networks

In highly virtualized simulations, thought must be give to the order of computation on each processor and its impact on storage requirements and off-chip bandwidths. In highly virtualized simulation of large networks, the number of synaptic weights that need to be stored local to each processor will be too large for on-chip storage and so will have to be fetched from external memory. However, storing the inputs or intermediate results of computation on-chip can minimize the required external chip I/O bandwidth. This is often the simulation bottleneck when employing high density VLSI, with which on-chip data fetches can be done much faster than external fetches. This is indeed true for the processor to be used in the simulation system to be described in Chapter 5.

The fundamental operation performed by a processor, independent of the chosen simulation topology, is the calculation of vector dot products between an input vector or input vector slice and many weight vectors. With sufficient on-chip storage, two methods can be used to prevent input values being externally fetched more than once. At one extreme, the dot products can be calculated, one after another, using an input vector stored on-chip; at the other, a single input value can be multiplied with a set of weight values, with each product being added, in sequence, to a corresponding partial-sum for each dot product, for which on-chip storage must be allocated. The second method is useful to allow computation to proceed while input values are being communicated. The rate of computation can be maintained if the partial-sums are stored in dual-ported registers so that the old sum can be fetched, and the new sum stored, in the same clock cycle.

However, if there is not enough on-chip storage for inputs or partial-sums, they must be fetched from (and, in the case of partial-sums, written back to) external memory each time they are used, greatly increasing the necessary chip I/O bandwidth. For the case where there is a small amount of on-chip memory, say 32 weight- and output-sized locations, a compromise can be struck that reduces the external chip I/O to close to that of the case of full on-chip storage. The on-chip memory is used as accumulators for a set of dot products. The required I/O bandwidth for inputs will now be a fraction of that for the synaptic weights as each input can be multiplied with a set of weights and the result added to the appropriate accumulators before the next input needs to be fetched. When all the inputs have been read, the accumulated values are either used to calculate the neuron outputs (bus and ring) or passed to the next processor (mesh). The accumulators are then cleared and the process repeats with a different set of dot products. There is an analogous compromise where the inputs are read in blocks, and a slice of all dot products updated by computing the slice corresponding to the current input block and adding it to an externally stored partial sum. The I/O bandwidth will be slightly higher for this variant.

This technique to minimize external I/O fetches will be used in Chapter 6 to calculate the speed at which the simulation system of Chapter 5 can simulate a Backpropagation network.

3.3.3 Simulation of sparsely connected networks

For neural networks with sparse effective connectivity, such as brain-like network architectures or networks where many weights are small, use of the simulation methods described in Section 3.3.1 would be wasteful of both storage and computation, as many of the weights would be zero. There have been five published solutions to the problem of sparse network simulation. Each must deal with the non-uniform communications structure, with software and hardware for routing outputs to their destination neurons.

1. Switched fully-connected blocks

This method is suitable if the network can be broken up into clusters of neurons that have dense connectivity within a cluster and smaller amounts of connectivity between clusters. Each cluster is simulated as a fully connected block using the methods of Section 3.3.1. Switches are used to route outputs to the appropriate inputs on the set of clusters it connects with. When determining the optimal cluster size, there will often be a tradeoff between the amount of output switching and the efficient use of storage and processors. At the most extreme, a full implementation uses a cluster size of one, and must route each output to all its destination synapses. This would be necessary in the worst case of a randomly connected network if redundant storage and computation was to be avoided.

The method is suitable for analog VLSI implementations as the efficient mesh layout can still be used. However, the method may not be suitable for networks where effective connections are added or deleted, either through learning or a constructive algorithm.

There are two related methods which are worth noting here. The first is the method of Gupta *et al.* (1990) for simulating the Hopfield-Tank neural network for solving travelling salesperson problem (TSP), in which they decompose the TSP weight matrix into a sum of matrices with regular structure and sum the effects of each matrix. The second is the simulation of so-called “convolutional” neural networks, where a fixed weight matrix is applied to a sliding input window to implement image processing (Graf *et al.*, 1991).

2. Simulations on distributed memory multicomputers

Ghosh and Hwang (1989) described a method for simulating sparsely connected neural networks on message-passing distributed memory MIMD computers. The method is suitable for highly virtualized simulations and constructive ANN algorithms. The network is partitioned into clusters in such a way that the amount of communication between the clusters is minimized. Each cluster of neurons is mapped to a different processor, and each neuron is updated as shown in Fig. 8.

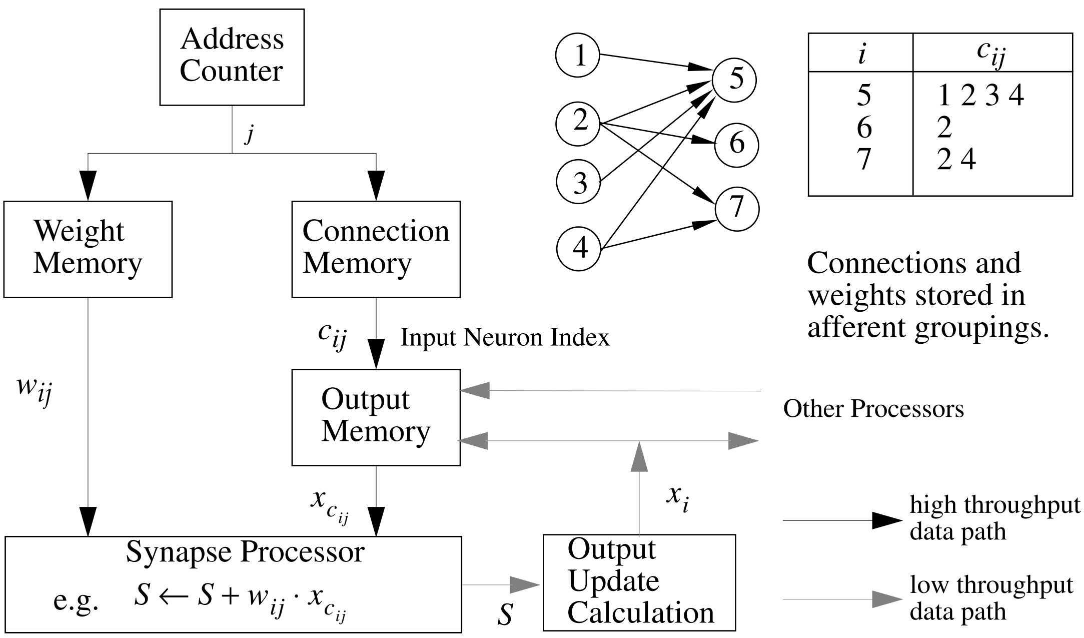


FIGURE 8.

Dataflow diagram for the simulation of sparsely connected networks on message-passing computers. Neuron *i* is shown being updated.

The output of a particular neuron is calculated by simultaneously reading synaptic weights and connectivity information sequentially from tables in memory. The connection information is used to index output values on the incoming axons. Synaptic inputs are accumulated, and the computed activation written back into output memory. The output is transferred to other processors that simulate neurons forming synapses with the output. Outputs must be re-numbered on receipt so that they can be placed at the correct index in the output table, ready for the next update cycle.

To minimize the required communications bandwidth and the amount of output re-numbering, the amount of virtualization should be as high possible, consistent with the required throughput. As well, an efficient multicasting algorithm should be used to deliver outputs to a set of target processors, rather than send them separately to each target. This becomes more difficult the more random the connectivity. Ghosh and Hwang suggested that outputs could be broadcast, with each processor deciding to accept or reject the outputs it receives.

3. Simulation on SIMD hypercubes

Rosenberg and Blelloch (1986) describe a method for simulating sparsely connected neural networks on the Connection Machine — a SIMD machine with hypercube connectivity. One (virtual) processor is allocated to each neuron and two to each synapse. The hypercube is unwrapped into a processor chain, with each neuron processor flanked to the left by a line of processors that implement the synaptic computation for each synapse made with the neuron's dendrites, and to the right by a line of processors that route the neuron's output to the synaptic processors for each synapse formed by the neuron's axon. The synaptic outputs are accumulated systolically through the synaptic processors and passed to the neuron processor. The calculated neuron output is shifted to all routing processors. Backward routing can be used for simulation of Backpropagation networks.

The price paid, however, for simulating a non-uniform structure on a SIMD machine is duplicated storage and inter-processor communications, and idle processors.

4. Simulation on SIMD machines with local communications

Tomboulian (1987) describes another method for SIMD simulation of sparsely connected neural networks, suitable for machines where processors have communications links to a small set of nearest neighbours, such as a mesh. Each processor simulates one neuron and stores the weights of synapses formed by the neuron's axon. Routing registers are set up to route, through a series of processors, weighted neuron outputs to their destination neuron for accumulation. The routing is designed so that each processor receives only one weighted output each clock, which can be forwarded on the next cycle without blocking.

The method works well for very sparsely connected and reconfigurable networks, yet the limited processor connectivity and single instruction control has lead to large storage and communications overheads, making it much less efficient than simulations on MIMD machines with more global communications.

5. Simulation on SIMD orthogonal bus machines

Misra and Kumar (1990) describe a sparse simulation algorithm for a SIMD machine with orthogonal buses, a communications topology also known as a reduced mesh of trees (RMOT). The hardware architecture comprises a two-dimensional array of memory modules, connected by a set of orthogonal buses, with the processors connected at the points where the row and column buses meet (Fig. 9). Each clock cycle, all processors perform either row or column accesses, thus avoiding memory contention. The architecture can be virtualized in software to emulate a larger array of smaller memory modules with a corresponding larger number of processors.

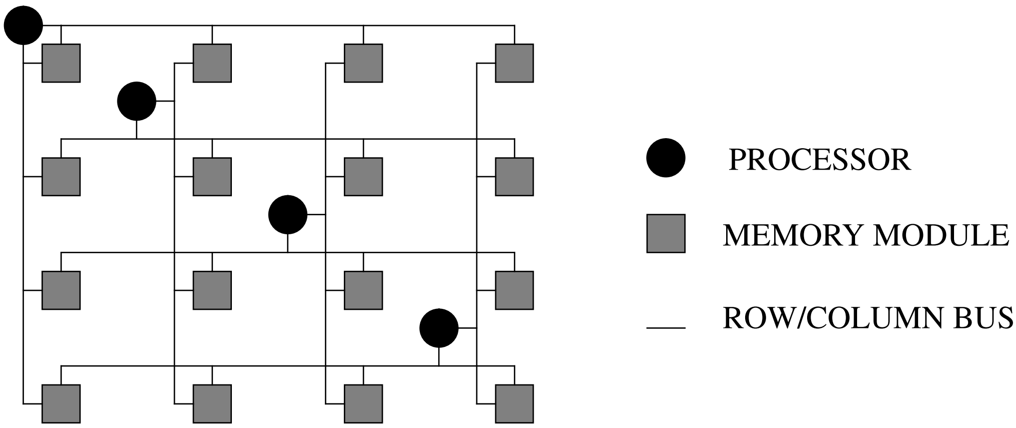


FIGURE 9. The RMOT architecture.

Each synapse and neuron is allocated a (virtual) memory module. The sparse weight matrix is mapped to the array of memory modules by scanning, snake-like, up and down the columns of the weight matrix and mapping synapses with non-zero weight to the memory modules in the same snake-like fashion. Thus weights in the same column of the weight martrix will be mapped to a group of adjacent memory modules. Simulation proceeds by:

1. routing inputs to the column groups and distributing the input to each synapse module in the group,

2. multiplying the input and weight in each module to calculate synaptic outputs,

3. permuting the data in the memory modules so that the synaptic outputs of a given neuron are in a contiguous group of memory banks,

4. summing the synaptic outputs in the group to calculate the total input for each neuron, and

5. sending the sum to the appropriate memory module.

The advantages of this simulation method include:

— a clear algorithm for mapping the network to the architecture,

— the ability to simulate the backward pass of Backpropagation networks by reversing the permutation process,

— the ability to fully parallelize the computation, and

— single instruction control of the simulation.

Disadvantages include:

— the high communications requirements (the single control stream has meant that synaptic outputs have had to be communicated between processors for each of the SIMD methods considered),

— the memory overhead for the registers that route the neuron and synaptic outputs, and

— the difficulty of adding or deleting synapses or neurons.

When large networks are to be simulated, the processor underutilization and high memory and communications overheads of the SIMD simulation methods are intolerable. If, in addition, the effective connectivity of the network is not-uniform or varies, the switched block method will not be suitable, and the Ghosh and Hwang method will be the method of choice.

The SIMD techniques are also unsuitable for the simulation of "heterogeneous" neural architectures such as the Counterpropagation ANN (Hecht-Nielsen, 1990) and various brain-like architectures which have multiple neuron types or weight modification rules. In any case, there is little cost penalty for choosing MIMD over SIMD. The simulation algorithms are so simple that there does not have to be large amounts of instruction memory at each node and, in addition, ALU/Instruction unit combinations are the common configuration for low-cost commodity processors.

3.3.4 Simulation of Networks with Sparse Neuron Activation or Slow Changing Outputs

The Ghosh and Hwang method for simulating sparsely connected neural networks calculates the effect of every synapse on every cycle, and so wastes considerable computation when simulating networks where the outputs of neurons change, or are active, at intervals much greater than the simulation time-step. Such networks include,

— competitive networks, or networks with phase-coded outputs (see Section 2.1.1), where most neuron outputs are zero at any time,

— networks of spiking neurons,

— simulations where the necessary neuron latency is much shorter than the time between output changes, and

— asynchronously updated networks.

There is no simple way to adapt their algorithm to allow only the synapses that need updating to be calculated at each simulation step.

The next chapter will describe efficient algorithms for simulating large networks of the above types, and consider methods to reduce the amount of memory needed to store the network connectivity and synaptic weights.

Chapter 4

Efficient Simulation of Brain-like Neural Networks

The cerebral cortex is distinguished as a neural network by its large size and its sparse and variable effective connectivity. These attributes make the simulation method of Ghosh and Hwang the best of those discussed in the preceding chapter. However, the cerebral cortex is also distinguished by neurons with small latencies and limited firing rates that compete with each other for activation. A simulation of such a network will therefore be characterized by very few synapses being active at each simulation time-step. The Ghosh and Hwang method updates a neuron by calculating, at each time-step, the influence of every synapse which the neuron's dendrites make, meaning that much computation is wasted if most synaptic inputs are inactive or do not require updating at any given time. This chapter describes variants of the Ghosh and Hwang method that simulate adaptive brain-like networks in a computationally efficient way. Consideration is also given to minimizing the amount of storage required for synaptic weights and neuron connectivity information.

4.1 Axon-major Simulation

Fig. 10 depicts a simulation dataflow that restricts the calculation of synaptic influences to those formed by a selected set of axons. Just as with the Ghosh and Hwang method, the network is partitioned so that each processor calculates the dendritic synapses and outputs of a subset of the neurons in the network. However, synaptic weights are now grouped in memory according to common pre-synaptic axons (efferent grouping) rather than common post-synaptic neuron (afferent grouping), as was the case in Fig. 8. The familiar multiply-accumulate process to calculate the total synaptic influence on a neuron cannot be used, as a

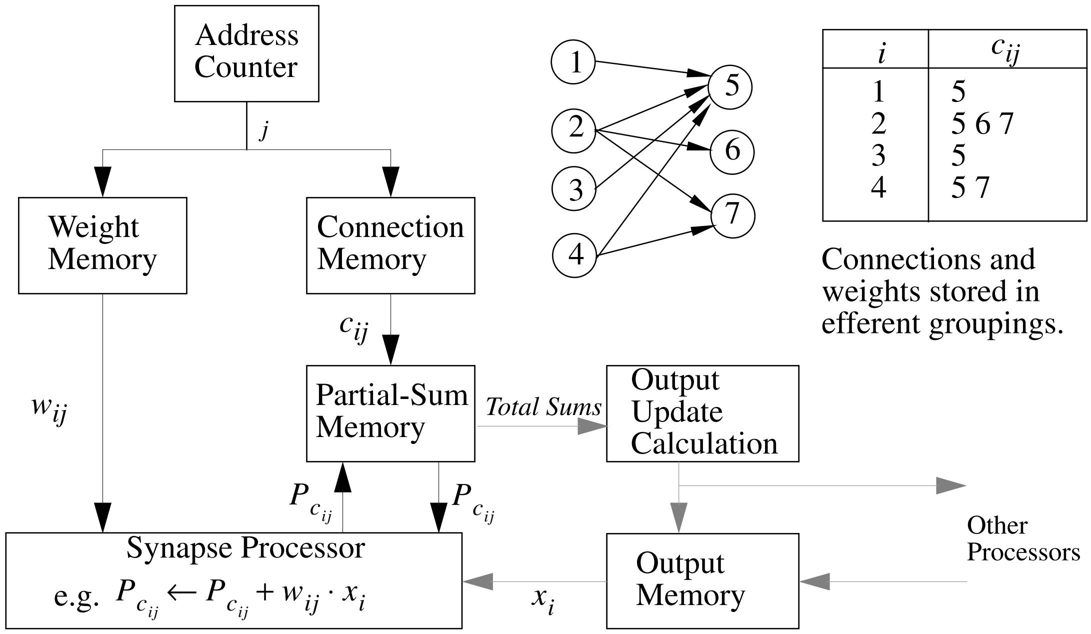


FIGURE 10. Axon-major simulation. Neuron *i* is shown being updated.

partial sum of synaptic influences for each neuron forming synapses with the current axonal output must be fetched from, and written back to, memory. The connection information is now used to index a table of partial-sums. Every time-step, after all necessary axons have been processed, the total activation sums are used to update the neuron outputs.

Although this structure yields higher I/O rates, and requires storage of partial-sums of synaptic input for each neuron mapped to a given processor, it can be justified if only a small percentage of synapses need updating at each simulation time-step. There are five reason why this will be true for brain-like networks.

1. Competition between neurons

As discussed in Section 2.2.1, neurons in the cerebral cortex compete with one another for activation via mutual inhibition. This inhibition causes different neuron modules to learn to respond to different combinations of active afferent axons. The result is that few neurons will have non-zero output over reasonable simulation time-step intervals.

This is especially true in high-level association modules of the cerebral cortex. For example, Miyashita (1990) has found that individual neurons in an area of cortex known to store short and long term visual memories respond to less than 10% of visually presented patterns. Selectivity increased as the stimulus became more familiar. Similar selectivity is likely in cortex subserving high-level language faculties. Anderson and Moser (1989) have estimated the percentage of neurons whose activities are changed by a given input to be between 1 and 10 percent.

Competitive networks of spiking neurons can only support a limited total spike rate and so there will be a limit on the computational load.

2. The Nature of Spiking Outputs

When simulating networks of spiking neurons, the simulation time-step must be small enough to accurately simulate the neuron's latency, the integration of synaptic inputs, and the spike arrival and generation times. Neurons with a few hundred hertz maximum firing rates will be silent for a number of time-steps between spikes, meaning that the synapses that they connect to will not be active. The rise and fall synaptic currents triggered by spikes arriving at a neuron can usually be simulated as a single amalgamated synaptic current because the time-evolution of synaptic current is a sum or difference of exponential terms, permitting use of superposition. For example, all synapses with currents that decay exponentially at the same rate due to the speed of their ion channels or dendritic position (which determines the effective input capacitance) can be simulated by adding a value representing the input strength to a single variable when the spike arrives, and decaying the value in the variable at the given rate. Alpha functions can be simulated with two variables (Wilson and Bower, 1989). Modelling of the temporal behaviour of synaptic currents can therefore be shifted to the output update calculations.

3. Slowly evolving outputs

Even if the average firing rate neuron model, rather than the spiking model, is used, all synapses with active (non-zero) input axons need not be updated every time-step. First, a maximum rate of output change will be imposed on the output to mimic the rate of change of spiking outputs, which must spike to signal a change. But also, slow changing external inputs to the network and the effects of positive and negative feedback within the network often results in an even slower rate of output change. Such networks can therefore be simulated by using the stored total input sums as a base from which changes in input on the following simulation time-step can be added and subtracted. Only outputs that have changed during that cycle need be communicated between processors and have their synaptic influences calculated.

4. Phase coding of outputs

As discussed in Section 2.1.1, there is evidence that the brain can segregate or bind neural activity by varying the phase of oscillating neuron outputs. Two frequencies of oscillation have been observed, one around 10 *Hz* (the theta rhythm) and the other about 40-60 *Hz* (the gamma wave). The synaptic influences of the neuron's axon need not be calculated at the phases when the neuron is silent.

5. Weight modification

In adaptive networks, updating a synapse means not only calculating the synapse's contribution to the membrane voltage, but modifying the synaptic weight according to a weight modification law. With the exception of Competitive Learning, the weight modification rules covered in Section 2.1.2 specify a weight change that is triggered by pre-synaptic activity. This means that weight modification calculations can be limited to the same synapses that have had their influences on the neuron's output calculated recently.

If a neuron has multiple input types with, say, different decay constants or different influences on the neuron's output, multiple partial-sums can be allocated. For example, a partial sum can be allocated for excitatory inputs close to the cell body, another for distal excitatory inputs, and another for slowly evolving inhibitory inputs. When a neuron's output is updated by combining the final set of total sums, the activation can be redistributed amongst these sums so that they truly reflect the current membrane voltage at the synapse type that the represent. The value can then be used in Hebbian synaptic weight updates.

Also, if axonal delay time must be simulated, such that a given neuron output activates different sets of synapses at different times, the efferent synapse and connection lists can be split into multiple time bins and calculated at the appropriate time.

A number of other advantages accrue from the partial-sum calculation process.

Synapse Swapping

Large sets of neurons can be inactive for long periods of time in networks with sparse connectivity and strong cooperative and competitive interactions. This is particularly true for large networks that accept highly variable input. For cost-effective simulation, it would be desirable to "swap out" the synaptic weight and connectivity parameters of inactive synapses into low-cost secondary storage. Although few neurons will receive enough active input per cycle to fire, many neurons will receive at least one active input. Thus although it is not possible to swap out blocks of weights and connections grouped afferently, it is possible if they are stored in efferent groups.

Real-time simulation is still feasible if weights and connections are swapped only if they are likely to remain on disk for times much longer than the update rate. This can be done by swapping out the efferent parameters of neurons that are still in memory and have the least recent activity, to make way for the swapped parameters of newly activated neurons. Stereotyped cooperative and competitive interactions such as lateral-inhibition can still be calculated without output connections needing to be retrieved, as the connectivity and weight information of this type can be stored as formulae. The full output parameters of a neuron need only be swapped in if it wins a competitive battle for activation. Swapped data must be able to be fetched from disk within one time-step plus synaptic and axonal delays. This should be possible with the use of fast-seeking magnetic disks.

The potential of swapping is partially confounded by the distributed output coding found in neural networks with moderate competition. Each neuron codes for a micro-feature which is present in more than one input, making activation more probable than situations where a neuron codes for a single input.

Each processor should control the swapping of the synapses mapped to it, rather than the home-processor of a particular neuron informing other processors which simulate synapses formed by the neuron's axon that all such synapses should be swapped out. Variations in activity across the network will mean that some processors will be short of memory and require swapping, while others will be simulating many dormant neurons and so will have sufficient memory to retain the synapses. Memory will be most efficiently used if the dormant neurons are evenly spread across the processors. This is not likely if nearby clusters of neurons are mapped to different processors, as neuron modules activate as units. A more diffuse mapping can even out the memory load, meaning that there will often be a tradeoff between the efficient use of memory and the communications bandwidth.

Locality of Addressing

The connection memory only indexes neurons mapped to the local processor because these are the only neurons for which a partial activation sum must be stored and computed. The outputs of all neurons that provide input to the local group of neurons must be indexed if the method of Fig. 8 is used. The restricted indexing requirement and the partially stereotyped connectivity of neural networks such as the cerebral cortex mean that the connectivity can often be stored in a highly compressed form, saving memory. Another consequence is that the need for fast random access is restricted to a smaller amount of memory.

As well, there is no need to store all outputs forming synapses with neurons mapped to a processor. Outputs can be processed from an input queue.

Simulation of Backpropagation Networks

Backpropagation networks perform calculations on weights in both afferent and efferent sequences during respective forward and backward passes through the network. Weights in fully connected back-propagation networks can be stored in two-dimensional matrices to allow reference in either order via sequential row or column accesses. However, in sparsely connected back-propagation networks, it is wasteful of both storage and computation to work with the sparse weight matrix directly. In order to employ the sparse computation method of Fig. 8, the weights and connections must be stored twice to permit access to the weights and outputs in both afferent and efferent orders. Weight updates must be written to two areas of memory. The provision of a high-speed write channel to indexed memory allows use of the sparse method of Fig. 8 on the forward pass, and partial-sum accumulation of output errors on the learning-mode backward pass. Weights need only be stored once for pipelined access in both passes. Partial output error sums will still need to be communicated to the home neuron processor to be summed.

Order sensitive weight update

In a possible biological learning method, synapses are primed for modification by powerful input. The synaptic weight is changed according to the state of activation of the post-synaptic neuron within a following time window, according to a Hebbian algorithm (Larson and Lynch, 1989). This method of weight modification allows the formation of temporal association chains. If weights are stored in efferent groups, this rule can be simulated by changing weights at each step as a function of a time-smeared version of each axonal input. Learning algorithms where strong input can also prime adjacent synapses (Alkon *et al.*, 1990) or include heterosynaptic depression (Section 2.1.2) will need more complicated data structures and update sequencing.

Efferent grouping of synaptic calculations has been previously used in the GENESIS simulator program developed at the California Institute of Technology (Wilson and Bower, 1989). Their motivation for using axon-major synaptic update was the second reason discussed here — "the nature of spiking outputs."

In GENESIS, data is contained within "elements" (each element being a computational entity). Although the synaptic connection and weight information is associated with the axon (pre-synaptic), they don't appear in grouped tables. However, the processing is the same. For each axonal projection, events arriving at the synapses are passed to a synaptic function with the connection information (weight and destination) as argument. In a like manner, the elements of the partial sum memory are not grouped in a contiguous memory, but are associated with the destination dendritic compartment. Axonal delays are handled by maintaining a buffer of events generated at the soma (cell body). The buffer is scanned each time step and any event within the synaptic duration period (after propagation and synaptic latency delays) gets processed.[2]

GENESIS has been applied to the simulation of networks of highly realistic models of biological neurons, and detailed models of single neurons. The axon-major simulation dataflow of Fig. 10 is less general than the GENESIS approach, yet is suitable for high speed simulation of more abstract networks on hardware such as that to be described in the next chapter.

In summary, the work described in this section has,

— extended the applicability of axon-major synaptic calculation to competitive networks, networks of neurons with slowly evolving outputs, networks employing phase coding, adaptive networks using biological weight update laws, and Backpropagation networks,

2. This paragraph is adapted from a description of GENESIS provided by Dave Bilitch at Caltech.

— presented an axon-major dataflow, suitable for high-speed execution on pipelined hardware, and

— outlined a method of synapse swapping for reducing the amount of high-speed memory needed to store synaptic weight and connectivity information.

4.2 Efficient simulation of brain-like networks with dense clustered connectivity

A variation of the axon-major simulation method can be used for networks with locally dense connectivity between neurons. It is a version of the "switched fully-connected blocks" method of Section 3.3.3, in which only selected axons are processed in each cluster.

Synapses are stored in afferent groupings but are accessed non-sequentially by a table of active/changing neuron outputs (Fig. 11). The input/output bandwidths are improved over the axon-major method but each neuron must form synapses with all axonal input to the cluster. When simulating networks of spiking neuron, no output value need be fetched.

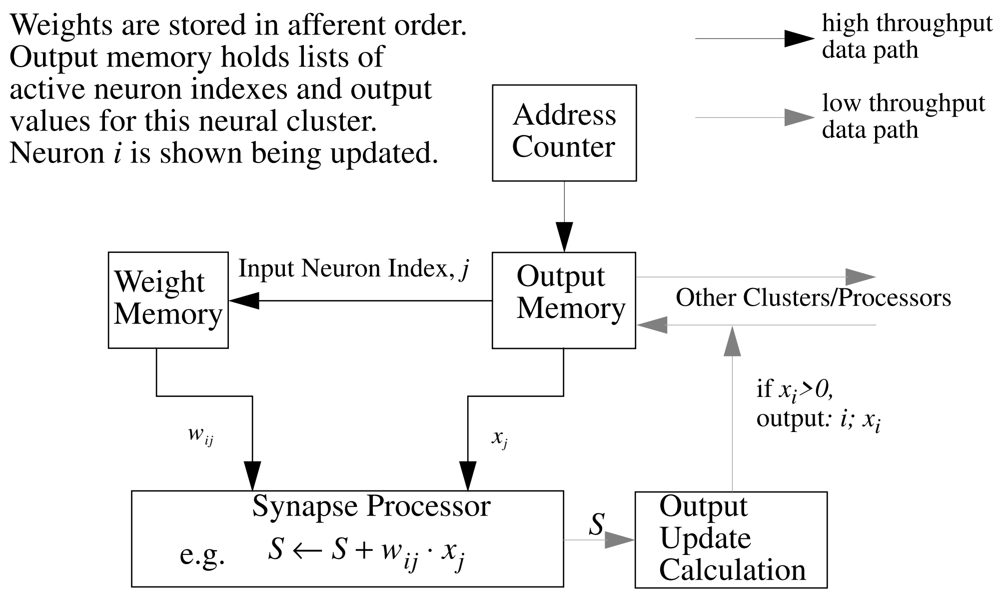


FIGURE 11.

Dataflow for simulation of brain-like networks with dense clustered connectivity.

However, if few axon influences need be calculated at each time-step, few memory access will be to sequential locations and memory pipelining will not be possible in some hardware implementations.

It is not necessary to have only one cluster simulated on a processor. The optimal cluster size will be a tradeoff between the amount of wasted computation in calculating zero weight synapses, and the amount of output switching or re-numbering that must occur when the outputs are communicated between clusters. If lateral inhibition provides fixed inhibitory input to all neurons in a cluster, it can be simulated as a cluster-wide output threshold shift.

The method is suitable for simulation of independent competitive neuron clusters, a “macrocolumn” (Section 2.2.1) for example, that interact via patchy connections. Overlapping clusters would be more difficult to simulate.

4.3 Techniques for reducing storage requirements

The size of the network that can be simulated is determined by the amount of memory available and the effectiveness of any swapping strategies employed. By factoring out regularities in the data to be stored, storage can be conserved in a number of ways.

First, the precision with which synaptic weights are stored can be varied according to need. Second, as inhibitory synapses are incapable of learning in many models, all inhibitory synapses of a particular type can have a fixed common weight. This means only the connectivity information must be stored for each of these synapses. Third, even synapses capable of modification can have a common “naive” weight until a given activity pattern causes the weight to change, forming an association. With some models, decorrelation between neurons leads to depression of the synaptic weights between them to a small value that can be chosen to be the same for all highly depressed synapses. As before, the common weight need not be stored separately for each synapse. If this technique is used, the addition of noise may be required to break ties in activation strength during competitive learning. Fourth, much of the connectivity in many models is stereotyped or of low entropy — that is, non-random — so the connections can be stored in a compressed state, and many connection addresses can generated from formulae.

Because different networks require differing ratios of weight, connection and output/partial-sum memory, efficient use of memory demands that each parameter not be stored in its own fixed size memory block as suggested by the dataflow diagrams. Rather, for efficient memory usage, the different parameters must be spread among the available memory blocks while still allowing access to each parameter in parallel or burst access to each parameter in turn.

Constructive ANN models (Fahlman and Lebiere, 1990; Hanson, 1990) allow the creation and deletion of nodes and connections as required. Such an option is not available in biological neural networks. Growth of new connections in the cerebral cortex is limited so connections must be laid down during development to handle all possible future input and allow the functions of areas that suffer damage to be taken over by adjacent neurons (Jacobs and Donoghue, 1990). The number of useless connections is minimized during development by culling an initial overproduction of connections through activity dependent self-organization (Miller and Stryker, 1990). The marrying of the power of biological networks with the storage efficiency of constructive networks could be an area for future research.

4.4 Estimated System Size for a Speech Recognition/ Understanding Application

A speech recognition system suitable for dictation must process input at least the level of the word. Speech understanding and language translation systems, capable of recognition and disambiguation at the syntactic and semantic levels, would require a larger network. How large would a neural network need to be to perform these tasks? This section attempts an estimate of the storage requirements for these tasks, only for the purposes of showing that a combination of memory saving techniques can make feasible the simulation of the very large networks required.

A sufficient network size for this task is the size and complexity of a network that is known to perform the task — the speech and language areas of the human cerebral cortex. The cerebral cortex has 10^{15} synapses. First, we need only simulate the speech and language areas — approximately 10% of total.

Second, a minicolumn can be represented as a single complex neuron with multiple input and output types (Section 2.2.1). By eliminating the simulation of synapses between neurons in a given minicolumn and collapsing multiple synapses between pairs of minicolumns to a single, stronger synapse, it is not unreasonable to estimate that only 1% of synapses remain.

Third, although hampered by the use of feature sharing through distributed coding, large numbers of neurons in semantic and lexical memory cortices should be inactive until a particular area of knowledge is accessed. A reasonable estimate is that only 10% of the network need be swapped-in at any one time.

Finally, we must apply the storage saving techniques of Section 4.3 to reduce the storage required for the 10^{11} remaining in-memory synapses. A minicolumn has far from random connectivity, forming both dense local connections and dense connections to selected patches in the same and other modules (Section 2.2.1). Considering there are approximately 100 minicolumns in a patch, storage of the connectivity should be dominated by the storage of synaptic weights. Every pair of connected minicolumns will not be connected with high weight. Competition for activation makes it likely that the micro-features encoded by a minicolumn are formed from a small subset of incoming axons. Also, the seemingly unbounded ability of people to learn more words, languages and concepts suggests that high language competency still leaves minicolumns unallocated or unassociated. Although highly speculative, we will estimate that only 5% of all weights are not “naive”, depressed or stereotyped and so must be individually stored. Consequently, the storage density will average to 20 synapses per byte.

Therefore, around 5 *Gbytes* of memory would be required. A proportion of this storage is for "unused" connections that may be able to be pruned after the network ceases training, or avoided through a constructive learning algorithm.

Chapter 5

A Multicomputer for Neural Network Simulation

This chapter presents the design of a multicomputer for low-cost simulation of large neural nets, based on the issues discussed in the previous chapter. ANN simulation boards exchange neuron activations with each other across a physically reconfigurable, hierarchical, message-passing communications network. The simulation engine is a high-speed, low-cost array processor and the network parameters are stored exclusively in low-cost, high-density Dynamic Random Access Memory (DRAM). The ratio between local memory size and processor speed is optimized for real-time network update rates.

Despite such fast update rates, local memories can be made large enough to dominate the cost of the simulation board. This means the cost of the memory required to hold the network parameters would be a large fraction of the total system's cost.

The need for a new, dedicated machine to simulate large brain-like neural networks in real-time will be demonstrated in Section 5.1. The subsequent three sections will describe the simulation board, the inter-board communications network, and the support cards respectively.

5.1 Why Design a New System?

This section discusses why current commercial parallel computers and current dedicated neural network simulators can not cost-effectively employ the simulation techniques of Chapter 4 to simulate large neural networks in real-time.

As the system will need to be able to simulate a wide range of neuron behaviour and learning laws, and employ high levels of virtualization, use of analog technology is unsuitable (Section 3.2).

The simulation dataflows of Chapter 4 are suitable for implementation on distributed memory multicomputers. Commercial shared memory machines can not exploit the localized nature of the computation and communication, and loosely coupled distributed memory systems will have insufficient interprocessor bandwidth and excessive communication latencies.

Commercial general-purpose parallel machines do not have processor and memory architectures suited to neural network simulation. Large coherent caches are useless for algorithms that perform simple and repeated calculations on large amounts of local data. The processors are slowed down by a large and complex instruction sets, can not move data in parallel with computation, have ALUs of higher precision than required, and often lack high-speed multiplication units.

To create a low-cost real-time implementation, one must maximize the amount of memory per processor while ensuring that the processor is able to simulate the neurons in the local memory at real-time rates. For real-time emulation of the brain, the neural network must be simulated at about 100 *CUPSPW* (Section 2.1.1). Commercial distributed memory computers and most special-purpose neural simulators are not designed for real-time neural simulation and so are best suited to run small networks at higher *CUPWPS* rates (Hammerstrom, 1990) or larger nets at much slower rates (ANZA, 1988). Their local-memory-size to processor-speed ratios are respectively too low and too high for efficient real-time simulation. A system specially geared for real-time simulation of neural networks can use resources more efficiently and therefore be more cost-effective.

In summary, the goals of low-cost and real-time simulation of large networks will only be met by a special-purpose, distributed memory, medium-grained multicomputer.

5.2 The ANN Simulation Board

5.2.1 The Simulation Processor

Initially, fixed-point Digital Signal Processor (DSP) chips were considered as simulation engines because they satisfy most of the requirements for neural network simulation; namely high speed multiplication, little overhead for execution of loops, and the ability for I/O fetches the proceed in parallel with, or even independently of, computation. However, the speed of conventional current generation 16- and 32-bit DSP chips allow for only a small amount of memory to be attached to each processor for full memory utilization at real-time update rates. The chips would therefore take up a large fraction of the board real-estate and the cost of the system would be many times the cost of the memory.

The processor chosen for this design is the “Datawave” chip manufactured by ITT Intermetall (Schmidt and Caesar, 1991). The chip is a 4×4 MIMD wavefront array of 12-bit processors (Fig. 12). Each processor has memory for 64 instructions and 16 12-bit data

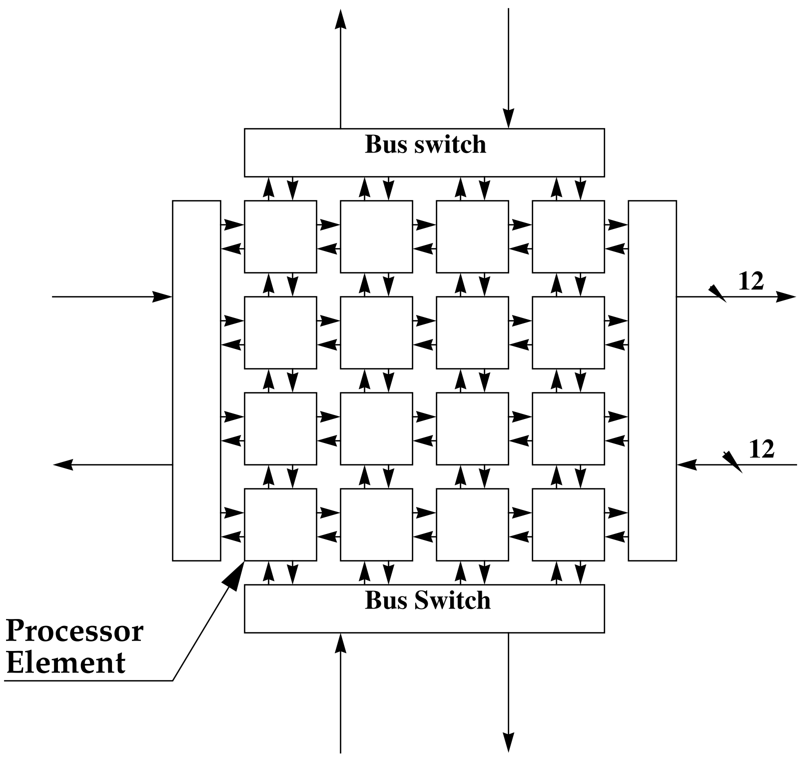


FIGURE 12. The ITT “Datawave” processor.

registers. Computation is performed by an ALU/shifter and multiplier-accumulator (MAC). Multiplication of 12-bit operands is accumulated to 29 bits. An instruction — issued every 8 *ns* (minimum) clock period — can specify simultaneous data moves, an ALU or MAC operation and a branch operation. The restricted computational precision allows such a high instruction rate, yet is sufficient for neural computation. Inter-processor communication links can transfer one 12-bit word per clock. Each is buffered by an 8-word FIFO. Off-chip communication is via 8 uni-directional ports, each capable of transferring one 12-bit word every two clock cycles, giving the chip a total I/O bandwidth of 750 *MBytes/s*. Though powerful, the chip is low-cost due to its CMOS implementation, its replicated structure, and its target application as a processor for digital television.

Banks of memory are connected to three of the Datawave's four bi-directional ports (Fig. 13). Each port simulates a separate block of neurons. Communication of neuron outputs between the blocks is through internal chip links. The fourth port is used to transfer neuron activations to and from other boards, and weights and connections to and from disk storage to support swapping. At least four processors can be assigned to each simulation port. This means that sets of functions such as calculations of synaptic influences, synapse weight updates, extraction of lower precision weights, connection table decompression, address calculation and noise generation can be performed simultaneously. This parallelism can also be efficiently exploited in networks where the synaptic influence and synapse update calculations are complex. If each processor is dedicated to a particular part of the simulation, the 64 word instruction memory should be sufficient for even complex neural network models.

5.2.2 Memory technology used

Simulating large neural networks at real-time update rates requires large amounts of high-speed storage to hold the synaptic weights, connectivity information, neural outputs and other network parameters. The dataflow graphs of Chapter 4 show that high-speed

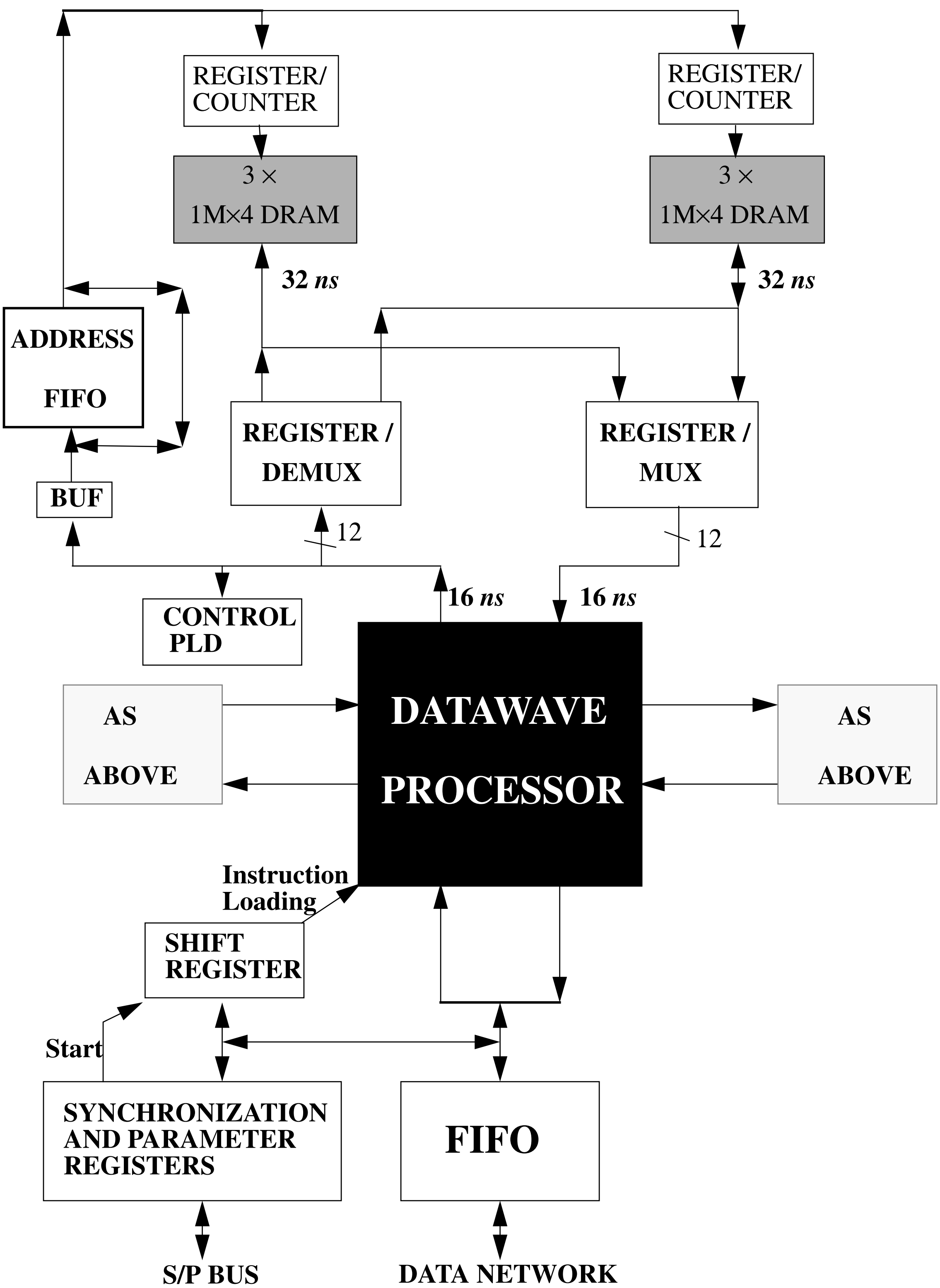


FIGURE 13. Block schematic of one possible ANN simulation board configuration.

sequential access to blocks of weight and connection memory will be required, while high-speed random access will be needed to the much smaller amount of memory holding outputs and partial-sums. The weights and connection lists for each neuron can be sorted so that the outputs and partial sums are addressed in a monotonically increasing sequence, enhancing the locality of reference to these memories. It is therefore feasible to use high-density page-mode DRAM chips for all parameter memory, as this form of memory provides fast random access to a selected small memory page.

To enable reads and writes from both weight/connection memory and output/partial-sum memory to be done in page-mode, either the memory system must allow at least two active page contexts for simultaneous fast access to both memories, or buffers on the processor must be provided to permit prolonged, alternating access to each. The burst access model is preferred as the proportion of I/O bandwidth required by the connections and weights will vary throughout the simulation according to the randomness of the connections and the precision of the weights that are currently being processed. No weights need be fetched if groups of synapses have a common weight (e.g. “naive” or highly depressed synapses) or can be generated by a formula (e.g. lateral inhibition). The Datawave can buffer data in both the link FIFO’s and in processor registers.

Burst access also minimises the complications that could occur at page boundaries with the fetching of partial-sum or weights and the rewriting of them a short time later. The alternative of ping-pong buffers, as used in the TRW Mark IV simulator (Hecht-Nielson, 1990), does not make efficient use of memory, which is necessary in large-scale simulations.

When simulating completely connected Backpropagation networks, use of the page-mode also permits full-speed access to both afferent and efferent weight orderings if the weights are stored as two-dimensional matrices within each page.

5.2.3 Memory Architecture

The stereotyped and repetitive nature of the network calculations mean that off-chip memory accesses can be pipelined, allowing the processor to run faster than the maximum access rate to one bank of memory. Just as in vector supercomputers, maximum throughput is achieved through round-robin access to a different memory bank every cycle. A multiplexer is used to funnel each memory bank in turn into a Datawave input port. For simulation of sparsely connected neural networks, we require fast random access as well as sequential access. This is a problem with such interleaved memory structures as there is usually no guarantee that consecutive random accesses will be to data in consecutive memory banks. Because only two-way interleaving is required here, the problem can be solved if partial-sums are stored as double words or if the synapses are sorted so that connections to odd and even neurons are interleaved. The odd and even partial-sums can be later summed.

Because the processor is data-driven, there are no address buses and the output port must be used for memory addresses as well as data. The address bandwidth is minimised through the provision of address counters, and an address FIFO to hold the addresses of the partial sums that have been fetched but not stored. The fact that address generation is fully programmable allows I/O to be fully decoupled from the processing of the data.

The configuration of Fig. 13 does not permit simultaneous read and write to memory at full speed. This is possible with a four bank per port configuration. One pair of banks can be fetching weights while the other is storing partial-sums. The address bandwidth means that the performance gain will be less than double that of the two bank configuration.

5.2.4 Estimation of the Optimal Amount of Memory Per Processor Port

If too little memory is attached to each of the three Datawave ports used for simulation, the processors will be under-utilized and the cost of systems designed to simulate networks of at least a given size will increase. Too much memory and the processor will not be able to simulate all the local neurons at a real-time rate (100 *CUPSPW*). The optimal amount of memory per port will now be estimated.

For networks with a large number of connections per neuron, most computation time is spent calculating synaptic influences. This means the time taken to calculate and communicate the output activation of a neuron, based on its total synaptic input, can be ignored. First, each port can fetch one 12-bit word every two 8 *ns* clock cycles, but approximately half of these fetches will be from the small amount of output/partial-sum memory that is read many times per update cycle. If the partial sum must be a double word to accumulate the inputs with sufficient precision, only the least significant word need be fetched at first. The most significant byte can be fetched if the least significant word overflows.

Second, weight updates can be done in the same pass as network update. The fetched partial-sum will be a good measure of the current post-synaptic activation level and can be used in Hebbian weight update calculations. A second weight update pass can be avoided. Alternatively, weight modification can be done periodically, rather than at each time-step, based on an averaged pre-synaptic activity and current post-synaptic activation.

Third, for some network architectures, a fraction of the weights and connections will be able to be generated internally by the processor as mentioned in Section 4.3. This loads the processor without using memory. Stereotyped connections implementing lateral-inhibition usually form fewer and more powerful synapses than modifiable excitatory synapses. Synapses with small depressed weights that are not stored (Section 4.3) will not need to be included in the synaptic influence cycle, and weight update is only necessary if the post-synaptic activation is high. A reasonable estimate is that 75% of synapses require individual weight or connection fetches every cycle.

Fourth, the assumption is made that, on average, only 10% of neurons providing input to the local group are activated or change their level of activation in any10 *ms* period. This is a conservative figure for networks with sparse connectivity and cooperative and competitive interactions (Section 4.1). Only 10% of the connections need be processed every iteration (although swapping could increase this.)

Thus,

$$\text{Memory/port/iteration} = \begin{pmatrix} \frac{1}{2}\text{word/port/clock} \times 1.25{\times}10^{6}\text{clocks/iteration} \\ \times 1.5\text{bytes/word} \times \frac{1}{2} \times 75\% \times 10 \end{pmatrix} = 4.5\,MBytes.$$

The cost of this amount of memory is likely to dominate the cost of a simulation card and so there is little cost penalty for conservative estimates of computational loads. Also, one need not worry about processors lying idle due to imbalances in the activity of neurons in the network. There is, however, a space penalty, as a system would have a higher number of lower cost nodes than would have otherwise been required. Balanced against this is the significant power dissipation of the processor running at its maximum rate (8 *W*) which could be more than halved if the memory per port and clock rate are both halved.

5.2.5 Alternative designs

The design requires a requires a modest amount of interface chips and glue logic to tie the DRAM to the processor port. These chips add to the cost of a system and take up valuable board space. Custom VLSI and Wafer Scale Integration would allow the DRAM and processor to be integrated on the same piece of silicon with savings in space and power dissipation. It remains to be seen if and when such implementations can be cost-competitive with designs using commodity memory. A middle course would be to design a chip that is more customized for the task at hand and so require fewer glue chips. Such a chip could have enough fast on-chip memory to hold all, or a cache of, partial sums, reducing the amount of external address and data bandwidths.

5.3 Architecture of the Interconnection Network

The system's communications network supports:

1. the transfer of neuron activations between simulation boards and between the simulation boards and I/O interface boards,

2. the swapping of weight and connection information to and from disk,

3. the loading and saving of network states, and

4. the loading of processor programs.

Only the first two are of high bandwidth.

The boards are designed to be connected to a 16-slot backplane via a single 32-bit wide communications network interface and a 16-bit wide synchronization / parameter (S/P) bus tap. The S/P bus has its own 8-bit address bus. The single wide bus is the most efficient form of communication when only moderate communications latency is required. Neuron outputs need only reach their destinations in a time less than the synaptic and axonal delays, or at worst one simulation time-step, if real-time simulation is desired.

The interface seen by the boards is independent of the connectivity provided by the backplane. This allows the use of either crosspoint switch backplanes or passive or active bus backplanes depending on the likely communication load patterns. Communication can be faster over the point-to-point links on crosspoint switch backplanes than on buses due to the lack of large capacitive loading effects. Narrower transfer widths can therefore be used to obtain the same port-to-port bandwidth of bus backplanes, making the use of crosspoint switches cost-effective.

Communications is implemented by message-passing. Data transfers on the backplane are directed by a microcontroller. Such central backplane control allows channel allocation and transfer sequencing to be optimized for a given application and cuts down the amount of on-card communications support and interface control required. Processors are not burdened with packet routing and dumb boards can be supported. A timer on the microcontroller can be used to pace real-time simulations.

The controller maintains a prioritized queue of pending transfers. Packets are sent to and from specified addresses in each card's memory space. Because the output of a neuron can project to many local and distant locations, a selective broadcast facility must be provided on the backplane to prevent communication saturation. Selective broadcasting is implemented with destination board selection masks in the packet headers. Address counters are required on cards with random access memory to obviate the need for an address to be sent over the network for every word transferred. Communication is synchronous at the maximum rate of the slowest card involved in a transfer. The controller has access to this rate for each card via the S/P bus. Parity error, data overrun or other problems during the transfer can be signalled via the S/P bus.

The S/P bus is used to synchronize the processing states of sets of boards on the backplane and to transfer message parameter information between the boards and the backplane control processor. When parameter information is not being transferred on the bus, the S/P bus addresses are cycled through a chosen range to continually read out a synchronization word of chosen length. There is a trade-off between the length of the synchronization word and the maximum synchronization rate. Unless synchronization bits need to be transferred between boards, the backplane controller need be the only reader of this information. Boards write into the S/P bus address space to request data transfers and provide information about the present local state of computation. Open collector drivers are used to allow wired-or and wired-and processing for the implementation of fast synchronization bits. S/P bus bits can be linked to processor interrupt lines or control aspects of a board's behaviour. The same synchronization scheme has been used by Hwang *et al.* (1990), without use of address cycling to increase the available number of bits.

Communication with other backplanes is via link cards. Link cards are used as broadcast destinations for sending outputs to boards on other backplanes. Special backplanes with internal ring or crosspoint switch architectures can accept many links and act as communication hubs. The interconnection structure can be customized for the neural architecture being simulated by choices of backplane type, and by the placing of link cards. The connection bandwidth provided between two backplanes can therefore be varied as a function of the neural connectivity between neurons being simulated on each backplane. The brain has a number of "axon highways", thick long axon bundles that carry signals between lobes. These could be given dedicated links in any simulation.

Three factors combine to reduce the interconnection bandwidth required. First, neurons usually have large numbers of synapses and so the bandwidth of external outputs is much less than the processor I/O throughput. Second, the large memories connected to each processor mean that a large number of connections are between neurons simulated on the same processor. Finally, as discussed in Section 4.1, a neuron output need only be communicated if non-zero or differs from its previous value. Worst case communication requirements occur with randomly connected networks with low numbers of synapses per neuron. For competitive, sparsely connected networks with large numbers of synapses per neuron, the required communications bandwidths can be easily accommodated.

5.4 Support Boards

To save network states, and to support synapse swapping, a board that interfaces to fast, random-access, non-volatile storage devices is required. A suitable interface is the Small Computer Systems Interface (SCSI) bus (ANSI, 1986). The board need not have much memory as the locations of swapped axons on the storage device can be held in the memory on their home boards. The SCSI bus can also be used for communication to a host computer.

For an integrated speech recognition system, would require digital signal processing boards and analog to digital conversion boards. Other useful boards would include a digital to analog conversion board and a digital audio interface board.

Chapter 6

Neural Network Simulation on the Multicomputer

This chapter presents details of how brain-like and Backpropagation networks would be simulated on the multicomputer described in Chapter 5.

6.1 Simulation of Brain-like Neural Networks

This section will step-through the simulation of a generic brain-like neural network using the techniques of Chapter 4. Processing will be displayed in flow-charts, and data structures in the C language.

The simulation program for each simulation cluster (processor port) can be split into two processes and one subroutine. One process handles packets arriving from other clusters and from the board controlling the swap device, the other is the main simulation loop. A common subroutine of these processes queues active outputs and handles axon swapping.

Fig. 14 shows the packet handling process. The main packet it must handle is a packet of neuron outputs from other clusters. Such a packet could have the following form.

```
struct
{
      doubleword  time_stamp;
      word        source_cluster;
      struct
      {
            doubleword  neuron_num; /* in source cluster */
            word        output_val;
            word        average_output_val,delta_output_val;
      } outputs[];
} outputs_packet;
```

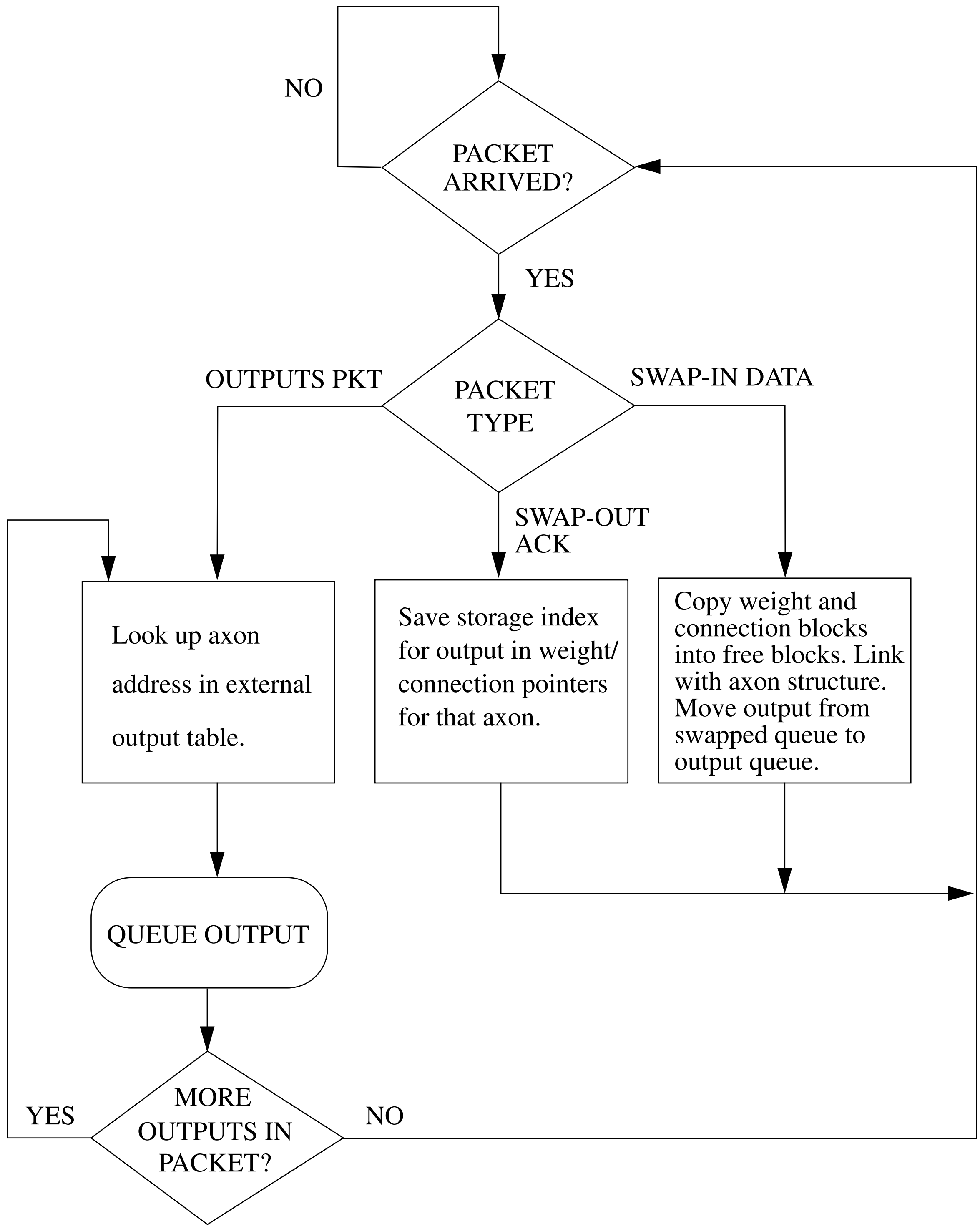


FIGURE 14. The output handler process.

To find the address of the axon structure for a given output, a conversion table must be searched. An instance of such a table should exist for each connecting cluster to reduce the search time.

```
struct      /* one table for each cluster providing input */
{
      doubleword  neuron_num;
      struct_axon *axon_address;    /* axon structure */
      boolean     swapped;          /* swapped out? */
} conversion_table[];
```

Networks with a range of axon delays would have a set of axon addresses for each output, each for a different delay, that must be placed on the appropriate output queues. If the connectivity of the network is regular, the need for search during the conversion process could be eliminated.

The axon structure could have the following form.

```
typedef     word  weight_block[ WEIGHT_BLOCK_SIZE ];
typedef     word  connection_block[ CONNECTION_BLOCK_SIZE];

struct
{
      word              delay; /* from output to synapses */
      weight_block      *weights[];
      connection_block  *connections[];
      word              num_wb, num_cb, num_connections;
      doubleword        next_lru, prev_lru;
} axon;
```

The use of connection and weight blocks allows neuron swapping without memory fragmentation, and so eliminates the need for memory compaction algorithms. The weight and connection block pointers can also be used to store the place on the swap media that the weight and connection blocks are stored when they are swapped-out. This process is shown as the response to a “SWAP-OUT ACK” packet.

The exact form of the weights and connections is not specified because they could be stored in various compressed forms. A general form of weight could be:

```
struct
{
      word  long_term_weight;
      word  short_term_weight;
      word  weight_modification_speed;
}
```

The modification speed parameter is important for simulations where multiple synapses between neurons are represented as a single synapse.

The LRU parameters are used to form a doubly-linked list which orders the axons according to their time of last use in synaptic update calculations. This list is managed by the output queuing and swapper subroutine (Fig. 15). This subroutine checks to see if a newly arrived or generated output is in memory, and, if it is not, immediately requests it from the storage device so that the weights and connections will arrive before the end of the time-step in which they will be used. Enough memory is freed for the weights and connections by swapping out weight and connection blocks of least recently used neurons. The output is then placed on a temporary queue till the data arrives from the storage device. Output axons that are in memory get placed on the final output queue for the simulation time-step at which the synaptic influences are to take effect.

The main processing loop is shown in Fig. 16. At the end of each time-step the outputs of all neurons are updated, and “active” axons are queued and dispatched to other clusters. Local neurons could be stored in the following structures.

```
struct
{
      word                    local_cluster_number;
      struct neuron_type      *neurons[];
}
```

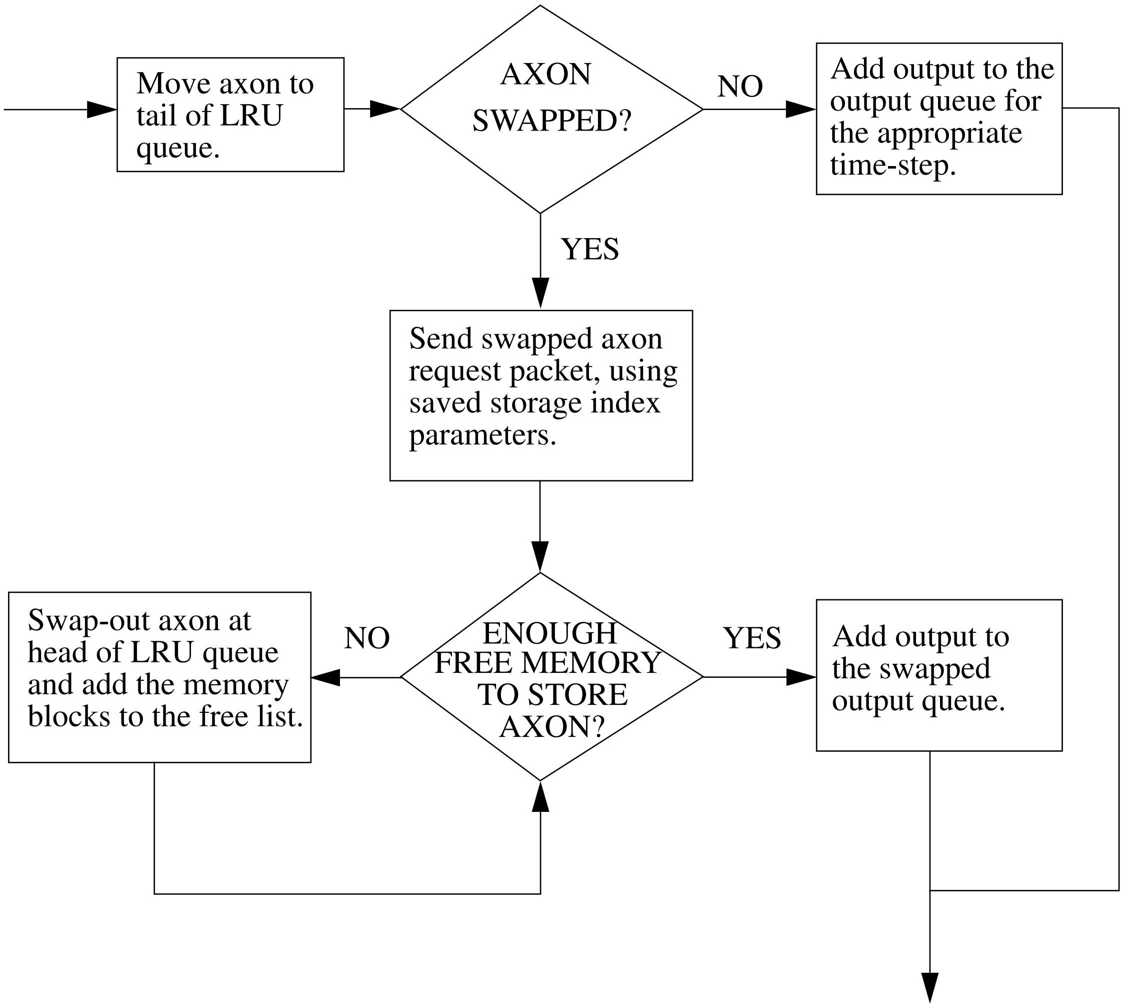


FIGURE 15. The output queuing and swapper subroutine.

```
struct
{
      struct local_neuron     *first_neuron_address;
      doubleword              first_partial_sum_address;
      word                    num_partial_sums;
      word                    num_internal_states;
      word                    partial_sum_decay_rates;
      word                    delta_w_threshold;
      word                    neuron_threshold;
} neuron_type;
```

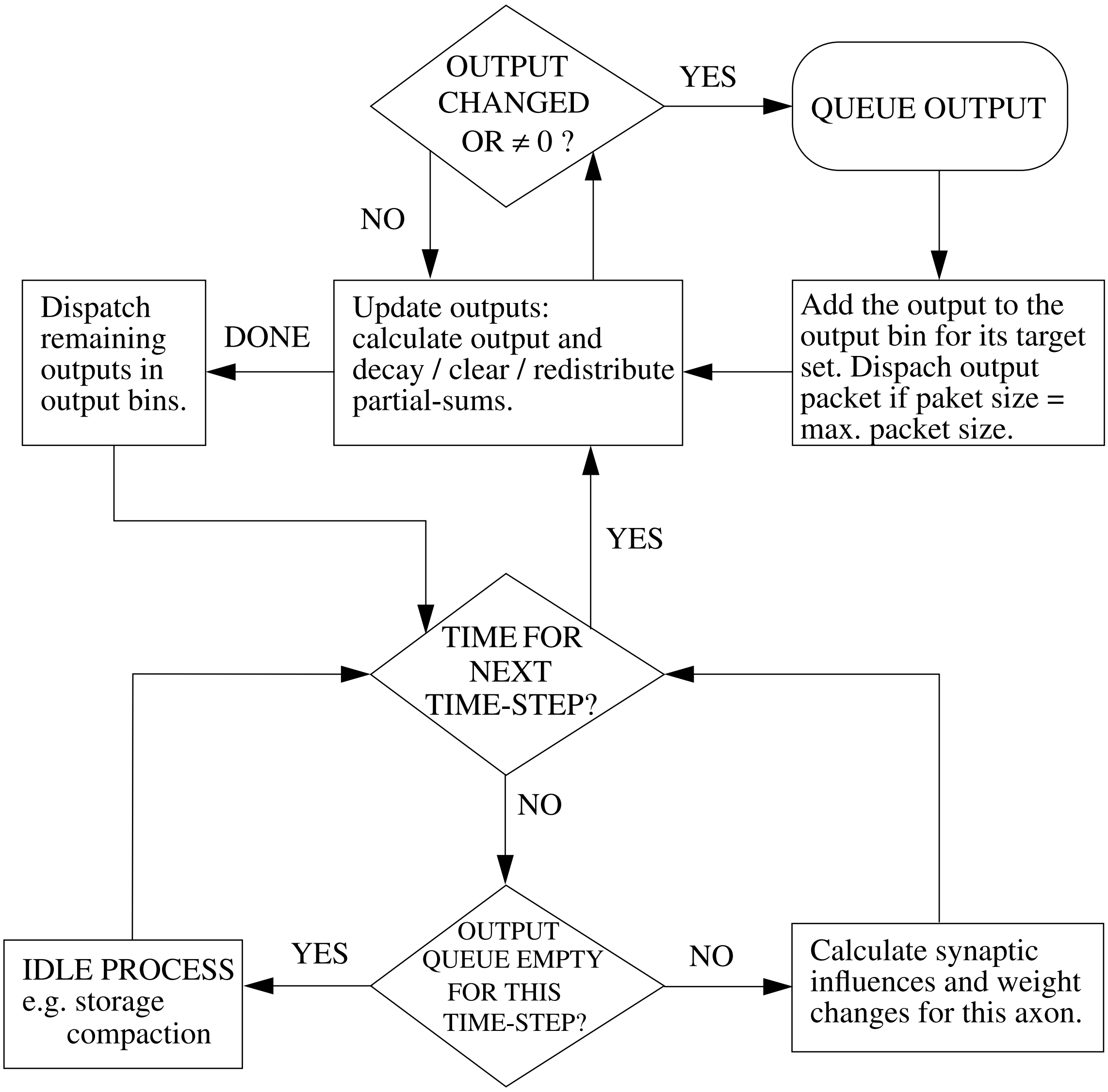


FIGURE 16. The output and synaptic update process.

```
struct
{
            struct axon       *axon_address;
            word              internal_states[];
            boolean           swapped;
            word              output_board_target_type;
            word              output, average_output;
} local_neuron;
```

The "output_board_target type" parameter holds a number representing the set of boards/clusters that this neuron's output communicates with.

The remainder of simulation time is used to calculate the synaptic influences of the axons in the output queue for the given time-step. The output update step is delayed as much as possible to allow time for packets carrying outputs and swapped data to arrive.

6.2 Simulation of Backpropagation Networks

To simulate completely connected Backpropagation networks on the multicomputer, two techniques mentioned in previous chapters can be employed. First, synaptic weights can be stored in two-dimensional matrices within each DRAM page. That is, a small sub-matrix of the weight matrix is mapped to a memory page so that row accesses will be sequential but column accesses will be in fixed jumps. With this scheme, page-mode can be used for both row and column access for sufficiently long run-lengths that average memory access times closely approximate the page-mode access time. Second, the method of Section 3.3.2 will be used to significantly reduce the number of times that inputs, and output errors, must be fetched. For the Datawave, 8 instructions can be executed for each external word fetched, so any technique that reduces the amount of external I/O will directly improve execution times.

A crosspoint switch, emulating a ring, would be the most suitable backplane choice. That is, horizontal slices of the weight matrix will be mapped to each processor.

The accepted benchmark for Backpropagation is the number of synapses that the simulator can update (including the feedforward, error propagation and weight update stages) every second. It will be assumed that weights, inputs, outputs, and errors, are all 12-bits (one word). The buffered input technique of Section 3.3.2 means that the simulation rate will be determined by synaptic weight I/O bandwidth. Each weight must be communicated four times (when using the batch weight update method) to update each synapse; once for the feedforward step, once for the error accumulation step and once each for fetching and storage in the weight update step. In each 16 *ns* period, one weight can be communicated over each of the 3 ports that have DRAM connected, yielding an update rate of approximately 50 million connection updates per second per simulation board.

Chapter 7

Conclusion

It is demonstrated that real-time simulation of very large neural networks at reasonable cost is possible if the hardware architecture is optimized for neural computation and unnecessary processing and storage is eliminated. It is significant that the memory for the storage of network parameters can be a major component in the cost of a simulation system. This, coupled with the fact that neural networks can self-organize to store information in a highly compressed form through feature sharing, suggests that neural networks could provide efficient solutions to many existing complex problems.

Even if neural network simulation does not turn out to be the most efficient way of implementing systems for tasks such as speech recognition, lessons learnt about mechanisms of speech perception from such simulations can be applied in more conventional systems. For example, psychophysics has taught us that people perceive low-level features of sensory input in a common way while differences emerge in the processing of more complex input. This suggests that much low-level neural processing is pre-wired or self-organizes into a standard set of feature detectors. The structure and nature of these features can be explored via neural simulation then implemented more efficiently as algorithms on signal processors. Statistical pattern recognition methods could be used to complete the system.

We hope to construct a small version of the multicomputer for use in research on real-time, large-vocabulary, word-level speech recognition. Even though the computer would be able to remove the impediment of insufficient simulation speed, there is still much work to do in the development of suitable network architectures and learning laws.

References

Abeles, M. *Local Cortical Circuits.* Springer-Verlag, Berlin, 1982.

Alkon, D. L., *et al.* Pattern-recognition by an artificial network derived from biologic neuronal systems. *Biological Cybernetics.* **62** (1990), 363-376.

Anderson, J. A., and Mozer, M. C. Categorization and selective neurons. In Hinton, G. E., and Anderson, J. A. (Eds.). *Parallel Models of Associative Memory.* Second Edition. Lawrence Erlbaum, New Jersey, 1989, pp. 251-276.

ANSI. *American National Standard for Small Computer System Interface (SCSI), X3.131-1986.* American National Standards Institute (ANSI), New York, 1986.

ANZA Plus VME. HNC Inc., 5501 Oberlin Drive, San Diego, CA 92121-1718, 1988.

Artola, A., *et al.* Different voltage-dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex. *Nature* **347** (1990), 69-72.

Barinaga, M. The mind revealed? *Science* **249** (1990), 856-858.

Bear, M. F., *et al.* A physiological basis for a theory of synapse modification. *Science* **237** (1987), 42-48.

Bienenstock, E. L. Cooper, L. N., and Munro P. W. Theory for the development of neuron specificity and binocular interaction in visual cortex. *Journal of Neuroscience* **2** (1982), 32-48.

Bregman, A. S. *Auditory Scene Analysis: the perceptual organization of sound.* The MIT Press, Cambridge, MA, 1990.

Brown T. H., *et al.* Hebbian synapses: biophysical mechanisms and algorithms. *Annual Review of Neuroscience* **13** (1990), 475-511.

Cauwenberghs, G., Neugebauer, C. F., and Yariv, A. An adaptive CMOS matrix-vector multiplier for large scale analog hardware neural network applications. *Proc. 1991 International Joint Conference on Neural Networks, Vol. 1*. IEEE, Piscataway, NJ, 1991, pp. 507-511.

Chapman, B., *et al.* Ocular dominance shift in kitten visual cortex caused by imbalance in retinal electrical activity. *Nature* **324**, 154-156.

Chernjavsky, A., and Moody, J. Note on the development of modularity in simple cortical models. In *Advances in Neural Information Processing Systems 2.* Morgan Kaufmann, San Mateo, CA, 1990, pp. 133-140.

Fahlman, S.E., and Lebiere C. The cascade-correlation learning architecture. In Touretzky D.S. (Ed.). *Advances in Neural Information Processing Systems 2.* Morgan Kaufmann, San Mateo, CA, 1990, pp. 524-532.

Foote, S. L., and Morrison, J. H. Extrathalamic modulation of cortical function. *Annual Review of Neuroscience* **10** (1987), 67-95.

Ghosh, J., and Hwang, K. Mapping neural networks onto message-passing multicomputers. *Journal of Parallel and Distributed Computing* **6** (1989), 291-330.

Gordon, P. C. Induction of rate-dependent processing by coarse-grained aspects of speech. *Perception and Psychophysics* **43** (1988), 137-146.

Graf, H. P., Janow, R., Henderson, D., and Lee, R. Reconfigurable neural net chip with 32k connections. In Lippmann, R. P., *et al.* (Eds.). *Advances in Neural Information Processing Systems 3.* Morgan Kaufmann, San Mateo, CA, 1991, pp. 1032-1038.

Grossberg, S. The adaptive self-organization of serial order in behavior: Speech, language, and motor control. In Schwab, E.C., and Nusbaum, H.C. (Eds.). *Perception of speech and visual form: Theoretical issues, models, and research.* Academic Press, New York, 1985.

Gupta, S. N., *et al.* Simulation of neural networks on a massively parallel computer (DAP-510) using sparse matrix techniques. *Proceedings Third Symposium on the Frontiers of Massively Parallel Computation.* IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 376-379.

Hagenzieker, M. P., van der Heijden, A. H. C., and Hagenaar, R. Time courses in visual-information processing: Some empirical evidence for inhibition. *Psychol. Res.* **52** (1990), 13-21.

Hammerstrom, D. A VLSI architecture for high-performance, low-cost on-chip learning. *International Joint Conference on Neural Networks, San Diego 1990.* IEEE Neural Networks Council, Ann Arbor, MI, 1990.

Hanson, S. J. Meiosis networks. In Touretzky D.S. (Ed.). *Advances in Neural Information Processing Systems 2.* Morgan Kaufmann, San Mateo, CA, 1990, pp. 533-541.

Hashemzadeh-Gargari, H., *et al.* Refining the temporal definition of an association at the neuronal level using long-term potentiation and long-term depression in the dentate gyrus. *Neuroscience Letters* **126** (1991), 37-40.

Hecht-Nielsen, R. *Neurocomputing.* Addison-Wesley, 1990.

Hinton, G. E., *et al.* Distributed representations. In *Parallel Distributed Processing, Vol. 1: Foundations.* The MIT Press, Cambridge, MA, 1986.

Hirsch, J. C., and Crepel, F. Use-dependent changes in synaptic efficacy in rat prefrontal neurons in vitro. *Journal of Physiology* **427** (1990), 31-49.

Holler, M. VLSI implementations of learning and memory systems. In Lippmann, R. P., *et al.* (Eds.). *Advances in Neural Information Processing Systems 3.* Morgan Kaufmann, San Mateo, CA, 1991, pp. 993-1000.

Hwang, K., *et al.* OMP: A RISC-based multiprocessor using orthogonal-access memories and multiple spanning buses. *Proceedings 1990 International Conference on Supercomputing*. Published as *ACM SIGARCH Computer Architecture News* **18,** 3 (Sept. 1990), 7-22.

Jacobs, K. M., and Donoghue J. P. Reshaping the cortical motor map by unmasking latent intracortical connections. *Science* **251** (1990), 944-947.

Johnson, K. Contrast and normalization in vowel perception. *Journal of Phonetics* **18** (1990), 229-254.

Jones, S. J., and van der Poel, J. C. Binaural interaction in the brain-stem auditory evoked potential: Evidence for a delay line coincidence detection mechanism. *Electroencephalography and clinical Neurophysiology* **77** (1990), 214-224.

Kohonen, T. *Self-Organization and Associative Memory.* Second Edition, Springer-Verlag, Berlin, 1988.

Larson, J., and Lynch, G. Theta pattern stimulation and the induction of LTP: The sequence in which synapses are stimulated determines the degree to which they potentiate. *Brain Research* **489** (1989), 49-58.

Mason, A., Nicoll, A., and Stratford, K. Synaptic transmission between individual pyramidal neurons of the rat visual cortex *in vitro*. *Journal of Neuroscience* **11**, 1 (Jan. 1991), 72-84.

Mel, B. W., and Koch, C. Sigma-pi learning: on radial basis functions and cortical associative learning. In Touretzky D.S. (Ed.). *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, San Mateo, CA, 1990, pp. 524-532.

Miller, K. D., and Stryker, M. P. The development of ocular dominance columns: mechanisms and models. In Hanson, S. J., and Olson, C. R. (Eds.). *Connectionist Modeling and Brain Function: The Developing Interface*. MIT Press, Cambridge, MA, 1990, pp. 255-350.

Misra, M., and Kumar, V. K. P. Massive memory organizations for implementing neural networks. *Proceedings 1990 International Conference on Pattern Recogntion, Vol. II.* IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 259-264.

Miyashita, Y. Associative representation of the long-term memory in the neurons of the primate temporal cortex. In Iwai, E., and Mishkin, M. (Eds.). *Vision, Memory and the Temporal Lobe*. Elsevier, NY, 1990, pp. 75-87.

Mountcastle V.B. An organizing principle for cerebral function: the unit module and the distributed system. In Edelman, G. M., and Mountcastle, V. B. (Eds.). *The Mindful Brain*. The MIT Press, Cambridge, MA, 1978, pp. 7-50.

Pisoni, D. B., and Luce, P. A. Trading relations, acoustic cue integration, and context effects in speech perception. In Schouten, M. E. H (Ed.). *Psychophysics of Speech Perception*. Kluwer Academic Publishers, Boston, MA, 1987, pp. 155-172.

Rosenberg, C. R., and Blelloch, G.E., An implementation of network learning representations by error propagation. *Tech. Report. No. RL87-3, Thinking Machines Corporation.* Cambridge, MA, 1986.

Rumelhart, D. E., *et al.* Learning internal representations by error propagation. In *Parallel Distributed Processing, Vol. 1: Foundations.* The MIT Press, Cambridge, MA, 1986, pp. 318-362.

Schmidt, U., and Caesar, K. Datawave: A single-chip multiprocessor for video applications. *IEEE Micro,* June 1991, pp. 22-25, 88-94.

Schultz, P. E., and Johnston, D. Evidence for differing mechanisms of short- and long-term potentiation expression. *Society for Neuroscience Abstracts* **17** (1991), 386.

Seldon, H. L. The anatomy of speech perception. In Peters, A., and Jones E. G. (Eds.). *Cerebral Cortex (Vol. 4).* Plenum Press, New York, 1985, pp. 273-327.

Sporns, O., Tononi, G., and Edelman, G. M. Modeling perceptual grouping and figure-ground segregation by means of active reentrant connections. *Proceedings of the National Academy of Science USA* **88** (1991), 129-133.

Sutton, R. S., and Barto, A. G. Toward a modern theory of adaptive networks: Expectation and prediction. *Psych. Rev.* **88** (1981), 135-170.

Suzuki, Y., and Atlas, L. E. A study of regular architectures for digital implementation of neural networks. *IEEE Int. Symposium on Circuits and Systems proceedings (1989)*. IEEE, New York, 1989, pp. 82-85.

Swindale, N. V. Is the cerebral cortex modular? *Trends In Neuroscience* **13**, 12 (Dec. 1990), 487-492.

Tesauro, G. Neural models of classical conditioning: A theoretical viewpoint. In Hanson, S. J., and Olson, C. R. (Eds.). *Connectionist Modeling and Brain Function: The Developing Interface*. MIT Press, Cambridge, MA, 1990, pp. 74-104.

Tomboulian, S. J. Introduction to a system for implementing neural networks on SIMD architectures. In *Proceedings of the IEEE Conference on Neural Information Processing Systems*. IEEE, NY, 1987.

Tsumoto, T. Long-term potentiation and depression in the cerebral neocortex. *Japanese Journal of Physiology* **40** (1990), 573-593.

van Essen, D. C., *et al.* Probing the primate visual cortex: Pathways and perspectives. In Valberg, A., and Lee, B. B. (Eds.). *From Pigments to Perception*. Plenum Press, New York, 1991, pp. 227-237.

Wilson, M. A., and Bower, J. M. The simulation of large-scale neural networks. In Koch, C., and Segev, I. (Eds.). *Methods in Neuronal Modelling*. The MIT Press, Cambridge, MA, 1989, pp. 291-334.