NEURAL CHIPS

NEURAL NETWORK CONSTRAINTS ON NEURAL CHIP DESIGN

Computer algorithms often perform poorly on tasks such as signal recognition, which humans perform readily. In particular, speech and image recognition generally require enormous processing power (1). Neural networks attempt to match the capabilities of humans by interconnecting networks of circuit elements that mimic the capabilities of brain cells. These artificial neural networks can be trained to identify and predict complex signals. Neural circuits are hardware implementations of neural networks; *neural chips* implement neural circuits as integrated circuits (IC) using microelectronics technology. Some basic concepts in neural networks are required to appreciate the issues involved in designing neural circuits and implementing them as neural chips.

Most artificial neural circuits are direct descendents of Rosenblatt's perceptron circuits (2). Their behavior is a greatly simplified model of a biological neuron in which the output of neuron i, x_i, is a nonlinear function of its inputs y_i

$$x_i = f(y_i) \tag{1}$$

$$y_i = \Sigma_j w_{i,j} x_j - \theta_i \tag{2}$$

$\{x_j\}$ is the set of outputs of other neurons connected as inputs to neuron i through a set of "synapse" weights $\{w_{ij}\}$. A threshold value θ_i is subtracted from this sum. Equations (1) and (2) define what is meant by neural circuits, neurons, and synapses in artificial neural networks. The neuron function $x_i = f(y_i)$ is typically a saturating function such as the sigmoid $x = (1 + e^{-y})^{-1}$. Similar functions such as $y = \tanh x$ may be used as well. If the neuron outputs are voltages and the weights are conductances $w_{ij} = G_{ij}$, then y_i is a sum of currents.

Neural networks differ primarily in the way the neurons are interconnected. In the one-layer *Hopfield network* the output of each neuron is fed back to the inputs of all other neurons. In *feedforward multilayer perceptron* networks, the outputs of neurons in one layer become the inputs to neurons in the next layer but not to later layers. The existence of "hidden" layers between the input and output layers allow multilayer perceptron networks to handle more complex classification problems. Historically, the interest in neural networks was satisfied for many years by the realization that one-layer perceptron networks could not reproduce the behavior of an exclusive-OR (XOR) Boolean function (3). The realization that multilayer perceptron networks could reproduce XOR behavior helped rekindle interest in neural circuits.

Different neural networks also differ in their applications. For example, Hopfield nets can be used as an associative (content-addressable) memory that can distinguish trained patterns from noisy inputs. The net initially stores a set of multibit input patterns by using a simple learning rule that increases the weights of the connections between neurons which are simultaneously active. Hopfield (4) showed that the stored patterns are stable and that, when a noisy example pattern is presented as an input, the network outputs will converge to the correct example pattern; however, there are limitations. The number of patterns that can be stored and retrieved successfully is limited by the size of the network and the closeness of example bit patterns in relation to the amount of bit-switching noise in an input pattern. This report by Hopfield in 1982 revived interest in neural networks.

An advantage of neural networks is that useful behavior can be *trained* or *learned* rather than *programmed,* as would be the case for deterministic algorithms executed on a digital computer. In training (supervised learning), the desired outputs of a neural network are fixed for a set of training data (e.g., handwritten characters) applied to the network inputs. The network learns by adjusting synapse weights so that network outputs match the desired outputs.

Interest in multilayer perceptron networks was rekindled by the discovery of effective procedures for adjusting hidden-layer weights. Backpropagation (of errors) popularized by Rumelhart, Hinton, and Williams (5) is the best-known procedure. The backpropagation training algorithm uses an iterative, gradient-search technique to minimize the mean-square error between the actual (x_i) and desired (d_i) outputs of neu-

ron i. The weights connecting neuron i to the neuron outputs from the preceding layer are changed by an amount

$$\Delta w_{ij} = w_{ij}(t+1) - w_{ij}(t) = \eta e_i f_i'(y_i) x_j \tag{3}$$

where

$$e_i = x_i - d_i \tag{4}$$

for a neuron on the output layer; η is called the *learning rate* and is chosen by the user. A small η (≈ 0.001) changes weights slowly. If η is too large (≈ 10), the weights may change so rapidly that the errors do not decrease (6); $f_i'(y_i)$ is the derivative of the neuron function with respect to its input. The derivative of the sigmoid function has a simple form, $f'(y) = x(1 - x)$.

Calculating the error for an output neuron is quite straightforward as both the desired output d_i and the actual calculated output x_i are known. For neurons in the hidden layers, the errors are calculated by backpropagation of errors from the succeeding layer. Thus, for a hidden layer

$$e_i = \Sigma_j e_j f'(y_j) w_{ij} \tag{5}$$

where the sum is only taken over neurons in the *succeeding* layer. The corresponding weight change for the synapse inputs to neuron i is given by Eq. (4).

Neural networks have mostly been implemented by software models. These have the advantage of high precision for the required additions and multiplications. However, because the computations of a highly parallel, connectionist architecture are being simulated by serial computations, software implementations of neural network behavior are inherently much slower than hardware implementations.

Analog neural circuits built with discrete components in the 1960s (7) already demonstrated neural network capabilities. However, they were limited to the capabilities of networks with only a few neurons by the size and cost of components. At present, optoelectronic (8), digital, and analog implementations of neural networks are being studied. This article focuses on the obstacles to implementation of digital and analog electronic neural circuits on IC chips with emphasis on analog neural circuits.

Equation (2) indicates that neural circuits *must perform sums of products efficiently* to be effective. Since the interconnected synapses that form the products are by far the most numerous component in neural networks, the *area-efficient implementation of synapses* is essential. In analog circuits, Ohm's law allows a single conductor to multiply a voltage to produce a current, and currents are summed at a node by Kirchhoff's current law (9). The simplicity of this implementation is seductively attractive, but it is complicated by the *need to adjust and store synapse weights* if the neural circuit is to learn new behavior. Another complication is the need for synapses to produce *negative as well as positive neuron inputs.*

The *computational precision* required in the learning and execution phases of neural circuits is another key factor in their design. Interestingly, greater precision seems to be required for learning than execution. This was investigated in a software simulation of a neural network with 60,000 synapse weights trained to recognize handwritten digits (9). Network performance was unchanged if weight resolution was reduced to 5 bits (3.1%) and neuron outputs to 3 bits (12.5%), compared with a full-precision weight resolution of 32 bits (2.3×10^{-8}%) for both. However, the full 32-bit precision was required for training with the backpropagation algorithm. Note that 32-bit precision would be extraordinarily high for analog instrumentation; 16-bit resolution, which is at the upper end of conventional digital-to-analog or analog-to-digital converters (DACs or ADCs), corresponds to a precision of 1.5×10^{-3}%.

The requirements of efficient sums of products, area-efficient synapse implementation, adjustment and storage of synapse weights, production of negative and positive neuron inputs, and sufficient computational precision all constrain the effectiveness with which neural circuits can be implemented as neural chips. For example, a linear, four-quadrant, analog multiplier requires more MOSFETs and increases the size of a synapse cell. Correspondingly, this reduces the number of synapses that can be implemented on a chip. Replacing analog multiplication by digital multiplication can increase precision but requires a large increase in area. This area can be reduced significantly if parallel multiplication is replaced by serial multiplication; however, it takes longer to execute operations.

Choosing the right balance of such trade-offs is the key for successful implementation of neural networks. Some of these trade-offs for representative implementations of neural chips are quantified in the following. The remarkable diversity of approaches to neural chip design is illustrated by the discussion of several representative circuits. Circuit and chip areas as well as processing speeds are given to allow comparison of alternative circuit implementations. Since circuit size depends on technology, each technology is characterized by its minimum feature size.

DIGITAL IMPLEMENTATIONS OF NEURAL CHIPS

Kolinummi et al. (10) review many digital implementations of neural networks that have been reported in the literature. These digital implementations may be realized at several levels. A neurocomputer may consist of many parallel processing units and other interface circuitry, an acceleration board, which can be used with a host computer, or a standalone chip. Purely digital systems are flexible and support a wide range of neural algorithms. However, because digital implementations are area-hungry, the size of networks that can be implemented on a chip is limited. Digital implementations may suffer from limited resolution because the chip area is proportional to the wordlength. Different implementations from the literature report precisions that vary between 8 and 16 bits. The maximum number of parallel processing units on one chip is generally on the order of several hundred or less, with several chips on several boards connected together for one system to realize large networks. Digital implementations of on-chip learning have been more successful because they do not have mismatch and nonideality problems. However, the area required for on-chip training is even larger compared to an analog network. Digital realizations of radial basis functions and self-organizing maps have been more successful up to now.

A good example of a digital implementation of a neural network is presented by Beichter et al. (11) who describe the

architecture and design of a VLSI array processor chip (MA16) at the heart of Siemens' 16-bit SYNAPSE neurocomputer. Although digital designs may be more flexible than analog designs, which have to be tailored to specific algorithms, their design illustrates the compromises required in the digital implementation of a neural network.

Multiply-accumulate operations such as $\Sigma_j\ w_{ij}x_{jp}$ for weights and inputs from Eq. (2), where p designates one of a set of P patterns to be recognized, must be implemented efficiently for high neural chip performance. Note that this expression can be regarded as a matrix-matrix Multiply-ACcumulate (MAC) since we have to consider several inputs $\{j\}$ for several patterns $\{p\}$. In addition to pattern recognition, learning and weight update require similar operations.

Implementation requires a compromise or trade-off between weight storage and the number of MAC chains on an MA16 chip. More MAC chains on a chip increase processing speed but take up area, which could be used for weight storage. If weights are stored on-chip, the processor area must be balanced with memory area; if they are stored off-chip, there must be sufficient memory bandwidth. This need for a balance between processor speed and memory bandwidth (Amdahl's rule) is a classic trade-off in computer design. Because they decided to store weights off-chip in inexpensive DRAM chips, the MA16 chip can be devoted to signal processing. This introduced two constraints on memory bandwidth: the number of I/O pins available for data transfer, and DRAM cycle times.

Each MA16 chip contains four systolic MAC chains. Each chain accumulates the sum of four multiplications, providing an array of 16 16 $\times$ 16-bit multipliers. In addition, each chain contains a scaling multiplier and accumulator along with other circuits that facilitate other essential neural algorithm operations. For pattern recognition each MAC chain computes a 4 $\times$ 4 matrix multiplication of 16-bit weights and inputs in 16 clock cycles. To achieve systolic computation rates a pipelined, 16 $\times$ 16-bit array multiplier was implemented. Each of its 16 $\times$ 16 = 256 1-bit multiplier cells contains a 24-MOSFET full adder, a NAND/NOR partial product bit gate, an inverter, and four latches.

Data transfer to the MAC chain takes place in 16-word blocks corresponding to a 4 $\times$ 4 submatrix. Using 4 $\times$ 16 = 64 pins and a clock rate of 40 MHz, each MA16 chip achieves a memory bandwidth of 2.56 Gbps. During the 16 clock cycles for 4 $\times$ 4 matrix multiplication in pattern recognition, the MAC weight buffer sends its 16 words to the MAC chain in a cyclical fashion. The top multiplier receives the first, fifth, ninth and thirteenth word and keeps this input for four clock cycles to multiply it with four words from the input-data buffer. Each buffer is a dual-port memory containing 2 $\times$ 16 = 32 16-bit words. The dual port allows 16 new words to be brought into the buffer during the 16 cycles while 16 words are entering the MAC chain. A third 32 $\times$ 16-bit buffer in the MAC chain stores and transfers the result of the 4 $\times$ 4 matrix multiplication.

Each 1-bit buffer cell was implemented as a three-transistor dynamic memory cell with nondestructive readout, which occupied about a third of the area of an eight-transistor static cell. In 1 μm CMOS technology, a total area of about 1 mm^2 is occupied by memory cells for each MAC chain, neglecting the area consumed by local control and power lines. This is relatively small compared with an area of about 160/4 = 40 mm^2 for each MAC chain and associated circuitry.

Burr (12), in reviewing the design of neurochips, notes that the performance requirements of a neural network can be displayed on a plot with axes corresponding to storage (connections) and processing speed (connections per second). In this plot, speech processing requires processing 10^5 to 10^6 connections (C) at speeds of 10^7 to 10^{10} connections per second (CPS) or 0.01 to 10 GCPS. The relative areas assigned to storage and processing may be expressed in connections per processor (CPP). Operating at 50 MHz, a single MA16 chip with 4 MAC chains each containing four multipliers can process information at a rate of 4 $\times$ 4 $\times$ 50 MHz = 800 multiply-accumulates per second = 800 MCPS. A SYNAPSE1 processor containing 8 MA16 chips and operating at 40 MHz has a processing rate of 5.12 GCPS. With 128 million 16-bit words of memory (connections), SYNAPSE1 should be an effective speech processor. With 4 $\times$ 4 $\times$ 8 = 128 multiply-accumulate processors, CPP = 10^6 for SYNAPSE1 and CPS/C = 5.12 G/128 M = 40. This is comparable to the values cited in Ref. 12 for biological nets with CPP = 10^7 and CPS/C = 10.

NONIDEALITY CONSTRAINTS ON ANALOG NEURAL CHIP DESIGN

Departures from ideal circuit operation make implementation particularly difficult for analog circuits. The effect of *nonideal* analog neural circuit behavior on backpropagation learning has been examined by Frye and co-workers (13). Although their circuit follows the simple approach of summing currents controlled by conductive synapses, their synapse is a photoconductor whose conductance (weight) is controlled by illumination, making it an optoelectronic neural circuit. Component variations are one nonideality affecting the behavior of neural chips. In their case, the photoconductive elements had an overall variation of $\pm$10% under uniform illumination; this was increased to about $\pm$30% by nonuniformities and misalignments of the optical illuminators.

They designed and built a layered feedforward network with three analog inputs, ten hidden neurons, and two output neurons. Each neuron was composed of four transconductance amplifiers. Back-to-back diodes in the feedback path of the final stage gave a sigmoidally shaped response. The strength of the synaptic connections was controlled by varying the length of a bar of light from 0 to 240 ($\leq 2^8$) pixels; this introduced weight quantization.

An advantage of neural networks is *robustness*; behavior should be affected little by component variations (missing synapses, etc.). Their network was trained to emulate an unknown system, in their case the ballistic trajectory of a projectile. This is common in control applications where the neural network emulates the behavior of the "plant" to be controlled. An ideal software model of the neural circuit hardware, which had learned from the same set of training examples, was run in parallel with the hardware to test the effect of component variation on the outputs. Although the hardware needed more training, it reached a comparable steady-state error that was less than 4% for components with more than 30% variation. In another experiment, the loss of 40% of the hidden layer circuitry only slightly increased the error to 5.5%, illustrating

the robust ability of neural circuits to adapt to severe component variations.

When the network was trained in signal prediction for a two-dimensional chaotic relation, it was discovered that while a learning rate η greater than 0.1 resulted in unstable, divergent outputs, smaller values of η gave stable outputs that were no better than random guesses. This was due to the effect of *weight quantization* on backpropagation learning. Since backpropagation learning involves gradient descent down an error surface in weight space, accurate differentiation requires this surface to be a smooth, continuous function of weights. If the weight change Δw calculated from Eq. (3) is less than one quantum, the weight remains unchanged. Overcoming this required accurate off-line calculations of weight changes that were accumulated during learning until Δw exceeded a quantum.

Several nonidealities (noise, weight quantization, and dynamic range or limited maximum connection strength) were studied by simulation. Simulations indicated that output errors were lower for low noise levels but became comparable for rms input or output noise levels greater than about 5%. Once training was established, simulations also indicated that quantization error was similar to noise in its effects. Residual errors only became comparable to weight increments (quanta) greater than about 5% ($\geq$ 4-bit quantization). Errors increased rapidly above a plateau when maximum synaptic weights were less than one, indicating that limited dynamic range was the most important limit to hardware performance.

Although the optoelectronic approach in Ref. 13 produces relatively linear multipliers, practical analog MOSFET multipliers are more complex circuits in order to improve multiplier linearity, dynamic range, and weight accuracy. Kub and co-workers (14) describe programmable analog multipliers in which weights are stored dynamically on the capacitances of MOSFET gates. The weights are refreshed periodically through pass transistors from values V_w stored off-chip in a digital memory and accessed through a digital-to-analog converter. A differential-pair two-quadrant multiplier cell can be built with three MOSFETs, to which two pass transistors must be added for weight refreshment. This circuit has the advantage of not requiring a current-summing amplifier with a low input impedance. Linear multiplication requires that the current-source transistor, controlled by the input voltage V_x, operate in the saturation region. Thus, this current has a problem with linearity at *low* values of input voltage.

As shown in Fig. 1 a four-quadrant six-MOSFET modified Gilbert multiplier cell can be realized by coupling two of these differential pairs with reversed polarities. For a supply voltage $V_{DD} = 10\ V$, and appropriate reference biases for V_x and V_w, this circuit has a total harmonic distortion (THD) less than 2% for V_x or $V_w \leq 1\ V_{pp}$. By adding another MOSFET as a current source for the two differential pairs, this circuit becomes the Gilbert multiplier shown in Fig. 2. This increases the dynamic range for linear operation somewhat with less than about 1.5% THD with V_x or $V_w \leq 1.5\ V$ peak-to-peak. Thus, the use of fewer MOSFETs in a synapse multiplier cell reduces cell area and allows more neural circuits to be implemented on a chip. However, multipliers implemented with fewer transistors are more nonlinear.

Capacitor storage of weights is limited by the need to refresh their charge. By using a balanced, double-capacitor arrangement it was possible to decrease weight decay by a factor of 50 from 30 to 0.6 mV/s at room temperature. This decrease occurs because voltage decays at the two storage sites tend to cancel when the difference between gate voltages is V_w. This circuit was implemented as a 32×32 vector-matrix multiplier chip using the MOSIS 3 μm p-well process. To reduce the number of input/output pads on the chip, the chip contained analog serial-to-parallel and parallel-to-serial multiplexers for input and output vectors, respectively. Their chip could be cascaded with *off-chip* amplifiers to form multilevel neural circuits.

ANALOG IMPLEMENTATIONS OF NEURAL CHIPS

Although more nonlinear synapses are smaller, the standard backpropagation algorithm fails to converge for nonlinear synapses because incorrect derivatives of synapse functions become critical in the end phase of learning when some weights are driven toward saturation. To overcome this problem in the implementation of neural chips, Lont and Guggenbühl (15) reformulated the backpropagation algorithm to allow the use of simpler, three-transistor synapses with reduced linearity. These synapses perform two-quadrant multiplication and produce a differential current that is converted to a bipolar current at the neuron input. They embodied a three-layer network with 18 neurons and 161 synapses on a small 3.8 mm^2 chip using 3 μm CMOS technology. The area of their synapse cell was 3564 μm^2, allowing a high density of 142 synapses/mm^2. Their soma circuit (the heart of the neuron) had an area of 9504 μm^2.

Once again, synapse weights were stored in an off-chip digital memory and loaded onto storage capacitors by a digital-to-analog converter. With a 5 V supply voltage, the weight-decay rate was 0.6 mV/ms. Clock feedthrough was the largest source of error (35 mV for zero switching times), which was comparable to the static offset voltages of the differential pairs.

The chip was tested with a simple pattern-recognition application. When initialized with a weight set having a normalized mean square error (mse) of 0.001, the chip performed correctly with mse = 0.012 after training (11% linear error). However, the chip failed to learn, mse $\geq$ 0.185 (43% linear error), when initialized with a random weight set. This was because the 8-bit resolution of DACs and ADCs used in their prototype was insufficient. A resolution of 15 or 16 bits is required for inputs/outputs and weights, respectively, during the learning process.

Hollis and Paulos (16) considered the use of MOS analog multipliers in *Hopfield-style* neural networks. Their implementation uses digitally controlled current sources to store weights together with two-quadrant analog multipliers. A set of parallel binary-weighted current sources is attached to a differential, current-steering multiplier transistor pair. A 6-bit programmable weight requires W/L ratios from 4 to 1/8. Negative weights can be achieved by reversing the polarity of the differential input signal. However, this requires four switches in front of each multiplier pair. An important difference of their circuit from the conventional Hopfield model is that multiplier nonlinearities limit the current into each multiplier pair, *before* the product-term currents are summed. Despite this difference, simulations indicated comparable be-

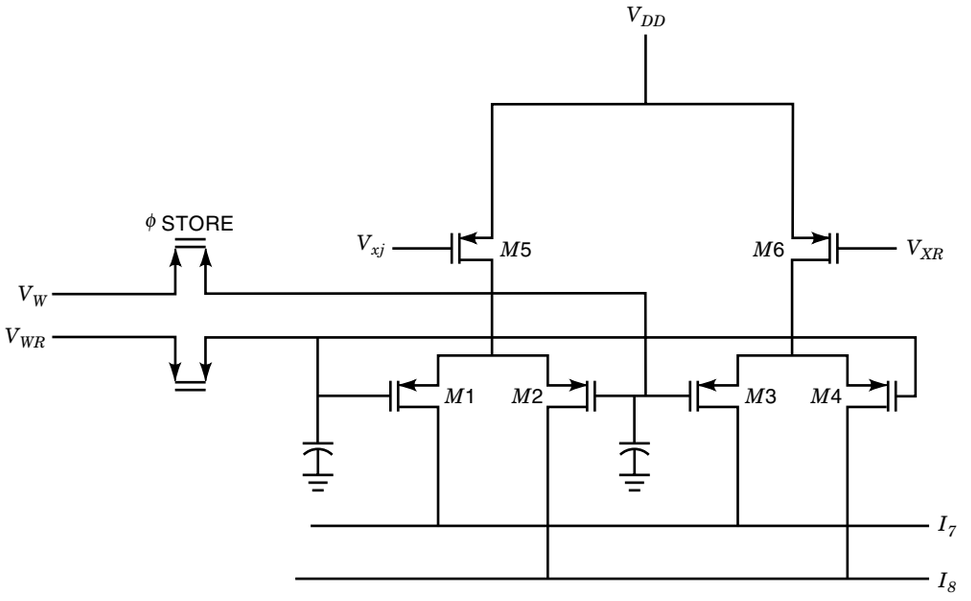

Figure 1. Modified Gilbert programmable analog multiplier (14). This is a simplified (six-MOSFET) multiplier which requires two differential pairs for linear and bipolar (four-quadrant) multiplication. Additional capacitors and transistors are required for weight storage.

havior, converging to optimal solutions for simple problems. For more difficult problems convergence was limited by neuron gain, as high gain is required to force a low-preference neuron to the desired final state. Once the neuron model was verified, simulations were performed using *quantized* weights to determine the minimum resolution that would not sacrifice solution quality. It was found that a 6-bit plus sign implementation of weights gave sufficient resolution for a broad range of applications. This probably reflects lower weight resolution requirements for Hopfield networks.

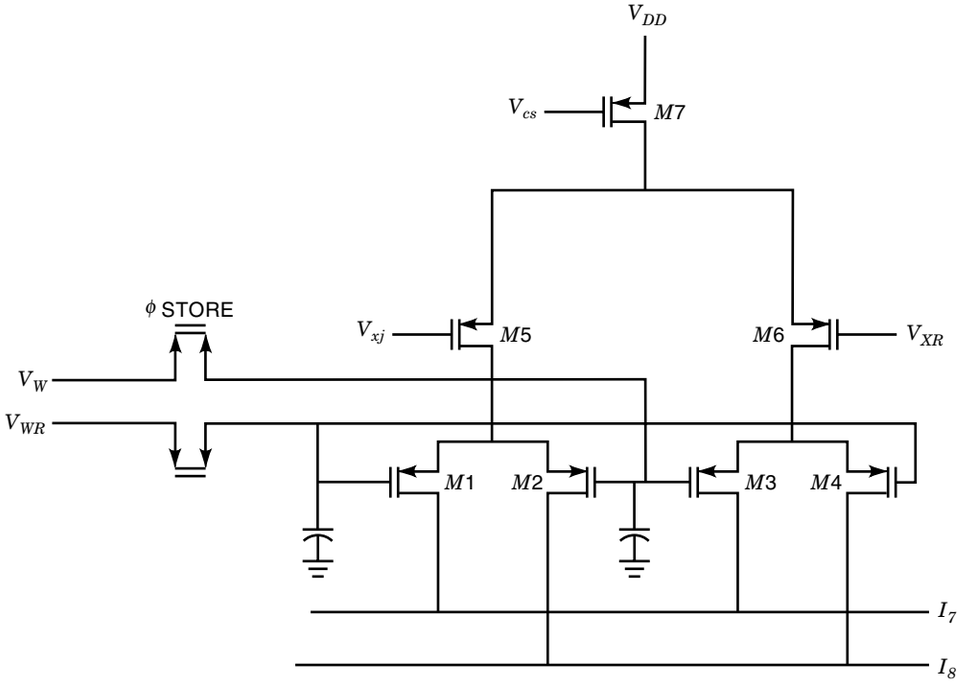

Figure 2. Gilbert programmable analog multiplier (14). This is the classic analog multiplier circuit implemented in CMOS technology. The additional transistors extend the dynamic range for linear multiplication.

It was possible to fabricate a fully connected, seven-neuron, 49-synapse Hopfield network using MOSIS 3 μm technology on a 6 mm^2 chip. Each synapse multiplier cell required 60,000 μm^2 to implement one 6-bit (plus sign) connection weight. The actual analog multiplier occupied only 40% of the cell area. To compensate for variations in chip fabrication, the MSB weight was implemented with multiple replicas of a unit current source MOSFET. It was estimated that a fully connected 81-neuron Hopfield network could be fabricated on a 1 cm^2 chip by using 1.25 μm design rules.

Masa and co-workers (17) have described a high-speed analog neural coprocessor, which classifies high-energy particle data taken at high speed (10^{16} bytes/s) from synchrotron detectors. This coprocessor operates at 20 MHz using a 2.5 μm CMOS technology and evaluates a 70 $\times$ 4 $\times$ 1 *feedforward* network within one clock cycle. This corresponds to a signal processing rate of 284 multiply-accumulates in 50 ns or 5.7 Gc/s. Their circuit uses a CMOS inverter driving a low-impedance load as a unity-weight synapse that converts voltage to current. Similar to the approach of Hollis and Paulos, variable weights were obtained by switching parallel inverters with different gate widths in or out of the circuit. A 4-bit (plus sign) synapse required 6000 μm^2, allowing the feedforward network to fit on a 26 mm^2 chip. Since their application is fixed, the high accuracy required for on-chip learning is avoided; their chip could be mask-programmed during fabrication.

It is interesting to compare their analog implementation of multiplication-accumulation with a purely digital implementation using a similar 2.5 μm CMOS technology. Their approach computes a 4 $\times$ 4-bit multiply-accumulate in 50 ns with a multiplier area of 6000 μm^2. To maximize speed for digital signal processing, Hatamian and Cash (18) developed a parallel, pipelined CMOS multiplier that can compute an 8 $\times$ 8-bit multiply-accumulate in 14 ns. However, their multiplier requires 12.5 mm^2. Scaling back the 8 $\times$ 8-bit multiplier area by a factor of four to match the 4 $\times$ 4-bit multiplier, the ratio of areas is 520 while the ratio of multiply-accumulate times is only 3.6. Analog multiplication requires much less area for comparable speed.

The floating-gate technology used in flash memories allows weights to be stored on single MOSFETs without requiring refresh. Intel used this technology to develop an Electically Trainable Analog Neural Network (ETANN) chip (19). When introduced, it was certainly the most complex chip developed for neural networks with 64 neurons and two 80 $\times$ 64 synapse arrays. Two groups of 80-wide array inputs corresponded to 64-dimensional vector inputs plus 16 fixed-bias inputs. The ETANN chip implements the inner product of an input vector with stored weight vectors, generating 64 scalars that are passed through 64 sigmoid neuron functions to the outputs. One synapse array was used for inputs, the other for feedback; either or both of the arrays could drive the neuron responses. Fully parallel processing yielded performance exceeding two billion multiply-accumulate operations (connections) per second (2 Gc/s). Typical resolution of the analog inputs and outputs was $\geq$6 bits. To change weights an individual synapse is addressed and pulses of different widths or voltages are applied externally to two chip pins to shift the threshold voltage of an individual floating-gate MOSFET. Weights are changed with 8-bit (0.4%) resolution, one synapse at a time. The physical limit on resolution (20 mV) was set by the ability of the neuron circuits to detect a change. This corresponded to a charge difference of about 2.5 million electrons on a floating gate. A 6-MOSFET differential, 4-quadrant multiplier synapse was implemented with a 2009 μm^2 cell size in a 5 V, 1 μm CMOS EEPROM technology (20).

IMPLEMENTING ON-CHIP LEARNING

All of the implementations discussed so far have involved off-chip learning or training to avoid the complexity and area costs associated with implementing on-chip learning. In many cases, chip nonidealities make it difficult to transfer off-chip learning on chip. Dündar et al. (21) studied the extent to which an on-chip synapse with quadratic nonlinearities degrades neural network performance when the network is trained off-chip with ideal, linear synapses implemented in software. Simulations indicated substantial deterioration of neural network performance, which could largely be recovered if the off-chip synapses matched the on-chip synapse nonlinearity. Bayraktaroglu et al. (22) perform the training on a SPICE-like circuit simulator and download the weights thus obtained to the circuit itself. This approach seems to circumvent most of the nonlinearity and loading problems observed. Edwards and Murray (23) classify what we have called limited resolution into two groups, namely, imprecision and inaccuracy. Through an algorithm that they have developed, they perform training, the results of which are much more fault-tolerant to quantization or errors in weights.

Training methods for neural chips can be studied under three headings: off-chip training; chip-in-the-loop training; and on-chip training. At present, off-chip training has mostly been abandoned. Most implementations in the literature employ chip-in-the-loop training, where the training algorithm runs on a host computer that collects the data from the chip. The approach in Ref. 22 can be considered chip-in-the-loop training, where a simulation model of the chip is used instead of the chip itself.

Although on-chip training may be desirable in many situations to exploit the speed of network parallelism without reaching an input-output bottleneck (24), very few successful implementations have been reported. The main problem with on-chip training has been the difficulty of implementing the training hardware, both because of the size of the training hardware and because of precision mismatches in forward and reverse operations. Masa et al. (17) estimate that the extra circuitry required for on-chip learning is at least as large as the neural circuitry without on-chip learning. They also note that the computational precision required for most learning algorithms is difficult to achieve with analog approaches and that cost increases more rapidly with increased precision for analog designs in comparison with digital designs. Montalvo et al. (24) describe a proof-of-concept chip that holds promise for implementing on-chip training while meeting stringent requirements on cost, power, flexibility and system integration.

They note that backpropagation assumes sigmoidal neurons and linear synapse multipliers. A training algorithm, which does not require linear multipliers, is desirable to reduce synapse sizes. Perturbation-based algorithms are well-suited for analog VLSI implementation because they do not assume particular synapse and neuron characteristics. In se-

rial weight perturbation, weights are updated according to *measured* error gradients by

$$\delta w = -\eta \, \Delta E / \Delta w \tag{6}$$

where η is the learning rate, Δw is a weight perturbation and $\Delta E = E_{\text{pert}} - E_{\text{nom}}$ is the difference between the observed error with the weight perturbation E_{pert}, and the observed error with weight perturbation E_{nom}. Serial weight perturbation is slow, of order $O(W)$ where W is the number of weights, because all weights are updated serially.

Montalvo, Gyurcsik, and Paulos (24) introduce a faster perturbation-based algorithm, CHain Rule Perturbation (CHRP), in which the outer layer weights are updated directly using Eq. (6) and hidden-layer weights are updated using the chain rule. CHRP is similar to the well-known Madeline Rule III algorithm (25), except that neuron outputs rather than neuron inputs are changed. If a network has J hidden nodes and I inputs, weight update requires only $O(2J + I)$ operations. Implementing CHRP requires placing weight-update circuits with every neuron. This is regarded as a reasonable trade-off between neuron compactness and weight-update speed.

It is noted that precision requirements are quite different for digital and analog implementations of neural networks. For digital weight storage, their simulations (26) and review of the literature suggest that 5-bit registers, corresponding to 5-bit weight resolution, are sufficient for feedforward computation, but 12-bit resolution is necessary for learning, although this is somewhat problem-dependent. Low analog precision is generally the result of nonlinearity and offset. Since nonlinearity can be handled by using an appropriate training algorithm as described in the foregoing, offsets are the most serious problem. Offsets tend to accumulate as weights are changed. This is particularly dangerous late in the training process where offsets can change the sign of small weight updates.

Precision requirements in the analog domain were studied by examining the effect on a 1–20–6–1 feedforward network solving a function mapping problem. The mapping was from a random number in the range $(-1, 1)$ to a sigmoidal function and was used previously to study digital precision requirements (26). Offsets on the order of 0.000244, corresponding to 2^{-12} or a resolution of 1 bit in 12, were required to keep rms output errors below 0.02. Of particular interest was the observation that with higher offsets ($0.000976 = 2^{-10}$), rms output errors reached low values (0.02) after less than ten training cycles but then rose to high values (0.1). Stopping at the right point in the training process requires a more sophisticated training procedure. Although adding a random offset of $0.03 = 2^{-5}$ to correctly trained weights produced an rms error of 0.1, the output closely matched the desired sigmoidal function.

The synapse circuit, shown in Fig. 3, is a nonlinear four-quadrant multiplier with floating gate current sources as was used in the ETANN chip. The weight is altered by varying the current of F2 about the constant current of F1. Changing the charge on the floating gate requires high-voltage pulses and may take hundreds of microseconds. During training, a DRAM cell is used to store the control-gate voltage, which may require many thousands of small weight updates. The DRAM circuit shown is a two-MOSFET CMOS inverter

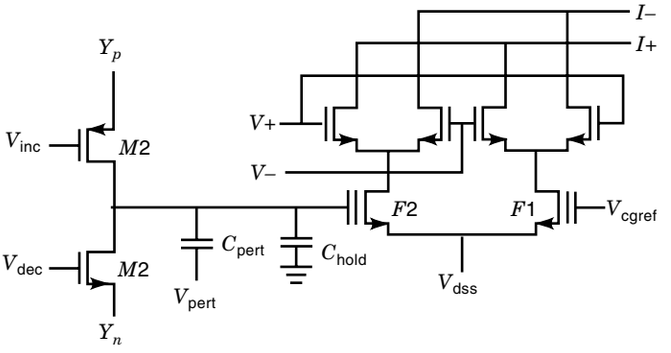

Figure 3. Complete synapse circuit (24). This is a nonlinear, four-quadrant multiplier circuit which uses floating-gate technology for storing weights.

attached to a 1 pF storage capacitor C_{hold}; Y_p and Y_n are held to the supply voltage or ground. In the hold mode, Y_p is at ground and Y_n is at the supply voltage, thus both transistors are off. During training or learning, the charge on C_{hold} is updated by pulsing Y_p to the supply voltage and Y_n to ground.

Very small charge packets can be added to or removed from C_{hold} depending on the learning-control signals V_{inc} and V_{dec}. For a 10 nA charging current having a 10 ns pulsewidth, the change in the stored voltage is 100 μV. Since the floating-gate MOSFETs have a transconductance of about 30 μA/V and a nominal current of 10 μA, the change in supply current is 3 nA, corresponding to a resolution of $0.0003 \approx 2^{-12}$; V_{pert} and C_{pert} are used to perturb weights. Since $C_{\text{pert}} \approx 15$ fF, very small perturbations can be applied without affecting the stored weight on C_{hold}. Since this circuit does not produce weight updates of the wrong sign, the offset problem is minimized. The result is a compact synapse circuit occupying 4900 μm^2 in a 2 μm technology.

After training, the current is temporarily stored as a voltage in a sample-and-hold circuit on the periphery of the chip. High-voltage pulses are then applied to $F2$ until its current matches that in the sample-and-hold circuit. High precision is difficult to achieve when programming floating-gate devices. Fortunately, since programming $F2$ is essentially a recall operation, 5 bit precision is sufficient. Floating-gate devices tend to make poor current sources because the floating-gate voltage, and consequently the drain-source current, depend on the drain voltage. Balanced input voltages help by limiting the voltage swing at the drains of the floating-gate devices.

Note that because the synapse is nonlinear, the neuron can be linear without affecting the behavior of the network. To allow flexibility the synapses are connected by a reconfiguration switching matrix that adds about 20% to the total synapse area. In their 8 neuron, 64 synapse proof-of-concept chip, over half of the chip area is consumed by weight-learning circuits. However, this would be expected to shrink below 10% for a 100 neuron chip.

SUBTHRESHOLD ANALOG NEURAL CHIPS

Mead (27) has argued eloquently that weak inversion subthreshold MOS circuits are a natural way to implement neural systems. He notes that two barriers have historically blocked the path to creating a nervous system in silicon. First,

neural systems require far greater connectivity than standard VLSI circuits. Second, there was not sufficient knowledge of the organizing principles of neural systems. He notes also that MOS device noise levels are higher and precisions are lower than for the bipolar technologies usually employed to implement analog functions. However, these factors are even worse for neural wetware, giving hope that MOS technology could be used to implement neural systems. MOSFETs act as controlled sources of positive and negative current. Because the control does not draw current, MOSFETs are a nearly ideal circuit element.

In weak inversion or subthreshold operation the drain-source current is given by $I_{ds} = I_0 \exp(\kappa V_{gs}/V_T)$ for saturation; V_{gs} is the gate–source voltage, $V_T = kT/q = 26$ mV at room temperature, and κ measures the effectiveness of the gate potential in controlling the channel current. In subthreshold saturation the MOSFET is a voltage-controlled current source with exponential transfer characteristics, corresponding to a transconductance $\kappa I_{ds}/V_T$. In addition to the general advantages of MOSFETs, three characteristics make subthreshold circuits attractive:

1. Low currents mean very low power dissipations of 1 pW to 1 μW per circuit.
2. Since the current saturates for drain-source voltages greater than a few kT/q, the MOSFET can operate as a current source over most of the voltage range from ground to the supply voltage.
3. The exponential transfer characteristic is an ideal computation primitive for many applications because it allows current to be controlled over many orders of magnitude.

Mead introduces the simple five-MOSFET *transconductance amplifier* shown in Fig. 4 as a primitive circuit. This produces a current that is a tanh function of an input voltage difference. This sort of saturating function is useful for realizing nonlinear neuron behavior. For small voltage differences it produces a product of the input voltage difference and the bias current. This is useful for realizing synapse functions. By adding three MOSFETs the simple transconductance amplifier can be converted to a *wide-range amplifier* that allows input and output voltages to cover almost the entire range between the supply voltage and ground. The intent is to design neural systems using primitive circuits that make efficient use of silicon real estate.

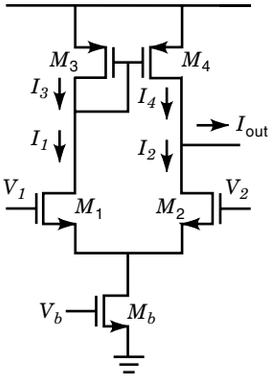

Figure 4. Mead's five-transistor transconductance amplifier (27). This produces a current which is a tanh function of an input voltage difference, which is useful for realizing nonlinear neuron behavior. This circuit has a limited dynamic range, and more transistors are required to achieve a wide-range amplifier.

Mead considers means of performing elementary arithmetic operations in analog fashion with voltages and currents. Among the arithmetic functions he considers are identity, addition, and multiplication. The transconductance amplifier acts as a two-quadrant multiplier; a four-quadrant multiplier can be created by using each of the output currents from the original differential pair as the inputs for a pair of differential pairs. Using a current mirror to combine these currents produces a 9-MOSFET Gilbert transconductance multiplier where

$$I_{out} = I_b \tanh[\kappa(V_1 - V_2)/2] \tanh[\kappa(V_3 - V_4)/2] \qquad (7)$$

This circuit produces output currents on the order of 30 nA for $|V_1 - V_2|, |V_3 - V_4| \leq 200$ mV. However, Mead shows how this multiplier becomes highly nonlinear if $V_2 \geq V_4$. A more linear wide-range multiplier can be created but this requires almost twice as many MOSFETs.

In the steep part of the tanh characteristics an input voltage change of 35 mV produces an output current change of 3 nA. Two MOSFET circuits can be created that generate an output current exponentially related to the input voltage or an output voltage proportional to the logarithm of the input current. Logarithmic compression allows circuits to handle the wide dynamic range of sensory signals.

Much neural network research has been focused on higher-level cognitive tasks such as image recognition. Mead's research has been focused on the lower-level precognitive tasks involved in handling sensory inputs, which have to be processed through many layers of representation before cognition occurs. He argues that it is relatively easy to make up an image and recognize it with a vision system, while it is really difficult when that vision system is exposed to real sensory data. Implementing the processing of sensory data has led to unconverted neural circuits such as modeling a retina as a hexagonal array of resistors. Such an array computes a smooth weighted average over a number of neighbors, with neighbors farther away contributing less to the average. If the resistors are photoconductors having resistance that corresponds to the intensity of light, such an array has retinal properties. Mead uses two pass transistors in series to implement a resistor in which the current is proportional to the hyperbolic tangent of the voltage across the resistor. The saturation of these resistors allows the retina to sense discontiuities by letting the network saturate and then discerning the boundaries at which saturation occurs.

Neural circuits need to process time-varying signals to handle the sensory inputs required for sound and motion detection. A *follower–integrator circuit* is formed from the transconductance amplifier by placing a capacitance to ground at the output and feeding the output back to the negative input. This circuit allows signals to be stored for short time periods. Because the output of the follower–integrator circuit is a moving average of the input signal to which earlier signals contribute exponentially less, it is particularly appropriate for neural systems where we expect old memories to decay. Implementation of the integrator-follower circuit in CMOS tech-

nology is complicated by the fact that the largest capacitance available is the gate capacitance of a MOSFET. The gate of the MOSFET used as a capacitor must be connected so that it remains in strong inversion, and its channel provides a good conductor and keeps the capacitance large.

To observe *changes* in the input signal pattern requires *differentiation* of the signal with respect to time. The sharpness of the signal any implementation of a differentiator can differentiate is limited because circuit resistances limit the current that can be drawn from any source.

A good illustration of the approach taken by Mead and his students to neural chip design is the SeeHear chip. This chip maps visible signals from moving objects into binaural signals which can be projected through earphones. This would enable a visually impaired person to locate moving objects aurally. The chip is a compact CMOS design that encodes the intensity and position of a light source in a two-dimensional retinoptic projection, processes the electrical signals representing intensity information to emphasize temporal changes, and synthesizes a sound having the appropriate psychophysiologically determined cues for a sound source at that position. The chip contains a retina of 32 rows of 36 pixels each connected to analog delay lines. Both horizontal and vertical displacements can be determined by direct analogy to the processing of auditory signals. Interestingly, the approach to detecting horizontal displacements, which is based on a binaural-head-shadow model, is more effective than the approach to detecting vertical displacements, which is based on modeling the pina and tragus of the outer ear.

Subthreshold operation of MOSFETs has one serious drawback. Because such operation is low current, it is very slow as well as very low power. The time to switch a MOSFET circuit Δt depends on the voltage change ΔV associated with the switch, and the capacitance C of the device and interconnects. Since $I = C\,dV/dt$, $\Delta t = C\,\Delta V/I$. Thus, switching times are inversely proportional to switching currents. For $C \approx 0.5$ pF and $\Delta V \approx 5$ V, typical for a 1 μm CMOS process, and $I \approx 1$ nA, typical for subthreshold operation, $\Delta t \approx 2.5$ ms. This is roughly a million times longer than the switching time for a conventional static CMOS gate using the same CMOS process.

PULSE STREAM NEURAL CHIPS

Pulse stream techniques for neural chips are interesting because they can combine the compactness of analog computation with the simplicity and robustness of digital signals and devices. Murray et al. (28) describe the Edinburgh Pulse Stream Implementation of Learning-Oriented Network (EPSILON) neural chips that use pulse streams for communication. The EPSILON-II neural chip contains 32 input neurons (hidden layer), 32 output neurons, and $32 \times 32 = 1024$ synapses on a 48 mm^2 die when implemented in 1.5 μm CMOS technology. It operates at a rate of 102.4 MCPS; each synapse output can be determined in 10 μs.

The synapse design is based on an analog *transconductance multiplier* in which three NFETs, connected as a pullup, a pulldown, and a pass transistor, form the multiplier. In this multiplier, the magnitude of the output current pulse is determined by the capacitance-stored weight voltage on the gate of the pulldown NFET. The width or frequency of current pulses is determined by the width or frequency of the voltage pulses on the gate of the pass NFET that come from a neuron output. This multiplier is naturally two quadrant because neural states are unipolar while weights are bipolar. The charge packets from the synapses are integrated to provide the total post-synaptic activity voltage.

Two neuron designs were incorporated on the EPSILON chips: a synchronous, pulsewidth modulation (PWM) neuron and an asynchronous, pulse-frequency modulation (PFM) neuron. The PWM neuron is a comparator that compares the activity voltage with a sigmoidal ramp voltage generated off-chip. The output is a digital, fixed-amplitude voltage pulse of width from 0 μs to 20 μs, which depends on the magnitude of the activity voltage. The PFM neuron is a voltage-controlled oscillator having an output that is a digital pulse stream of fixed amplitude but varying interpulse spacing.

The EPSILON chips store weights as analog voltages on capacitors. This dynamic weight storage requires external refresh circuitry. Murray et al. (28) report experiments using amorphous-silicon analog memory devices as an alternative for fast, nonvolatile weight storage. The EPSILON chip performs only the forward-pass phase of neural computation; the learning phase is performed off-chip in an associated computer. Loading weights require 2.3 ms in the EPSILON-II chip.

Murray et al. (28) regard on-chip learning as essential if neural chips are to be used in autonomous neural systems that address real-time, real-cost applications. They report a target-based training algorithm related to backpropagation, but which uses only local information to update weights and has identical weight-update strategies for both output and hidden-layer neurons. Implementing this algorithm requires weight-by-error multiplication. Since both the weight and error are bipolar, a four-quadrant multiplier is required. To make a four-quadrant multiplier, a second transconductance multiplier with complementary input voltages is added in parallel to the transconductance multiplier used as a synapse.

Purely digital pulse-stream implementations can be implemented as neural chips. Such implementations can use conventional digital logic circuits, but consume considerable area. Masaki et al. (29) report a chip that contained six neurons, 42 excitory synapses, and 42 inhibitory synapses implemented in 1.3 μm CMOS gate-array technology. The result consumed 18 kgates of the 24 kgates possible on an approximately 1 cm^2 chip in a 240 pin-grid package. Their neuron circuits were based on a biological neuron model in which the synapse circuit transforms an input pulse density f into a pulse density proportional to the synapse weight wf. This is accomplished by a 6 bit rate multiplier driven by a 6 bit storage register. The synapses drive a dendrite circuit consisting of an OR gate that sums synapse output pulses. Two dendrite circuits, for excitation and inhibition, drive an up-down counter that serves as the neuron cell body. An associated computer was used to execute learning algorithms and update synaptic weights.

Similar circuits were implemented using wafer-scale integration in 0.8 μm CMOS gate-array technology; 576 neurons were implemented on a 125 mm wafer; 576 synapse connections were calculated simultaneously in a 464 ns step time for a processing rate of 1.2 GCPS. A complete forward pass through the neural network required 576×464 ns = 267 μs.

Of the 40 million MOSFETs fabricated on a wafer, 19 million were used to implement the 576 neuron network.

IMPLEMENTING ALTERNATIVE NEURAL NETWORK ARCHITECTURES

Murray (30) provides an excellent summary of neural network architectures and algorithms. He notes that while the Hopfield network revitalized neural network research, feedforward networks have been much more useful for applications in pattern recognition and classification. In addition to the multilayer perceptron, neural networks have been constructed around kernel nodes that implement Gaussian Radial Basis Functions (GRBF) in the hidden layer. One can show that any function can be approximated by a train of impulses if sufficiently many impulses are present. Actually, these impulses can be generated by subtracting one sigmoid from another, if one can offset the sigmoids. Another way to generate these impulses is to use Gaussian functions. The addition of many Gaussians whose centers can be adjusted can be used to approximate functions. If the widths of these Gaussians are adjustable as well as their centers, one can use much fewer units where "fat" Gaussians can be used in some regions of the function and "thin" Gaussians in other regions. Furthermore, the amplitudes have to be multiplied by some weights before being added to form the final function. This leads to the concept of Radial Basis Functions (RBF) where Gaussians are preferred most of the time. The advantage of this type of network is the smaller number of units required compared with multilayer perceptrons for the same application. This has been an incentive to study hardware implementations of these networks.

In Gaussian Radial Basis Function (GRBF) networks, besides multiplication, which corresponds to synapses, one has to implement Gaussian functions that have variable widths and centers. This has proven to be a very difficult task with CMOS technology because the Gaussian requires an exponential function while MOS devices have square-law behavior when operating above threshold. Some researchers have attempted to use the subthreshold region of MOSFET operation, resulting in a very reduced output swing and sensitivity to noise. Others have used BiCMOS technology because the base-emitter junction of the bipolar transistor has exponential behavior. This has made the circuits so large and expensive that GRBF implementations have lost their attractiveness. Other researchers have tried to approximate the Gaussian function by piecewise-linear approximations. However, none of the implementations have been too successful.

Another incentive for using GRBF networks is their close connection to fuzzy logic. If the Gaussians are replaced by the trapezoidal "membership functions" of fuzzy logic, one obtains a network where any input could belong to several membership functions to a degree determined by the connection weights. Hence, one could train a GRBF network and from the weights and the locations and widths of the Gaussians, one could deduce the "fuzzy rules" governing a system. This is another incentive to design GRBF chips. Many fuzzy chips have been designed with trapezoidal or triangular membership functions instead of Gaussians, using OR or MAX operations to combine the outputs instead of weighted addition. These have been much simpler to implement as integrated circuits.

In neural networks learning can be either supervised or unsupervised. In supervised learning, there is always a teacher or at least a critic (who, unlike a teacher, does not give the correct answer, but indicates whether or not the answer is correct), but in unsupervised learning the network is alone. Unsupervised learning can be either Hebbian, where more than one output unit is active at any time, or competitive, where only one output unit is active at any time. In *competitive learning,* the output units compete among each other to become the winner. The winner inhibits all the other units to ensure that it is the only ouput unit that corresponds to that particular input combination. These output units are known as Winner Take All (WTA) cells. Competitive networks are often used in many classification problems where the correct input-output units are not known. In the competitive learning scheme described in the foregoing, the actual location of the output units with respect to each other is immaterial. However, if the geometrical locations of these units are also utilized we have Self-Organizing Maps (SOM). In SOMs, the location of the winning output conveys some information, and nearby output units correspond to nearby input patterns. Then, we have a feature map.

An algorithm to obtain this effect was introduced by Kohonen (31). In this algorithm, we start with random weights. The output that wins for a particular input pattern is enhanced. However, that particular output is not the only one that is enhanced. In addition, neighboring outputs are also positively encouraged. The neighborhood function is large at first, but slowly becomes smaller and smaller. Finally, outputs that are close to each other in the n-dimensional input space become close to each other in the m-dimensional output space. Generally, m is chosen as two so that it is easy to visualize the grouping of the input space on a map. The output is a topology-preserving map, such that the n-dimensional topology is mapped to a two-dimensional topology with distance information remaining intact. Kohonen Self-Organizing Feature Map (K-Map) networks are a two-dimensional array of neurons that are trained so that neurons, which are physically closer in physical space, respond to data inputs that are closer in a multidimensional feature space. Thus, the K-map implements a competitive clustering algorithm without external training. Note that the K-Map is a good method for selecting kernel sites in an RBF classifier.

Electronic implementations of the SOM have been rather few due to the difficulty of the system. Besides synapses, one requires distance-calculating circuitry, WTA cells and nonlocal interconnections, at least for the first few iterations where the neighborhood is very large. Carlen and Abdel-Aty-Zohdy (32) describe an SOM implementation in which purely digital neural pocessing units (PU) perform parallel computations. The system was designed to process 2-D input vectors at a 10 MHz rate. Each input vector was 7 bits long; an 8 bit data bus allowed a range of similarity distances from 1 to $2^7 + 2^7 = 2^8$. Each PU contains adder, subtractor, adder-subtractor, and multiply units connected with four registers and an 8×8 SRAM in addition to weight- and neighborhood-compare units. This allowed each PU to calculate the distance between input vectors, update weights, and compare weights and neighborhoods. Some simplification of Kohonen's algorithm was required; calculating a Manhattan rather than a

Euclidian similarity distance avoided computing square and square-root functions; 93 cycles are required for one iteration of the system, with the bulk of the time consumed by SRAM fetch and store operations. This corresponds to 100 K iterations/s, compared with 200 iterations/s for a Matlab simulation. When implemented in 2 μm CMOS technology, a *4-bit wordlength* PU containing 3000 MOSFETs could fit within a 5 mm^2 area.

An impressive implementation is described by Fang and co-workers (33) of a VLSI neural processor chip for image data compression using a self-organizing map network. They note that neural approaches are attractive because it is difficult to implement high-speed vector quantization (VQ) for image compression using more conventional digital signal processing (DSP) circuits. Their frequency-sensitve self-organization (FSO) approach systematically distributes code vectors to approximate the unknown probability function of the training vectors. Code vectors quantize the vector space and converge to cluster centroids; a synapse weight is stored as a code vector in their implementation. The learning rule moves the winning code vector toward the training code vector by a fractional amount, which decreases as the winning frequency increases.

Their system was implemented as a VQ codebook generator chip and a VQ chip attached to an external computer. The VQ codebook generator chip is implemented with conventional DSP circuits, while the VQ chip implements a highly paralleled neural network. The VQ chip is implemented as a mixed-signal VLSI design in which analog circuitry performs highly parallel neural computation and digital circuitry processes multiple-bit address information.

The VQ chip is composed of two neuron layers, M input neurons and N current-summing output neurons, followed by a bank of winner-take-all (WTA) cells and a digital encoder. The $M = 25$ neurons in the input layer respond to the elements of an M-dimensional input vector. Each input neuron distributes its output to $N = 64$ distortion-computing output neurons in the competition layer through a matrix of $25 \times 64 = 1600$ programmable synapses. The synapse cells correspond to N M-dimensional code vectors. Each distortion-computing neuron in the N-neuron output array calculates the square of the Euclidian distance between its code vector and the input vector. The WTA block contains N competitive circuit cells that compare the N distortion values and declare a single winner. This is followed by an N-to-n digital encoder. The updated code vector is written to the digital codebook memory on the VQ codebook generator chip and from there to the analog synapse memory.

They note that detailed studies to improve the performance and reduce the area and power of circuits are essential to implement complex neural systems in VLSI technology. Their studies included both computer simulations and laboratory experimentation. For example, for $N = 64$, the input neuron must be designed to handle a large 5 pF load capacitance. Consequently, each input neuron was designed as a unity-gain buffer, implemented as a conventional operational amplifier in a unity-gain configuration. It requires 80 ns to settle to within 0.1% accuracy for the ± 1.5 V input pulses. The programmable synapse is a modified, wide-range, four-quadrant Gilbert multiplier with 8 bit precision. One 15 MOSFET synapse occupies 0.0092 mm^2 in 2 μm CMOS technology. The nonlinearity error is less than 2% over the ± 1.5 V input range.

The output summing neuron is an operational amplifier that converts the summed current into an analog voltage that is sent to the WTA circuit. This output neuron supports a large summing current range which exceeds 1 mA. To ensure linear current-to-voltage conversion over this wide range of operation, a 2 kΩ linear feedback resistor is used to convert current into voltage. Multiple MOSFETs biased in the triode region were used to synthesize an accurate resistor. The area of the output summing neuron is 0.026 mm^2; the area of the linear resistor is 28% of that. A WTA cell has an area of 0.0056 mm^2.

Less than 500 ns simulated time was required for one network interation that includes input buffering, synapse multiplication, neuron summing, WTA, and index encoding. Power lines for the digital and analog blocks were separated to avoid coupling digital switching noise to the highly sensitive analog circuits. The overall chip area for a 25-dimensional vector quantizer of 64 code vectors was 31 mm^2. Its throughput rate is 2 million vectors/s, corresponding to a processing rate of $2 \times 1600 = 3.2$ GCPS.

CONCLUSION

This article has examined integrated circuit implementations of neural circuits. Although Eqs. (1) and (2) describing neural circuits are quite simple, neural chip implementations are remarkably diverse and surprisingly complex. Neural chips have been implemented in the dominant integrated circuit technology, CMOS, in both digital and analog forms. Three keys to neural chip implementation are that there are many more synapses than neurons, that training requires much higher precision than execution, and the need for linear multiplication. For many neural networks the number of synapses required is roughly the square of the number of neurons, and Eq. (2) indicates that a linear multiplication is required for every synapse. In digital implementations, linearity is assured, but greater precision generally requires correspondingly greater area. High-precision, high-speed (parallel) digital multipliers are large circuits. Analog multipliers are generally faster and smaller but have much less precision; maintaining linearity requires more transistors. Training requirements and the need to store the weights of trained synapses have required significant off-chip resources in many designs. The requirement of linear, bipolar multiplication limits the effectiveness of single-transistor synapses that use a floating-gate storage technology. Algorithms are needed which minimize the effects of component variation and limited precision and linearity on neural chip performance. Highly variable, nonlinear chips will likely require individualized training for each chip. This is unlike conventional digital chips where one program can run on all chips.

The diversity of approaches to neural chip design suggests we are in a period of ferment from which a dominant neural chip technology is yet to emerge. Different applications may favor different approaches, such as subthreshold analog chips and pulse-stream chips that mix analog and digital technology. As one shrinks CMOS circuit dimensions to improve speed and integration density, the dominance of interconnect

delays will favor locally connected neural network architectures which minimize long, global interconnects.

BIBLIOGRAPHY

1. R. P. Lippman, An introduction to computing with neural nets, *IEEE Acoust. Speech Signal Process. Magazine,* **4** (2): 4–22, 1987.
2. G. Nagy, Neural networks—then and now, *IEEE Trans. Neural Netw.,* **2**: 316–318, 1991.
3. M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry,* Cambridge, MA: MIT Press, 1969.
4. J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. National Acad. Sci.,* **79** (8): 2554–2558, 1982.
5. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning internal representations by error propagation, in D. E. Rumelhart and J. L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. vol. 1: Foundations,* Cambridge, MA: MIT Press, 1986.
6. P. DeWilde, *Neural Network Models,* Berlin: Springer-Verlag, 1996.
7. P. Mueller, T. Martin, and F. Putzrath, General principles of operations in neuron nets with applications to acoustic pattern recognition, in E. E. Bernard and M. R. Kare (eds.), *Biological Prototypes and Synthetic Systems, vol. 1,* New York: Plenum, 1962.
8. N. H. Farhat, Optoelectronic neural networks and learning machines, *IEEE Circuits Devices Magazine,* **5** (5): 32–41, 1989.
9. H. P. Graf and L. D. Jackel, Analog electronic neural network circuits, *IEEE Circuits Devices Magazine,* **5** (4): 44–49, 55, 1989.
10. P. Kolinummi, T. Hammalainen, and K. Kaski, Designing a digital neurocomputer, *IEEE Circuits Devices Magazine,* **13** (2): 19–27, 1997.
11. J. Beichter et al., A VLSI array processor for neural network algorithms, *Proc. IEEE Custom Integrated Circuits Conf.,* 1993, pp. 821–823.
12. J. B. Burr, Digital neurochip design, in K. W. Przytula and V. K. Prasanna (eds.), *Parallel Digital Implementation of Neural Networks,* Englewood Cliffs, NJ: Prentice-Hall, 1993.
13. R. C. Frye, E. A. Rietman, and C. C. Wong, Back-propagation learning and nonidealities in analog neural network hardware, *IEEE Trans. Neural Netw.,* **2**: 110–117, 1991.
14. F. J. Kub et al., Programmable analog vector-matrix multipliers, *IEEE J. Solid-State Circuits,* **25**: 207–214, 1990.
15. J. B. Lont and W. Guggenbühl, Analog CMOS implementation of a multilayer perceptron with nonlinear synapses, *IEEE Trans. Neural Netw.,* **3**: 457–465, 1992.
16. P. W. Hollis and J. J. Paulos, Artificial neural networks using MOS analog multipliers, *IEEE J. Solid-State Circuits,* **25**: 849–855, 1990.
17. P. Masa, K. Hoen, and H. Wallinga, A high-speed analog neural processor, *IEEE Micro,* **14** (3): 40–50, 1994.
18. M. Hatamian and G. L. Cash, A 70 MHz 8-bit $\times$ 8-bit parallel pipelined multiplier in 2.5 μm CMOS, *IEEE J. Solid-State Circuits,* **SC-21**: 505–513, 1986.
19. Anonymous, 80170NX Electrically Trainable Analog Neural Network (ETANN). In *Intel Specifications,* 1991.
20. M. Holler et al., An electrically trainable artificial neural network (ETANN) with 10240 "floating gate" synapses, *Proc. IEEE Int. Conf. Neural Nets (INNS),* 1989, pp. II191–II196.
21. G. Dündar, F-C. Hsu, and K. Rose, Effects of nonlinear synapses on the performance of multilayer neural networks, *Neural Computat.,* **8** (5): 939–950, 1997.
22. I. Bayraktaroglu et al., ANNSyS: an analog neural network synthesis system, *Proc. Int. Conf. Neural Nets,* 1997, pp. II190–II915.
23. P. J. Edwards and A. F. Murray, *Analog Imprecision in MLP Training,* Singapore: World Scientific Press, 1996.
24. A. J. Montalvo, R. S. Gyurcsik, and J. J. Paulos, Toward a general-purpose analog VLSI neural network with on-chip learning, *IEEE Trans. Neural Netw.,* **8**: 413–423, 1997.
25. D. Andes et al., MRIII: a robust algorithm for training neural networks, *Proc. Int. Joint Conf. Neural Netw.,* **I**: 1990, pp. 553–556.
26. P. W. Hollis, J. S. Harper, and J. J. Paulos, The effects of precision constraints in a backpropagation learning environment, *Neural Computat.,* **2** (3): 363–373, 1990.
27. C. Mead, *Analog VLSI and Neural Systems,* Reading MA: Addison-Wesley, 1989.
28. A. F. Murray et al., Pulse stream VLSI neural networks, *IEEE Micro,* **14** (3): 29–39, 1994.
29. A. Masaki, Y. Hirai, and M. Yamada, Neural networks in CMOS: a case study, *IEEE Circuits Devices Magazine,* **6** (4): 12–17, 1990.
30. A. F. Murray, *Applications of Neural Networks,* Boston, MA: Kluwer Academic Publishers, 1995.
31. T. Kohonen, *Self Organization and Associative Memory,* 2nd ed., New York: Springer-Verlag, 1988.
32. E. T. Carlen and H. S. Abdel-Aty-Zohdy, *Proc. IEEE Midwest Symp. Circuits Syst.,* 1993, pp. 958–962.
33. W-C. Fang et al., A VLSI neural processor for image data compression using self-organization networks, *IEEE Trans. Neural Netw.,* **3**: 506–518, 1992.

GUNHAN DÜNDAR
Bogazici University

KENNETH ROSE
Rensselaer Polytechnic Institute

NEURAL COMPUTING. See COMPUTATIONAL INTELLIGENCE; NEURAL NET APPLICATIONS.