AN ABSTRACT OF THE THESIS OF

Jeanette K. Skelton for the degree of Master of Science in Electrical and Computer Engineering presented on April 27, 1989.

Title: Analysis and Synthesis of Neural Networks.

Abstract approved: Redacted for Privacy

John M. Murray

The brain has long attracted the interest of researchers. Some tasks, such as pattern recognition and optimization, have proven to be exceptionally difficult for conventional computing systems to perform, but are executed by the brain almost effortlessly. Due to the large number of neurons and interconnections, it has proven impossible to model the brain's architecture exactly. Researchers have instead created modest networks of artificial neural elements which can perform some interesting functions, such as generalization, pattern recognition, and the solution of certain optimization problems. These artificial neural networks do not work in exactly the same manner as the brain but they are proving useful in numerous practical applications.

In this thesis the analysis, design, and hardware implementation of an artificial neural network is presented. In order to fully appreciate the architecture of the neural network described, a discussion of the properties of biological neurons and a brief history of neural network research is included. This is followed by a detailed discussion of the design, implementation, and testing of the network.

The network under consideration is a general purpose time-multiplexed neural network suitable for VLSI implementation. A non-multiplexed neural network implementation generally requires full connectivity of the neural elements. This approach consumes too much chip area. The multiplexed approach taken in this thesis offers substantial area savings over the non-multiplexed approach.

Analysis and Synthesis of Neural Networks

by

Jeanette K. Skelton

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed April 27, 1989

Commencement June, 1989

APPROVED:

Redacted for Privacy

Associate Professor of Electrical and Computer Engineering

Redacted for Privacy

Head of department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

Date thesis is presented April 27, 1989

ACKNOWLEDGEMENT

This thesis could not have been completed without the help of several people. First, my parents, John and Katherine Skelton, gave me support and encouragement throughout my education. My advisor, John Murray, guided my studies and recommended some good books. Joe Minne helped me in my minor, and special thanks are due to my other committee members Jim Herzog and Keith Levien.

Finally, credit should be given to Dave Allstot, for his idea of using switched-capacitors in a neural network implementation, and to Jim Hansen, for designing and actually building the circuit and for buying me a few cups of coffee.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDIX FIGURES

Figure Page

ANALYSIS AND SYNTHESIS OF NEURAL NETWORKS

INTRODUCTION

Conceptually, computers should be able to outperform people. In an era when the power of a typical computing machine is measured in megaFLOPS (one million FLoating Point Operations Per Second), the brain cannot achieve the equivalent of several FLOPS. On the other hand, a person can readily understand speech or recognize a face, far surpassing the capabilities of the most complex, high-performance computer systems. The speed and accuracy achieved in solving certain very complex pattern recognition problems has long fascinated researchers, leading them to try to model neurons, the basic element of the brain. Some of the basic properties of neurons have been modelled fairly well, but the overall functionality of the brain is not yet understood well enough for modeling purposes.

Over the last three decades a number of models of certain specialized subsystems of the brain have been generated. These simplified models are the basis for the hardware implementation of networks of simple processing elements and their associated interconnections that can perform a number of practical engineering functions. Such networks are called "neural networks". Whether or not the brain actually works like these networks is not yet known. In practice, these networks have shown some particularly interesting emergent properties, such as the ability to generalize. Some of these networks have specific functions wired into them; others can modify themselves, so that they can learn, over time, to solve a broad class of problems.

Many neural network architectures may be readily implemented in VLSI due to the simplicity of their architectural building blocks. However, a general neural network requires full interconnection of the processing elements. In the general case, the space required for the interconnections increases in proportion to n^2, where n is the number of processing elements. In order to overcome this problem, a unique, time-multiplexed neural network circuit was designed and tested. The focus of this thesis is the analysis and synthesis of this unique neural network circuit. The final breadboard version of the circuit contained sixteen neural processing elements, and was based on the use of a switched-capacitor multiplying digital-to-analog converter. The analysis of a broad spectrum of neural network models plus a significant amount of tutorial material has

been included in the thesis in order to provide the reader with an evolutionary perspective of past research, and what may yet be accomplished in the future in this rapidly changing, interdisciplinary field.

MOTIVATION

In many problems, conventional computers cannot yet match the capabilities of the human mind. For example, in playing chess a master chess player can still beat a machine. A master may consider many positions and look many moves ahead to decide on a move. Rarely, though, does a human look at more than a hundred possible positions. The strength of computer chess programs lies in their ability to look many moves ahead and evaluate more possible positions (on the order of a million) than the human (Levy, 1982). Winning a chess game, on the other hand, depends on more than simply the ability to look at positions:

> The great difference between computers and human chess players lies in the area of selecting moves and evaluating positions. Most human chess decisions are in large part intuitive and are based on associative memory. A human player KNOWS certain things without being consciously aware of the process: — that square will be weak — this Bishop will be bad — positions of this type are disagreeable to him, even if objectively satisfactory — White must have a winning attack in this position — and so on. These value judgments are based on past experience, and represent things which the player has LEARNED.... A computer can compare two positions and tell whether they are exactly alike rather easily, but it is extremely difficult to program a computer to recognize that two positions are SIMILAR. As a result, today's programs make no attempt to recognize positional patterns or to form positional strategies.... If a computer has such difficulty in grasping such concepts as "square", "circle", "round", "sharp" and the like, how will programs deal with chess patterns and the abstractions that relate them? The whole problem of pattern recognition and forming and recognizing abstractions appears to be crucial to giving programs any real "chess knowledge" in a human sense (Welsh, 1984, pp. 90-92).

Thus, the human ability to recognize patterns can be used to solve problems better than a traditional computer. Artificial neural networks, however, have been used to solve complex pattern recognition problems, and can modify themselves in order to learn a solution. So far, the problems studied have not been as difficult as learning to win a chess game, but advances in neural network capabilities make this a possibility. In areas where the human mind and associated sensory systems perform better than traditional computer algorithms, neural networks may eventually find wide application, although they may not model the architecture of the brain exactly.

BIOLOGICAL NEURONS

To understand how artificial neural networks work, it is necessary to first understand the form and function of biological neurons and neural networks.

NEURAL ANATOMY

The nervous system consists of the brain, the spinal cord, peripheral nerves, and the sense organs. The neuron, illustrated in figure 1, is the basic unit of the nervous system. Neurons serve to transmit information via electrical impulses or action potentials throughout the body. The cell wall of the neuron has many outgrowths called dendrites, which act as input terminals to the neuron. The axon, a very long outgrowth of the neuron, acts as the output terminal from the neuron to other neurons in the network. Neurons are connected through special junctions called synapses. A synapse is located between the axon terminal of a presynaptic neuron, and a dendrite of an associated postsynaptic neuron. When the presynaptic neuron fires, producing an output signal or action potential (figure 2), it releases neurotransmitter chemicals from the axon terminals into the space between the axon terminal and the postsynaptic neuron, the synaptic cleft. The neurotransmitter molecules diffuse across the synaptic cleft, and bind with specific receptor sites on the cell wall of the postsynaptic neuron. The binding induces a small potential change (typically much less than one millivolt) inside the postsynaptic neuron. At rest, the potential of the inside of a neuron with respect to the outside of the cell is approximately -70 mV. The detailed biochemical mechanisms responsible for these potentials are beyond the scope of this thesis.

There are two types of synapses: excitatory and inhibitory synapses. When an excitatory synapse is activated by an action potential, the neurotransmitters which bind with the cell wall change the properties of the wall so that channels open and allow small ions to freely cross the membrane. A few potassium ions, attracted by the negative potential of the cell, enter the cell, but the high intracellular concentration of potassium opposes this movement. Sodium, however, is attracted both by the negative potential and the low intracellular sodium concentration, and more sodium ions enter the cell. The increase inside the cell of so many positive ions increases the potential of the cell at the synapse, making it more positive. This analog excitatory postsynaptic potential (EPSP) travels away from the synapse. Eventually mechanisms which restore the cell to its resting potential make the EPSP decay to nothing. Inhibitory synapses

have the opposite effect. They may be caused by the opening of potassium channels in the cell wall. This allows the positively charged potassium ions to move out of the cell, which has a high internal concentration of potassium. This makes the cell more negatively charged. The resulting change in voltage is referred to as the inhibitory postsynaptic potential (IPSP). Some inhibitory synapses also work by allowing negative chlorine ions into the cell, or by blocking sodium channels.

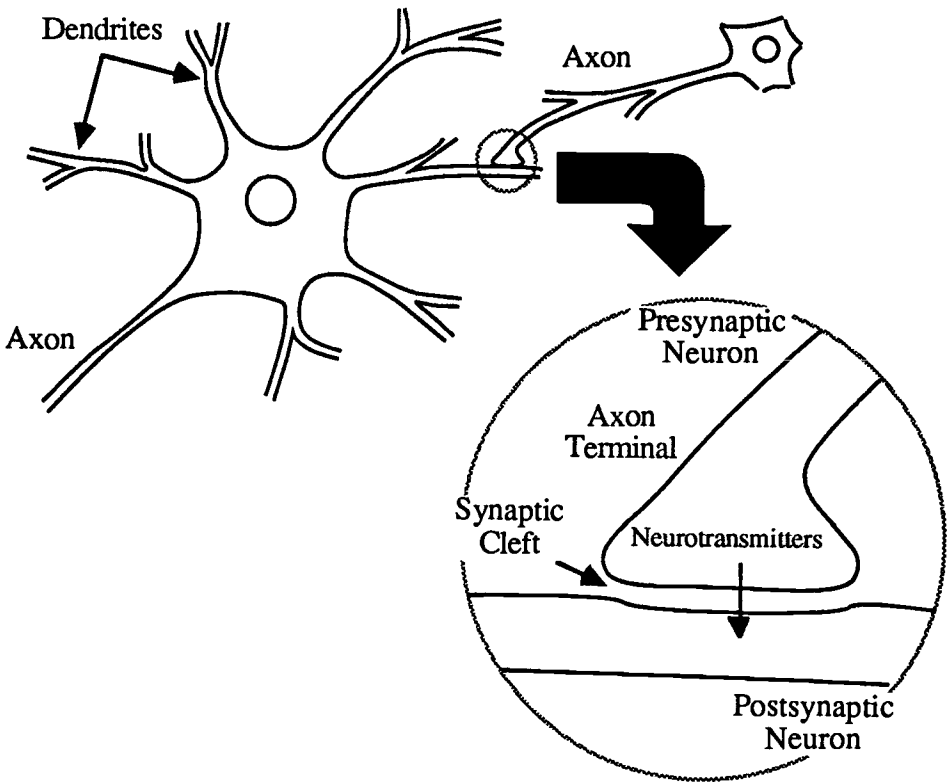

Figure 1: Biological Neurons. Inset shows detailed view of synapse.

This sequence of events, from the presynaptic to postsynaptic potential, takes less than a millisecond. On its own, an EPSP or an IPSP does not have much effect; the potential simply dies away. However, if the several potentials are close enough together in time and space, they will add, so that many EPSPs will produce a larger depolarization of the cell. When the depolarization exceeds a threshold, typically -50 mV to -60 mV, the neuron will fire, generating an action potential. During the action potential, the potential inside the cell increases dramatically as shown in figure 2. The action potential then travels down the axon of the cell to the axon terminals at a velocity of approximately 1 to 100 m/s.

If the stimulus is below the threshold an action potential will not occur, and there will be no transmission at the axon terminals. The action potential is an all-or-none response. Thus, the neuron takes an analog input and converts it to a digital output. After the neuron fires, there is an absolute refractory period during which the neuron cannot fire again, even though an applied stimulus may be above the threshold. This absolute refractory period limits the frequency at which the neuron can fire. However, following the absolute refractory period, a relative refractory period begins. A stimulus significantly above threshold is required to cause firing during this time. A strong applied stimulus from a sense organ or another neuron can decrease the relative refractory period and increase the effective firing frequency of the neuron, if this stimulus is above the threshold value. Therefore the strength of an external stimulus is proportional not to the levels of the individual action potentials (which are constant), but to their frequency. The information carried by a particular neuron is therefore related to the frequency of occurrence of its action potentials.

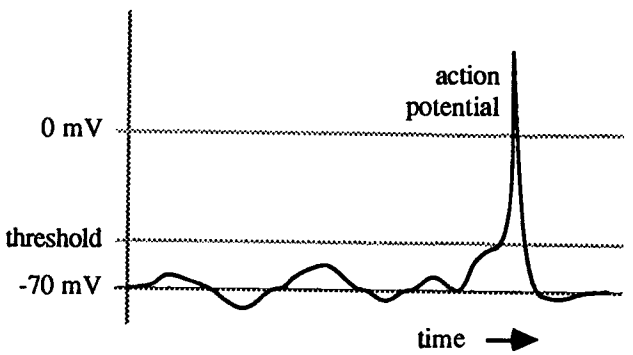

Figure 2: Action Potential. Intracellular potential showing inhibitory activity (below the resting potential of -70 mV), excitatory activity (above the resting potential), and the action potential after neuron is excited above the threshold voltage [adapted from Vander, 1985, p. 211].

BIOLOGICAL NEURAL NETWORKS

One neuron alone obviously cannot fulfill all the functions required of the nervous system. Therefore, all animals have complex networks of neurons which work together to implement complex functions. In the human cortex, there are approximately 10^{10} neurons, and each neuron is connected to approximately 10^4 other neurons. A few

specialized biological neural networks which illustrate important collective network principles are shown below.

MUTUAL INHIBITION

Some fish have a "startle" response, which may be observed by tapping the glass of an aquarium. The fish strongly contract a wall of muscles along one side of their bodies, which causes the tail to flip to one side and makes the fish jump (Bullock, 1977). To be effective, the muscles should only contract on one side of the body. The network shown in figure 3 insures that only one set of muscles contracts.

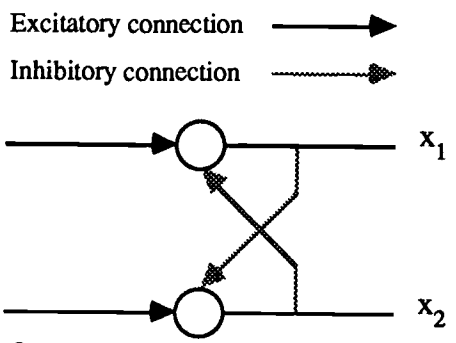

Figure 3: Mutually Inhibitory Neurons

The vibrations in the water reach the fish's ears and stimulate both of the neurons in this network. However, one neuron may receive a stronger stimulus than the other. This neuron will then have a stronger output which will lead to a stronger inhibition of the other neuron. The other neuron will then have a weaker output, which will lead to a weaker inhibition of the first neuron. The less inhibited first neuron will then fire even more strongly. Thus the network amplifies the difference between the incoming signals, ensuring that only one of the outputs is active at a time.

LATERAL INHIBITION

Lateral inhibition is a more complex configuration that occurs in many situations, serving to sharpen the edges of stimuli. It has been extensively studied in the compound eye of the horseshoe crab *limulus*.

The network of figure 4 is one possible configuration illustrating this phenomena. Assume that the stimulus shown at left is light shining on the eye of the crab, with a bright spot in the middle. In the middle of the darker regions, the neurons all receive some level of stimulation from the stimulus, and some inhibition from the two inhibitory neurons on either side. In the middle of the brighter region, the neurons receive

a stronger input from the stimulus, and a stronger inhibition from the two inhibitory neurons. In the middle of both regions, the output from the neurons is fairly constant. The neurons in the darker regions at the edge of the bright spot receive the same stimulation from the input, and some inhibition from their one neighbor in the dark. In addition, they receive a stronger inhibition from their neighbor in the lighter region.

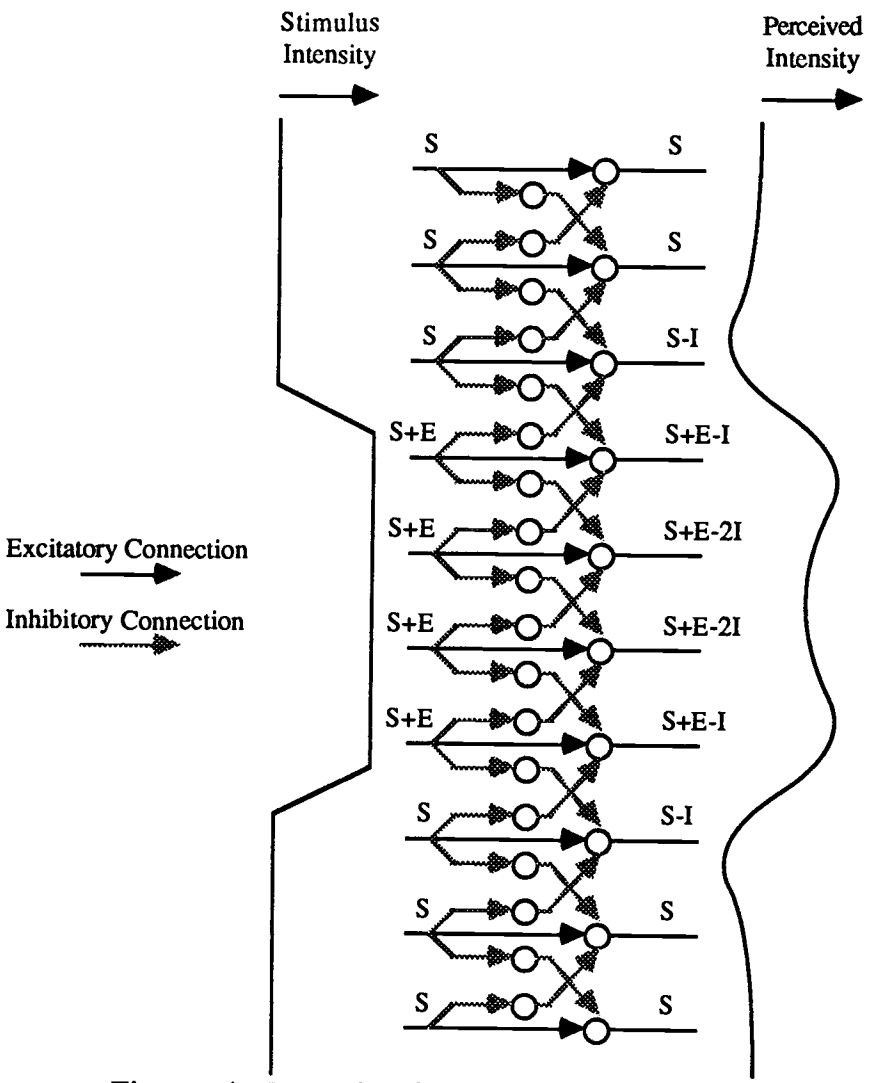

Figure 4: Lateral Inhibition Network. The stimulus level shown at the left passes through the network to obtain the sensation shown at the right [adapted from Bullock, 1977, p. 260].

Neurons in the dark regions at the edge of the bright spot are more strongly inhibited than their neighbors in the dark. Similarly, the neurons in the lighter region at

the edges receive less inhibition than neurons in the middle of the bright spot. Thus, the output from the middle of either region is fairly constant, but the output at the edges is perceived as darker in the dark regions and lighter in the light regions. Therefore, the edges are sharpened by this network so that the contrast appears to be greater.

LEARNING AND MEMORY

In most animals, the nervous system is not completely developed at birth. During infancy many drastic changes take place in the brain as the animal learns to respond to its environment. After adulthood, however, changes in the brain are more subtle than simple cell growth, and are not at all understood. Even so, learning still occurs in adulthood.

At the psychological level, some characteristics of learning and memory stand out (Vander, 1985). Learning may be very rapid, sometimes occurring after only one trial. Learning may be permanent, surviving sleep, trauma, and anesthesia. Memories may not decay with disuse. Memories may become stronger with time. A theory of the biological basis of learning must be able to explain these psychological phenomena.

There are basically two schools of thought on the encoding of memory in the brain (Kohonen, 1988 b). The chemical theory is that memories are stored in long molecular chains, similar to DNA or RNA. The neural theory is that learning and memory result in structural changes in neural networks. Unfortunately, evidence for either theory is inconclusive. Of the two, computer researchers prefer the neural theory, which is easily modelled.

HEBB'S THEORY OF SYNAPTIC CHANGES

The physiologist S. Ramon y Cajal first suggested that changes in the synapses may be responsible for learning. In 1949, D.O. Hebb built on this suggestion.

> Let us assume then that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability. The assumption can be precisely stated as follows: *When an axon of cell* A *is near enough to excite a cell* B *and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that* A*'s efficiency, as one of the cells firing* B*, is increased*....When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell. (Hebb, 1949, pp. 62-63)

> Nowhere ... has there been any assumption as to the permanence of the synaptic connections established by learning.... This is because until now, at least, there has been no decisive reason to assume either that synaptic knobs once formed are permanent or that they deteriorate with disuse....[A] solution may be found if we can adopt ... [the assumption] that an unused connection decays; or, better yet, if we can assume that a very frequently and long-used connection becomes permanent, but that there is a stage in the development of synaptic knobs before which the development is reversible.... One is accustomed to thinking of the nerve cell as a static structure (as it is after being fixed and stained), and of growth as a very slow process. Yet it may be recalled that Cajal (among others) conjectured that the change at the synapse in learning is an ameboid outgrowth of the cell..., which might need very little time for its occurrence.... [I]t is possible to assume, in the light of the psychological evidence, that synaptic decay occurs slowly and perhaps is never quite complete. Thus each repetition of the ameboid change at a particular point might leave a higher residual level, and with a frequent occurrence the outgrowth might become permanent. (pp. 228-231)

According to Hebb, synaptic connections are reinforced with use, and weakened with disuse. Although some physiologists doubt the anatomical evidence supporting this theory, it is very easily modelled using computer techniques. Hebb's theory is utilized to implement learning in many of the artificial neural network models in use today.

NEURON MODELS

Neurons appear to be very simple elements to model. If the summation of the inputs to a neuron is greater than a certain threshold, it fires, otherwise it does not fire. Elements with this property may be defined mathematically, and their operation may be simulated with a computer. With sufficient effort, these elements may be built in analog or digital hardware.

Unfortunately, an artificial neural network may not yet be connected in the same way as the brain, and end up with the same functions as a human. Since there are 10^{10} to 10^{11} neurons in the brain, and each may be connected to 1000 to 100,000 other neurons (Rumelhart, et. al., 1986), the hardware required would be excessive by current standards. It is also extremely difficult to simulate this many neurons and connections with a computer. However, a more manageable number of neurons may still be created in hardware or simulated in order to compute useful neural-like functions, and researchers have created many such networks with interesting abilities.

HISTORICAL DEVELOPMENTS

The first work attempting to define mathematically the properties of networks of simplified neurons was the 1943 paper of Warren S. McCulloch and Walter Pitts. Some of the mathematical precedents seem clear. Russell and Whitehead's *Principia Mathematica* established the necessary formal logic. Alan Turing's 1937 paper, "On Computable Numbers, with an Application to the *Entscheidungsproblem*" defined the Turing machine[1] , and gave some idea of the problems which automata could solve. The computer, in the form of the IBM-Harvard Mark I, had been introduced. War work at Aberdeen Proving Grounds on the ENIAC in the United States and at Bletchley on the ACE in England was well under way, but hardly public knowledge. Nevertheless, it was a time of great interest in computing machinery.

MCCULLOCH-PITTS

The first step McCulloch and Pitts took was "to conceive of the response of any neuron as factually equivalent to a [logical] proposition which proposed its adequate

---

[1] A Turing machine has a reading and writing head, which operates on a tape under it. The machine may read the symbol on the tape under the head, write a symbol on the tape under the head, or move the head in either direction along the tape. Although primitive, the concept of the machine is used to define the computability of problems.

stimulus" (p. 117). They made several simplifying assumptions about their neurons, many of which are still used:

1. The activity of the neuron is an "all-or-none" process.
2. A certain fixed number of synapses must be excited within the period of latent addition in order to excite a neuron at any time, and this number is independent of previous activity and position on the neuron.
3. The only significant delay within the nervous system is synaptic delay.
4. The activity of any inhibitory synapse absolutely prevents excitation of the neuron at that time.
5. The structure of the net does not change with time. (p. 118)

Some McCulloch-Pitts networks are shown in figure 5.[2] The inputs and output of any element may be either 0 or 1. To get the output of an element, the inputs are multiplied by weights and added together. If they exceed the threshold of the neuron, the output is 1; otherwise it is zero. The elements shown are assumed to have thresholds between 1 and 2.

Since these networks can compute AND, OR, and AND-NOT functions, they are, with the addition of a "1" or "true" input, functionally complete, so a combination of them can realize any switching function (if the x_1 input of figure 5 c is replaced by a "true" input, it is a NOT function, and the set of AND, OR, and NOT functions is functionally complete).

With some minor changes, binary threshold elements like these are used in almost all neural network research.

Such a network, they argue, can compute a subset of the computations of a Turing machine. They did not assert that these networks explain the biological nature of the mind, but implied that they are functionally equivalent to the brain. However, it is doubtful that this is true. With the possible exception of the first of the above assumptions, all of them are obviously false. In addition, these networks assume a global synchrony, for which there is absolutely no biological evidence.

Even though their conclusions may have been faulty, the elements they constructed excited much interest.

---

[2] In order to emphasize the continuity with later models, these networks are actually somewhat different from the originals. Their function, however, is the same as the McCulloch-Pitts models.

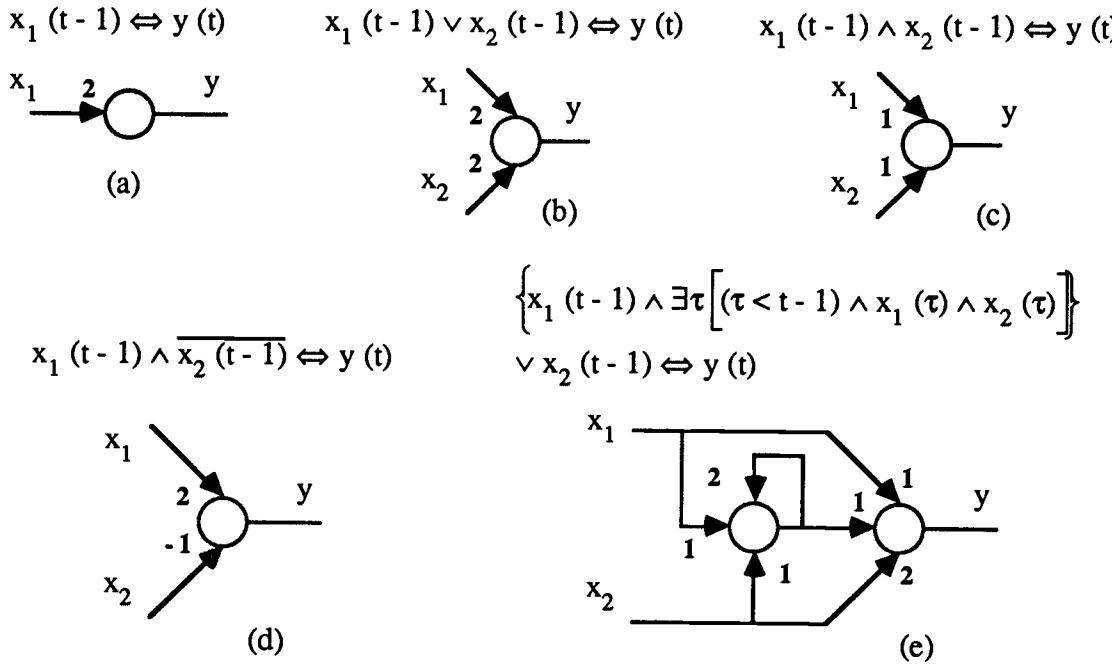

Figure 5: McCulloch-Pitts Networks, showing the logical proposition realized by each network. Weights shown as numbers on input lines. Note that discrete time is assumed, and that an element requires an input of two or more for excitation, where the total input is determined by the sum of the products of the inputs and their weights [adapted from McCulloch and Pitts, 1943, p. 130]. (a) delay network, (b) OR network, (c) AND network, (d) AND-NOT network, (e) existential proposition network. The expression for (e) may be translated as: iff at time t-1 x_2 was true, or at time t-1 x_1 was true and at some time before t-1 both x_1 and x_2 were true, then y is true.

VON NEUMANN

Among those who read McCulloch and Pitts' paper was John Von Neumann. He became interested in the problem of how noisy elements, such as neurons, could reliably compute data (Von Neumann, 1956). Normally, cascading binary gates which each have a certain finite probability of error leads to a probability of error approaching 1/2 when the gates are cascaded deeply enough. Therefore, the output becomes no better than random. Von Neumann showed that under certain conditions, realizing the function redundantly as in figure 6 reduces the probability of error in the output.

This scheme is costly, but it is still important in the design of fault-tolerant switching networks. Von Neumann realized that real neural networks are not quite this simple:

> The problem of understanding the animal nervous system is far deeper than the problem of understanding the mechanism of a computing machine. Even plausible explanations of nervous reaction should be taken with a very large grain of salt. (p. 96)

Still, it has long been thought that at least some of the functions of the nervous system are redundant, and this model led to other models more complex in their redundancy. Von Neumann also saw the importance of the representation of stimulus strength by frequency as opposed to a binary code. The neural method is not as accurate, since analog data is noisier, but the loss of one "bit" of information is not significant as it can be in a binary representation.

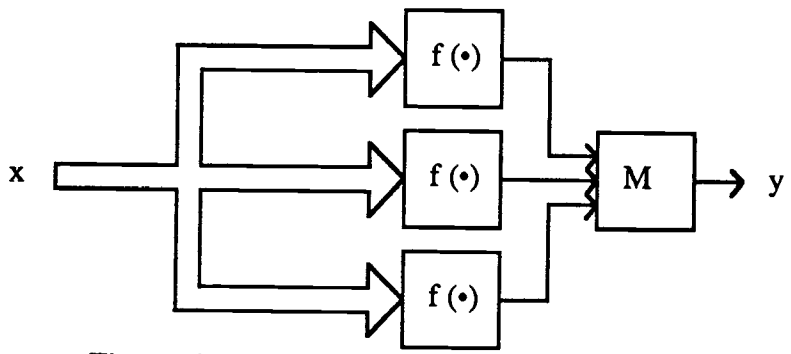

Figure 6: Von Neumann's Redundant Network. Computes the function $y = f(x_1, x_2, \ldots, x_n)$. The function is computed redundantly (three times in this instance), and passed into a majority gate (M) to obtain the output. The output of the majority gate is one if the majority of its inputs are one, otherwise the output is zero.

ROSENBLATT

Today, the first step taken after conceiving the architecture of a new computing machine is to simulate its behavior on another computer. During this era, however, computer time was a valuable commodity not readily available to those involved in developing neural networks. Most of the interesting characteristics of neural networks only arise when a large number of neurons are connected, but it is extremely difficult to trace the behavior of so many elements by hand. It was not until computers became

more available in the late fifties that actual networks were built and well tested. One of the earliest neural networks was Rosenblatt's Perceptron, shown in figure 7 (Rosenblatt, 1958). In the perceptron, an image is projected on the retina, and stimulates cells (neurons) in the projection area. These neurons stimulate random neurons in the association area, which in turn stimulate a response.

Kohonen (1988 b) summarized the output of a response unit, denoted by x_j^R. The output is determined by the outputs from the association units, x_i, and by the weights, w_{ij}, from each association unit to the response unit. The actual relation is

$$x_j^R = \begin{cases} 0 & \text{if } \sum_i x_i w_{ij} < \theta_j \\ 1 & \text{if } \sum_i x_i w_{ij} \geq \theta_j \end{cases}$$

where the variable of summation i ranges over all the association units, and θ_j is a characteristic threshold of the response unit. Thus, one of these units roughly resembles a biological neuron. Both have an all-or-none response, and some finite threshold. The weights of the perceptron unit are like synapses with differing strengths.

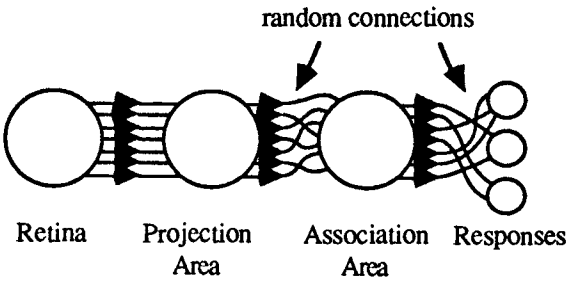

Figure 7: Perceptron. The retinal units have fixed connections to the projection units . The projection units have fixed, but random, connections to the association units. Connections from the association units to the responses are random and modifiable.

The purpose of the perceptron is to learn to recognize particular patterns, and to produce some pre-defined output at the response units for each pattern. For example, pictures of digits could be projected onto the retina, and the output could be the binary representations of the digits. Each pattern presented to the retina produces a particular pattern of activity at the outputs of the association units. Assume that this pattern is some set of binary values for each association unit, $\{x_i\}$. Assume also that for this pattern, the desired output or target value is known for each response unit, and let this

be denoted by t_j. Learning is accomplished by adjusting the weights from the association units to the response units so that the desired response is obtained.

Rosenblatt investigated several ways of adjusting the weights, one of which, the α-reinforcement rule, is described here. If the response output is the same as the target value, no learning is necessary for that unit and no changes are necessary in the weights to that unit. If the target output is "1" and the response output is "0" then the weights from the association units to the response unit should be increased, so that the output will more likely be "1". However, any changes in the weights from association units which are not on ($x_i = 0$) will not have any effect on the response output, so changes should only be made to weights from association units for which $x_i = 1$. Similarly if the target response is "0" and the actual response is "1", the weights to the response unit should be decreased for the association units which are on. These adjustments may be summarized by the equation:

$$\Delta w_{ij} = \lambda \left(t_j - x_j^R \right) x_i,$$

where λ is a positive constant which controls how fast the system learns. It can be seen that this equation satisfies the requirements outlined above.

WIDROW-HOFF

In 1960, Widrow and Hoff created a similar machine called the Adaline, shown in figure 8. The inputs, which may be +1 or -1, are presented by setting switches. The desired output is specified by setting a reference switch. The inputs are multiplied by weights $w_0, w_1, \ldots, w_{n-1}$ (w_0 is the threshold weight, from an element which is always +1) and summed to obtain an output. Subtracting the output from the desired output yields a measure of the error of the system.

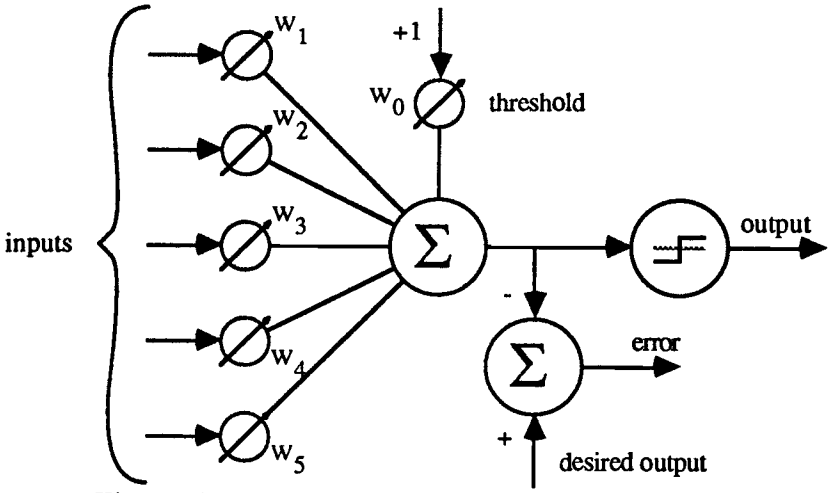

Figure 8: Adaline. [Adapted from Widrow and Hoff (1960), p. 102]

The weights are then adjusted sequentially so that each adjustment reduces the error by 1/n. If the error is γ, the change in the weight w_k on the x_k input is then

$$\text{if } x_k = -1 \text{ then } \Delta w_k = -\frac{\gamma}{n},$$

$$\text{if } x_k = +1 \text{ then } \Delta w_k = +\frac{\gamma}{n}.$$

After changing each of the n weights, the error will be zero, and the output will be the desired output. A second set of inputs may then be applied, and the procedure repeated. The error for the second pattern of inputs will then be zero, and the error for the first pattern will be small but non-zero. If this procedure is repeated for a few patterns many times, the system will eventually find weights which give a very small error for each pattern, and may be said to have "learned" the correct responses to each pattern.

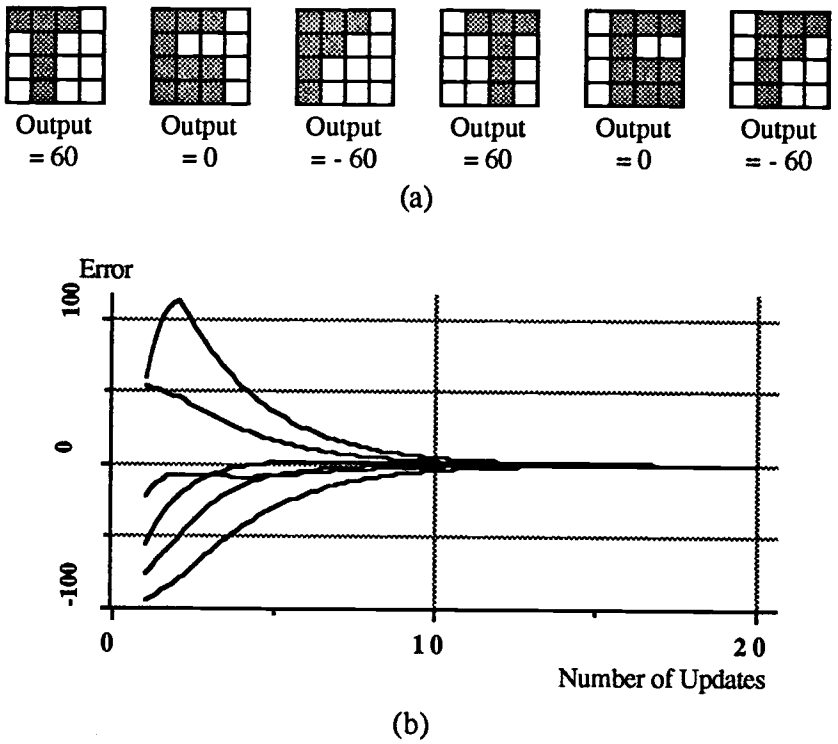

Figure 9: Adaline Learning Problem. (a) Six patterns presented to the machine, and desired output values for each. (b) Error in output of Adaline for each pattern versus number of updates. Error is computed as difference between desired output and actual output. Each update consists of the presentation of each of the six patterns.

One of the problems Widrow and Hoff tested is shown in figure 9. The networks must learn to recognize the patterns, and learn to give the same output for the shifted patterns. Note that the outputs are specified as zero, 60, and - 60, or before the output is quantified (the quantifier was not actually a part of the Adaline built). The system learns fairly quickly, as shown in figure 9 (b).

This system is able to learn many patterns, but is limited by the fact that learnable patterns must be linearly separable functions of the inputs. Widrow and Hoff were not disturbed by this limitation, but it would be shown by Minsky and Papert that it severely restricted the applicability of such neural networks.

MINSKY-PAPERT

Rosenblatt's perceptron inspired many people to work with neural networks, but not Marvin Minsky. He had built a primitive neural network machine at Harvard in

1951, and encouraged by von Neumann, he wrote his Ph.D. thesis on learning in the nervous system. However, "he had become aware of the limitations of such machines, and had come to the conclusion that it was more profitable to concentrate on finding the principles that will make a machine learn than to try building one in the hope that it would work." (Bernstein, 1981, pp. 96-99). Thus he became a leading advocate of artificial intelligence, and an opponent of the neural network approach.

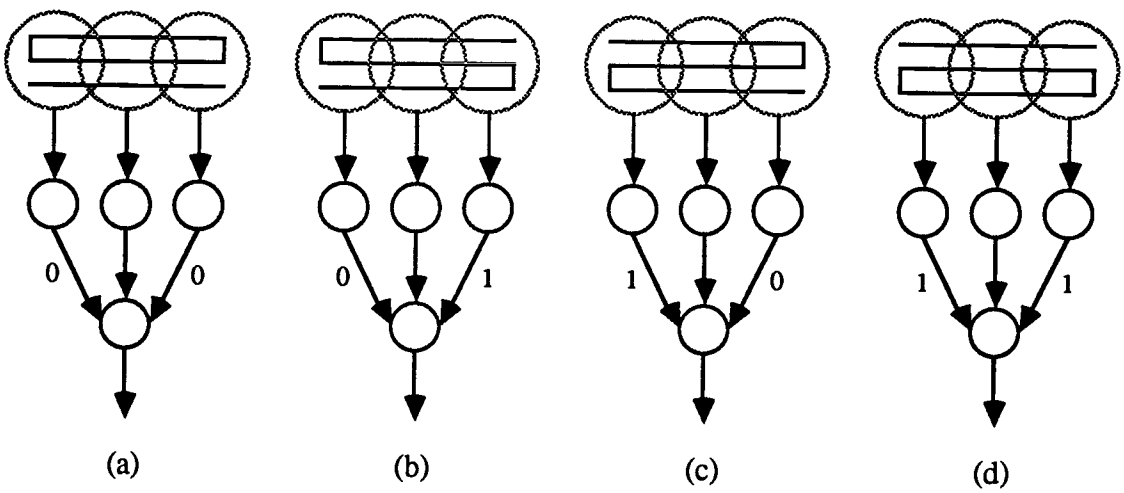

Figure 10: Determining Connectedness. The images projected on the retina are at the top. The gray circles represent the areas connected to each association unit below. (a) and (d) are unconnected; (b) and (c) are connected images. [Adapted from Minsky and Papert, 1969, p. 9]

Together with Seymour Papert, Minsky wrote *Perceptrons*, which proved that the perceptron could not learn to solve an important class of problems. This book did much to discourage research in neural networks by showing the types of problems which Perceptrons and Adalines cannot solve.

One example of such a situation is shown in figure 10. The problem is to determine whether the figure projected onto the retina is connected or not. It is assumed that the association units may only be connected to a limited area of the retina. The leftmost association unit sees one image in figures 10 (a) and 10 (b), and another image in figures 10 (c) and 10 (d). Thus its output must be zero for (a) and (b), and one for (c) and (d), or vice versa. Similarly, the rightmost association unit sees one image in figures 10 (a) and (c), for which the output is zero, and another image in figures 10 (b) and 10 (d), for which the output is one. The output of the response unit at the bottom

must be zero for the unconnected figures 10 (a) and 10 (d), and one for the connected figures 10 (b) and 10 (c).

Unfortunately, the only difference between these figures is at the endpoints, so the response unit must differentiate between inputs of 00 and 11, and inputs of 01 and 10. This is then the exclusive-or problem, and it can be shown that no single threshold element can realize an exclusive-or (Appendix B). According to Minsky and Papert, this problem may arise for any number of levels if the area of the sensory inputs and the number of neurons are limited. For more than two levels, the units between the input and output levels are hidden so it is not apparent how the weights to these units should be changed in order to learn a function.

KOHONEN

Although *Perceptrons* discouraged popular interest in neural networks, a few researchers continued work. Among them was Teuvo Kohonen, who defined the theoretical foundation of associative memory (Kohonen, 1972, 1988 a, 1988 b).

In a human brain, partial data may be used in order to retrieve specific memories. For example, a person might use the data "a man with long blond hair who wears glasses" in order to retrieve more data, like the name of the man. In a conventional computer memory, the only way to retrieve the memory is to find the address of it, which may involve searching all the memories until a match is found. Unlike a computer, a person has a content-addressable memory (CAM) in which data may be retrieved by using partial data as the key.

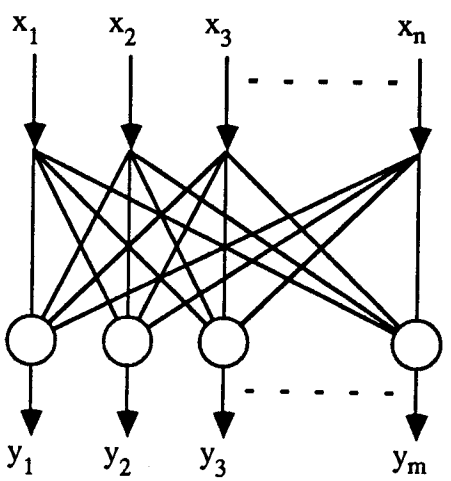

Figure 11: Correlation Matrix Memory.

According to Kohonen, all neural networks may be viewed as associative memories. Given some input key, the network generates "memorized" output data. If the key is part of the data, the network is an auto-associative memory, or CAM. If the key is different from the data, the network is a hetero-associative memory. Suppose a network is used as a pattern completion device, where it is given an incomplete or noisy pattern as input, and it generates a complete pattern. This is an auto-associative memory. If instead it is given a pattern as input, and generates an unrelated output (as in the Adaline problem of figure 9) it is a hetero-associative memory.

One of the first associative memories Kohonen studied was the correlation matrix memory. Assume a network structure like that shown in figure 11, with n inputs $x_1, x_2, \ldots, x_n$, and m outputs $y_1, y_2, \ldots, y_m$. Each output is determined by the neural element transfer function.[3]

$$y_i = \sum_j w_{ij} x_j.$$

The neural elements assumed here are not binary, but rather have a linear, unthresholded response.

The output of the entire network may be expressed as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_m \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

or

$$\mathbf{y} = \mathbf{W}\,\mathbf{x}$$

where the boldface letters denote vectors and matrices.

Then the problem is to find one set of weights which will generate the output pattern for each input pattern. Define a matrix $\mathbf{W}_\Sigma$

$$\mathbf{W}_\Sigma \equiv c \sum_p \mathbf{y}_p \mathbf{x}_p^T,$$

where $\mathbf{x}_p$ and $\mathbf{y}_p$ denote particular input and output pattern vectors, the superscript T denotes the transpose of the vector, and the sum is over all the patterns to be stored in the memory. Then the output for a particular input pattern $\mathbf{x}_k$ is

$$\hat{\mathbf{y}}_k = \mathbf{W}_\Sigma \mathbf{x}_k$$

---

[3] The term w_{ij} in this section means the weight from x_j to y_i, in order to be consistent with the usual matrix notation. Throughout the rest of the text, the term w_{ij} means the weight from element i to element j.

$$\hat{y}_k = c\left(\sum_p y_p\, x_p^T\right) x_k$$

$$\hat{y}_k = c\, y_k \|x_k\|^2 + c \sum_{p \neq k} y_p\, x_p^T\, x_k .$$

Then the output of the system will approximate the memory

$$\hat{y}_k \cong y_k$$

if the following are true:

$$c \equiv \|x_k\|^{-2}$$

$$x_p^T\, x_k \cong 0,\ p \neq k.$$

The second condition means that the input patterns should be nearly orthogonal, or as different from each other as possible. If they are too similar, the output of the network will be affected by this "cross-talk" between patterns.

Even if this error is small, the number of connections causes problems. The size of the matrix W_Σ is $n \times m$, where n and m are the number of inputs and outputs. Fortunately, it is possible to implement the network even if it is not fully connected, using a number s of random connections, where

$$s < m\, n.$$

Kohonen proved that the standard deviation of an output due to the incomplete connections is

$$\sigma = \sqrt{\frac{mn}{s} - 1}\ \frac{\sqrt{\sum_{i=1}^{m}\left(\sum_p x_{ip}\, x_{ir}\, y_{jp}\right)^2}}{\sum_p \sum_{i=1}^{m} x_{ip}\, x_{ir}\, y_{jp}},$$

where x_{ip} is the ith element of pattern vector x_p and y_{jp} is the output of interest. The standard deviation approaches zero as s approaches m n, as expected. For a particular system, if an acceptable standard deviation is known, this equation may be solved for s, the minimum number of connections required.

HOPFIELD

At least part of the current resurgence of interest in neural networks is due to John J. Hopfield. Although others have claimed that he contributed no new ideas, his influence

in the field is indisputable. His work began when he defined an auto-associative memory algorithm (1982).

Like the Kohonen associative memory, the Hopfield network shown in figure 12 has one neuron for each output. Unlike the Kohonen associative memory, the outputs of the network are fed back to the input. Given a set of binary vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_s\}$ (the memories of the system), where each vector represents the desired state of the neurons $\mathbf{x}_i = [x_1, x_2, \ldots, x_n]^T$, the weight between neurons i and j may be determined by

$$w_{ij} = \begin{cases} \sum_{p=1}^{s} (2\, x_{(p)i} - 1)\,(2\, x_{(p)j} - 1) & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \qquad [1]$$

where $x_{(p)i}$ denotes the x_i element of memory x_p. This means that two neurons which are both on or both off in most of the memory vectors will tend to excite each other, and neurons which have different values will tend to inhibit each other. Also, the weights are always symmetric. The binary neurons are updated asynchronously according to the threshold function:

$$x_i = \begin{cases} 0 & \text{if } \sum_j w_{ji}\, x_j < \theta_i; \\ 1 & \text{if } \sum_j w_{ji}\, x_j > \theta_i. \end{cases} \qquad [2]$$

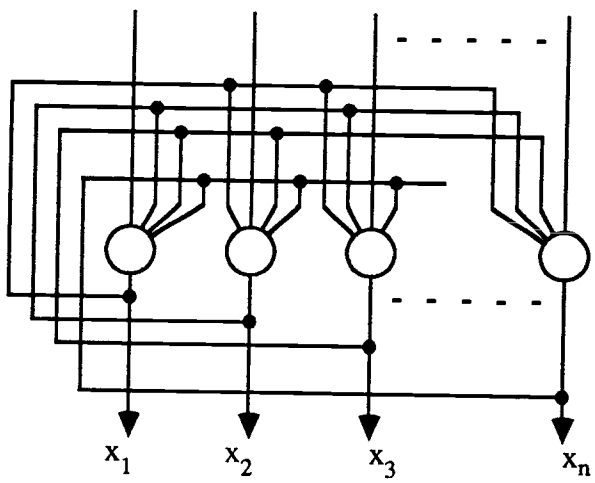

Figure 12: Hopfield Associative Memory.

The energy of the system is defined by Hopfield as:

$$E = -\frac{1}{2} \sum_i \sum_j w_{ji}\, x_i\, x_j + \sum_i x_i\, \theta_i. \qquad [3]$$

If one of the elements x_k changes, the change in energy (see Appendix B) will be

$$\Delta E = -\left(\sum_j w_{jk}\, x_j - \theta_k\right)\Delta x_k. \qquad [4]$$

If the change in x_k is positive, then from equation 2 the term in brackets will be positive, so the change in E will be negative. Similarly, if the change in x_k is negative, then the change in E will also be negative. Later, Hopfield proved that this also holds for continuous threshold functions (1984).

The energy of the system may be viewed as a terrain in n dimensions (Tank and Hopfield, 1987). The state of the system is then a ball which starts out at a some point in the terrain, and rolls downhill to the nearest minimum. The starting point is set by the initial values of each neuron. In a content addressable memory, some part of the memory (the key) is known, and is used to retrieve the entire memory. The initial values of the neurons are set to the key. The memories stored in the system should be minima of the terrain, and the key should start the network close enough to the desired minimum so that the state rolls down into that minimum.

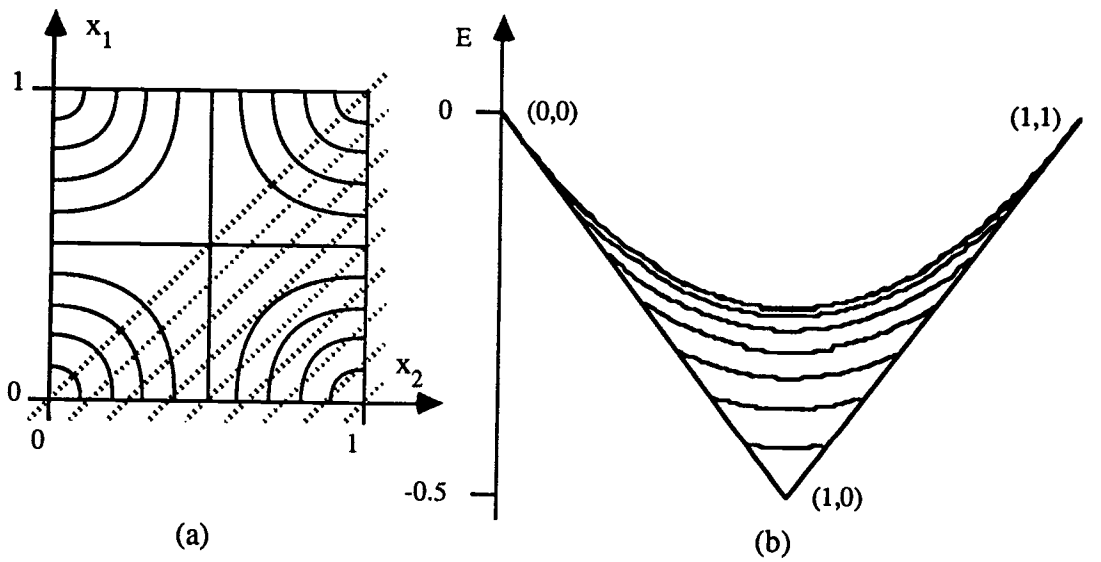

Figure 13: Energy Surface. (a) Equipotential hyperbolas in x_1 x_2 coordinate plane. (b) View of parabolas formed by plotting along the dotted lines of figure (a). The minimum at (0, 1) is behind that at (1, 0), so it does not appear.

To understand how the energy surfaces work, consider the mutual inhibition network of figure 3, where the output of each neuron is fed back to inhibit the other neuron. The energy surface for this network is shown in figure 13. The maximum

energy is when the neuron outputs are equal: $(x_1, x_2) = (0, 0)$ and $(x_1, x_2) = (1, 1)$. The minima are at the points where the outputs are not equal: $(x_1, x_2) = (1, 0)$ and $(x_1, x_2) = (0, 1)$. The surface is shaped like a saddle, so if the network starts out with an energy on either side of the top, the states will "roll" down to the closest minimum. If it starts out at a state where the inputs to the two neurons are equal, however, the state is balanced on the top of the ridge, and cannot roll down to a minimum.

For more complex networks, the energy surface is more intricate. For n neurons, the surface has n+1 dimensions, so it cannot be easily visualized. However, the concept is used to understand how networks converge, and how learning algorithms can reshape the surface so that the desired output of the network corresponds to a minimum of the surface.

There are, of course, problems with this convergence procedure. If the minima are too close together, the network may easily be attracted to the wrong minima. Too many memories also increased the number of errors in the final state; Hopfield found that the number of memories should be less than 0.15 n. The other problem is that of local minima: since the energy of the system cannot increase, the system may become stuck in a local minimum. Later procedures were developed which allow the system to escape local minima.

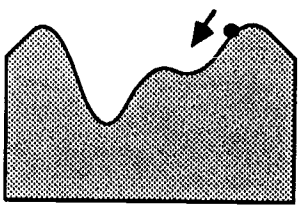

Figure 14: Minimization of Energy Function. The state of the system may be viewed as a ball, which will roll down to the closest minimum.

Unless the memories are completely opposite each other, the algorithm stores the opposite of every memory just as strongly as the memory itself. Some memories may be stronger than others, so that it is nearly impossible to get the network to converge to anything else (Hopfield called this "the '50% of all stimuli remind me of sex' problem" [Hopfield, Feinstein, and Palmer, 1983]). Later procedures, like the Boltzman machine (Ackley, Hinton, and Sejnowski, 1985), correct some of these problems.

BACK-PROPAGATION

A very powerful learning procedure called back-propagation was developed in Rumelhart, Hinton, and Williams (1986), and Rumelhart, McClelland, and the PDP Research Group (1986). Like Widrow and Hoff's Adaline, the procedure changes the weight at the outputs to minimize a sum-squared error function. Unlike the Adaline, the networks may contain several layers of elements with modifiable connections.

A back-propagation network contains a first layer of input units, a last layer of output units, and one or more layers of hidden units between the input and output layers. In addition, there is one unit the output of which is always on which is connected to all the other units, and which may be viewed as a modifiable threshold for each unit. The outputs of a layer may be connected to higher layers, but not to lower layers, so it is a feed-forward network. Since back-propagation networks may have many layers, they do not have the limitations pointed out by Minsky and Papert regarding Perceptrons. But with hidden layers in the network, it is not obvious how to change the weights to elements in these layers in order to minimize the output error. This is the credit assignment problem, and its solution makes back-propagation an extremely useful algorithm.

The output of each unit is not strictly zero or one, but is some continuous function of the total input to the unit. Let the total input to a unit j be

$$\sigma_j = \sum_i x_i \, w_{ij}.$$

Then the output of unit j should be a continuous, monotonically increasing function of the inputs, which ranges between zero and one. The usual function is

$$x_j = f(\sigma_j) = \frac{1}{1 + e^{-\sigma_j}},$$

shown in figure 15.

The network is first presented with one of several stimulus patterns at the input layer. The signals propagate forward through the network, resulting in some output at the output layer. Then, starting at the output layer, error signals are calculated for each unit. For an output unit i, the error is

$$\delta_i = (t_i - x_i) \, x_i \, (1 - x_i),$$

where t_i is the desired output for unit i (see Appendix B for derivation). The desired output for hidden units is not known, however. The error for a hidden unit i is

$$\delta_i = x_i \, (1 - x_i) \sum_k \delta_k \, w_{ik},$$

so it is a function of the weights from it to other units and the error signal of those units. Therefore, the error is computed for the output units first, then propagated backward through the network.

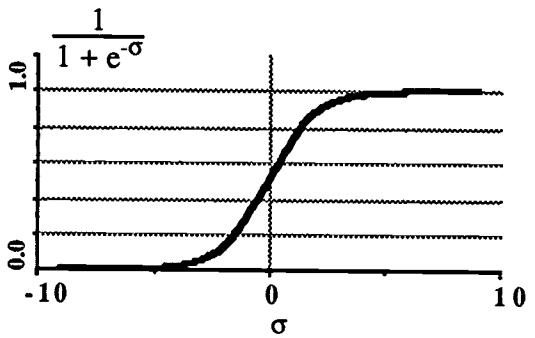

Figure 15: Sigmoid Threshold Function.

After the error signals have been calculated, the weights are changed by an amount

$$\Delta w_{ij} = \eta \delta_j x_i,$$

where η is a constant between zero and one, which controls the learning rate: how quickly the network converges to a solution. The weight from unit i to unit j then depends on the output of unit i and the error signal of unit j. If η is large, the network learns quickly, but it may oscillate. Viewing the error of the system as a surface again, the system may become stuck in a local minimum. To avoid this, a momentum term is added to the change in the weights:

$$\Delta w_{ij}(n) = \eta \delta_j x_i + \alpha \Delta w_{ij}(n-1),$$

where α is a constant between zero and one controlling the momentum. Then the change in a weight also depends on the last change made in the weight.

Figure 16 shows two networks for realizing an exclusive-or (XOR) network. Thresholds for each element are learned by making a threshold unit which is always on. The weights from the threshold unit to the other units are the thresholds for those units. The weights are initially set to small random values, and the weights shown are learned after 160 iterations (each iteration consists of the presentation and back-propagation of each of the four possible input patterns). For both networks, $\alpha = 0.9$ and $\eta = 0.8$.

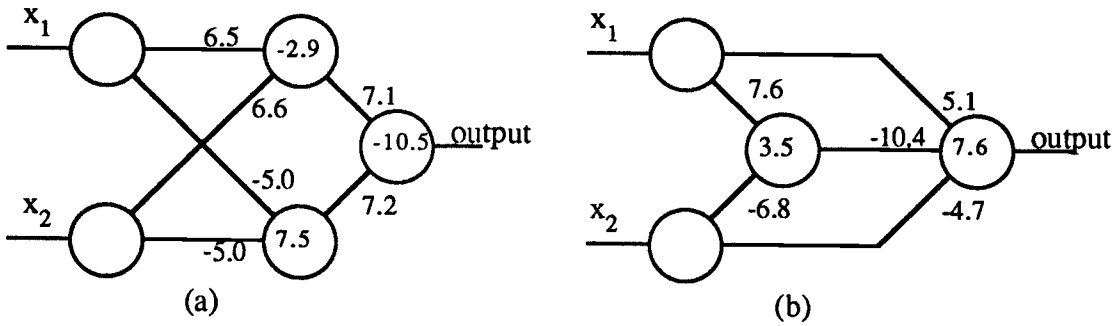

Figure 16: Back-Propagation XOR Networks. Weights from threshold unit shown inside elements.

Figure 17 shows the output of both of the networks versus the number of iterations. Convergence to the correct output was assumed if the output was within 0.1 of the correct output (above 0.9 for outputs of 1, and below 0.1 for outputs of 0). The network of figure 16 (a) converged within about 60 iterations, and is typical of the performance of the back-propagation algorithm. The outputs oscillate for the first 20 or 30 iterations, while the weights change in many directions. Then the outputs change quickly toward the correct outputs when the appropriate direction of change for each weight is found. As the outputs converge toward the correct outputs, the δ's for each element become smaller, and learning slows down. By contrast, the network of figure 16 (b) took almost 160 iterations to converge. The outputs first settle to around 0.5 before starting to converge.

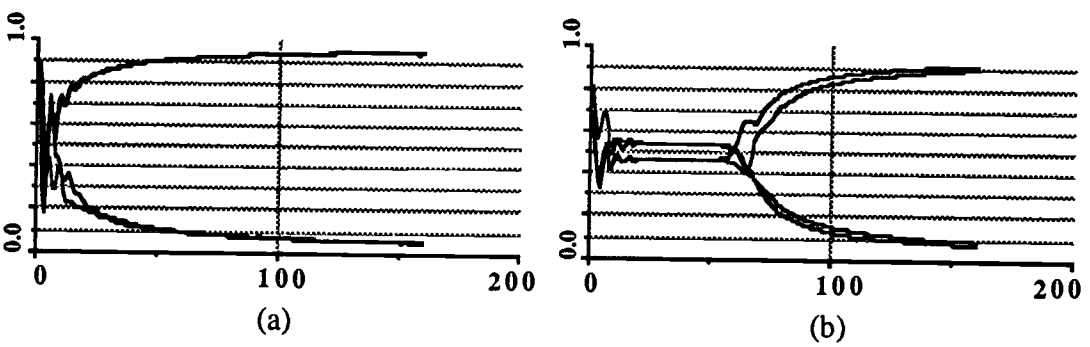

Figure 17: Convergence of Back-Propagation XOR Networks. The output for each of the four inputs shown separately as they converge to correct outputs. (a) shows the convergence for the network of figure 16(a); (b) shows that of figure 16(b).

These results agree with those of Rumelhart, McClelland, et. al. (1986). Networks with more hidden units tend to converge more quickly. However, not all networks will converge for different initial values of the weights and for different values of α and η. Therefore, there is no guarantee that a given network will converge.

Even without guaranteed convergence, experience suggests that it offers a fairly reliable method for learning in complex networks. In contrast to some other algorithms like the Boltzmann machine, it learns quickly. These factors make it one of the most useful algorithms for neural networks.

SUMMARY OF NEURAL NETWORK ALGORITHMS

Starting with McCulloch and Pitts, the field of neural networks has grown and diverged rapidly. In the present, there are so many algorithms that it would be impossible to summarize all of them, or even to cover every branch of research. A few common elements appear, like the minimization of an error or energy function. The usual network has a large number of simple, highly interconnected neural elements, but the neuron models are quite varied. It is apparent that there is no unifying theory of neural networks, and some of the algorithms have few common elements. For a given problem, there is no straightforward way to design a network to solve it.

The basic elements of neural networks are models of biological neurons. There are many differences between these models and actual neurons, some of which are described in Appendix D. Most of the recent networks make no attempt to model biological neural networks. The connection of the artificial neural elements into networks is not necessarily inspired by biological networks. Physiologists often criticize these artificial networks, since many of them bear little relation to biological neural networks. But the plausibility of artificial neural networks is beside the point if they are useful.

NEURAL NETWORK HARDWARE IMPLEMENTATIONS

Despite the differences in the algorithms, most use highly interconnected networks of simple elements. Several hardware implementations have been built which realize these networks more or less successfully.

STRAIGHTFORWARD IMPLEMENTATION

A straightforward hardware implementation of a neural network is found in Graf, Hubbard, Jackel, and deVegvar (1987) and Graf, Jackel, and Hubbard (1988). Resistive coupling elements connect 54 amplifiers, representing 54 neurons. Like biological neurons, the circuit is part analog and part digital. The chip, implemented in 2.5 μm VLSI CMOS technology, measures 6.7 mm × 6.7 mm. Almost 90% of the area of the chip was used for interconnection. Since the neuron elements are fully connected, and the algorithms may execute in parallel, the running time of an algorithm does not depend on the number of neurons. Excluding the time taken to read in and write out the data, one iteration of an algorithm runs in only 1-2 μs.

Despite the impressive speed of this circuit, there are some problems with it. Some algorithms, as will be shown below, cannot be implemented with the limited resolution of the analog data. This is disappointing, but many other algorithms which can be implemented using this circuit will work with the available analog data resolution. More crippling, perhaps, is the area required for interconnections. It is clear that the interconnection area will grow much faster than the number of neurons when more amplifiers are added, since adding one neuron will add n interconnections, where n is the total number of neurons. Thus, a circuit of this type may only contain a relatively small number of neurons.

SWITCHED-CAPACITOR IMPLEMENTATION

The brain has a distinct advantage over conventional computers in terms of the number of processing elements available, and the number of interconnections between the processing elements. Neurons are also very slow, with a delay on the order of milliseconds — about a million times slower than logic gates. It has been suggested by Cleary (1987) that neural network circuits could trade off some of the speed of CMOS logic gates for an increased number of interconnections by multiplexing the updating of the neuron parameters in time.

A possible configuration for a time-multiplexed neural network circuit is shown in figure 18. The operation of the circuit is as follows: The neurons are first initialized. Each neuron is then selected, in turn, to be updated. During the update phase for each neuron, all other neuron output values are sampled, and fed back through a multiplying digital-to-analog converter (MDAC), where they are multiplied by the appropriate weight. The result is added to the state of the neuron currently being updated.

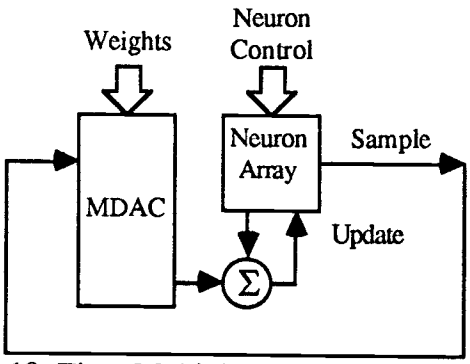

Figure 18: Time-Multiplexed Network.

A switched-capacitor implementation of the time-multiplexed neural network circuit (Hansen, Skelton, and Allstot, 1989) was built and is shown in figure 19. The array of capacitors and switches on the left constitutes the MDAC. Each capacitor in the array at the right represents a neuron. The charge stored on the capacitor represents the state of the neuron. The capacitor is sampled on ϕ_1 of the clock and fed through the MDAC. On ϕ_2 of the clock, the neuron to be updated is selected and the signal from the MDAC is either added to or subtracted from the state value.

The neural network circuit shown requires external control. An external microprocessor is used for this purpose. Control words, including the clocking signals, are sent to the neural network via the microprocessor output port. Figure 20 shows a possible format for the control words. The control signal PHI is equivalent to the circuit control ϕ_1, and ϕ_2 is generated as the complement of ϕ_1. The signal SH controls the switch across the op-amp used for feedback. The signal DEC enables the decoder, which takes the encoded signals $N_0, N_1, \ldots, N_m$ and generates the appropriate neuron control n_i. The SEL signal controls the mux. When SEL is low, the circuit input comes from the analog input, which at this time is connected to a high logic level. When SEL is high, the circuit input comes from the output of the switched-capacitor network. The LAT signal controls the serial to parallel converter. The LAT signal is activated after

each neuron is updated, latching the circuit output. Finally the weight signals W_0, W_1, ..., W_k control the weight switches. The resultant binary setting of the switches specifies the numeric value of a weight. The value is between zero and one.

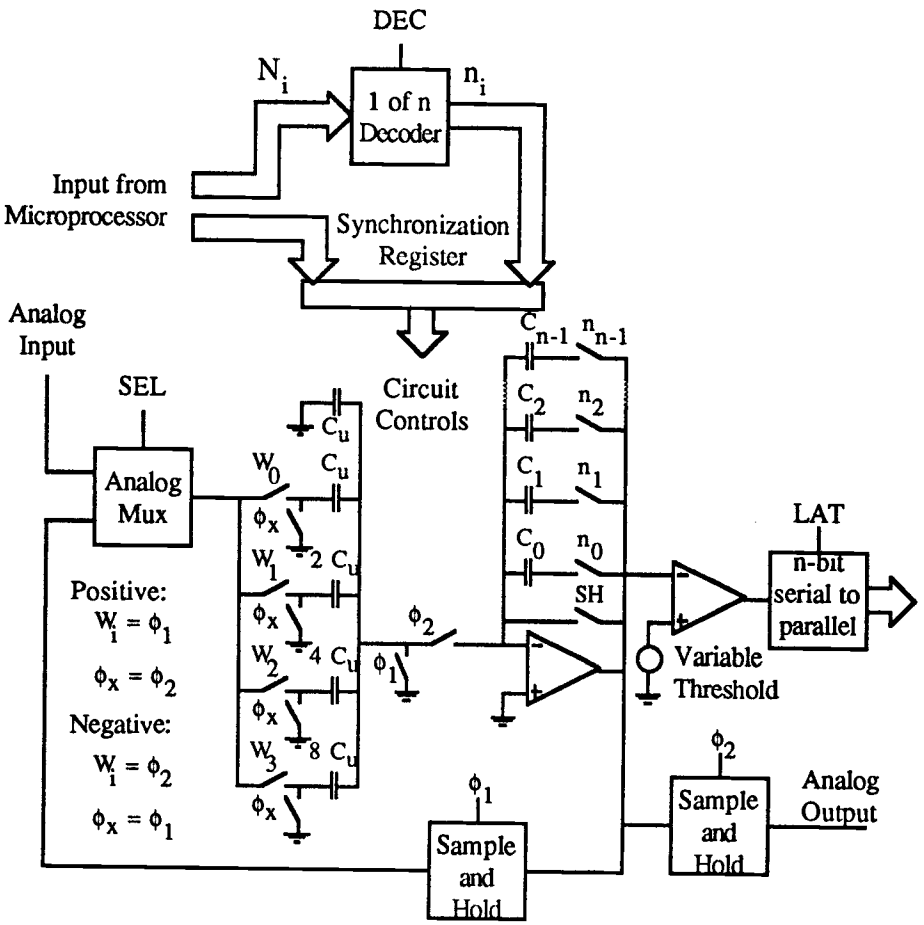

Figure 19: Switched-Capacitor Network.

Figure 21 shows the timing of a complete operational cycle for eight neurons. The weights for this case are specified by one switch only (W_0). A cycle consists of three main phases: Initialize, Update, and Discharge. During the initialize phase, the circuit charges the neuron capacitors to their initial values. SEL is low for the entire phase, so the circuit input comes from the analog input line. PHI alternates between zero and one on successive cycles. The neurons and the appropriate weights are selected when PHI is low. In this case, neurons 0, 5, 6, and 7 are initialized to states of +1, and neurons 1, 2, 3, and 4 are initialized to states of zero.

During the update phase, each neuron is selected in turn. The other neurons are sampled, their output values multiplied by the appropriate weights, and the products are

added to the selected neuron. During each sample, PHI is high, and the sampled neuron is selected. PHI then goes low, and the updated neuron is selected. Finally, PHI is low again, and the decoder is disabled. During this time the short switch is closed. This activity helps to discharge parasitic capacitances. The timing of the weight switches determines the sign of the weighting factor. If the weight signals are high when PHI is high initially, the sampled neuron's weighted output is added to the updated neuron. If the weights are high when PHI is low, the sampled neuron's weighted output is subtracted from the updated neuron. On the last clock phase of each neuron update, the LAT signal is set high to latch the output of the circuit, which is the final value of the updated neuron.

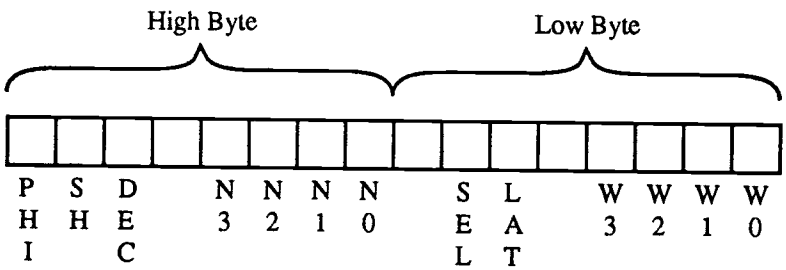

Figure 20: Pin Assignment for Output Port.

As an example, consider Update 0 in figure 21. First neuron 0 is updated by subtracting the values of neurons 2 and 7, and adding the values of neurons 4 and 5. Neuron 2 is selected on the first clock phase when PHI = 1, then neuron 0 is selected and the weight signal is set high on the second clock phase, subtracting the value of neuron 2 from neuron 0. After the third cycle when the SH signal is high, PHI is again high. Neuron 4 is selected, and the weight signal is high. On the second clock phase, neuron 0 is selected, so the value of neuron 4 is added to neuron 0. Similarly, neuron 5 is added, and neuron 0 is subtracted. On the last clock phase of the update of neuron 0, the LAT signal is set high. The other neurons are updated in the same way.

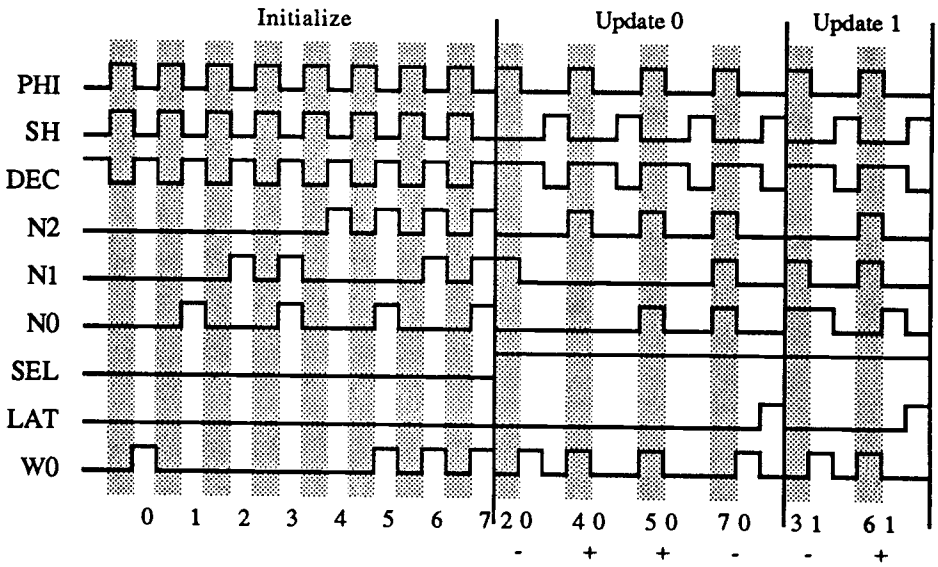

Figure 21: Circuit Timing.

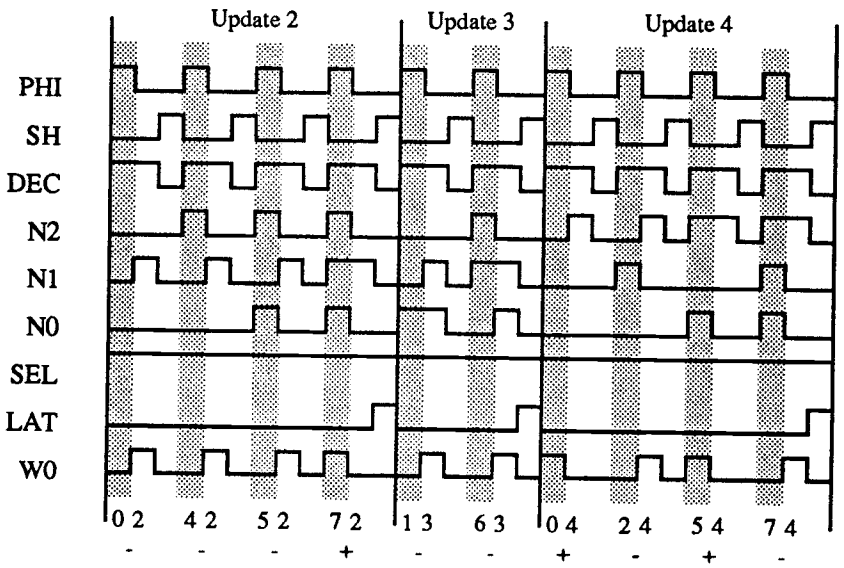

Figure 21: Circuit Timing, continued.

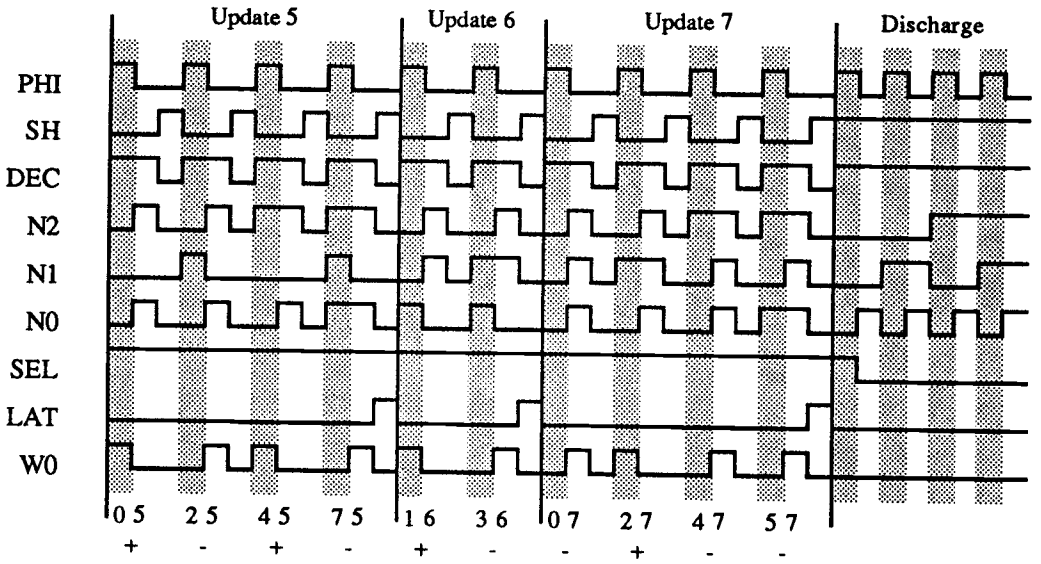

Figure 21: Circuit Timing, continued.

After all the neurons have been updated, the microprocessor should read in the final state of each neuron from the output latch. This feature has not yet been implemented. The state of the system is determined by reading the state of n light-emitting diodes connected to the output of the latch. The neural network capacitors are then discharged.

SYSTEM TEST

The circuit was tested with Hopfield's associative memory algorithm. The results of these tests are contained in Appendix A. The control words necessary to execute this algorithm were sent to the circuit via the microprocessor. The Hopfield network algorithm has severe drawbacks. A key often converges to an incorrect or spurious memory, and the algorithm has no learning capability. However, it has the advantage of being fairly easy to implement. Since the circuit is limited by the speed of the controlling microprocessor, all of the control words are stored in memory. The controlling program then simply fetches the control words and outputs them to the circuit.

The circuit differs somewhat from the Hopfield model. First, the op-amp saturates at an analog level corresponding to a state of ±2 (or ±10 V, since 5 V represents a state of 1). This means that the sum of equation [2] is not represented completely accurately. For example, the sequence 1 + 1 + 1 + 1 - 1 - 1 - 1 would evaluate to -1, instead of +1. Part of this problem could be avoided by changing the scaling factor and having 2.5 V or 1.25 V represent 1. However, for circuits containing more neurons, the problem would eventually occur. Another difference is that the capacitors, once updated, store the sum of equation [2], which ranges between ±2. This tends to amplify the effect of

the previously updated numbers. The effect of these differences is not devastating, perhaps because the algorithm is fairly fault-tolerant. In fact, some noise actually helps the algorithm converge to the correct result, because it tends to prevent it from settling into a local minimum. Therefore, in spite of these differences, the circuit can realize the Hopfield algorithm fairly well.

Probably the most commonly implemented algorithm is the Hopfield associative memory algorithm, because the neural elements required are so simple. But the algorithm itself is not very useful, because of the problems with it. For an implementation to be practical, it should be able to run neural network algorithms which converge more reliably, and which can learn.

Back-propagation is a more useful neural network algorithm, because it learns and is more reliable. However, simulation results have shown that this algorithm will probably not work if implemented on this circuit. The problem has to do with analog signal accuracy and dynamic range. On a conventional digital computer, back-propagation can be simulated. Assuming a floating point word size of 32 bits, the smallest representable number is about 10^{-38}. In contrast, the smallest representable analog signal in the circuit implemented is on the order of 10^{-4}. This alone will not prevent back-propagation from working, although it may not work as well. Another limitation is that the weights must be limited in range, in this case to ± 1. The combination of these limitations prevents the algorithm from performing correctly, by forcing the weights into a small region, and limiting the smallest change that they can make. This limits the usefulness of this circuit, but it also makes a point about back-propagation. The algorithm is very sensitive to noise and signal range constraints. Some implementations of back propagation cannot tolerate the noise present in analog signals. Biological neurons, however, do use analog signals, have a limited range of signal levels, and are noisy. Thus an algorithm which will not work in the presence of these factors cannot work with real neurons.

Since the circuit was developed with the constraint that it have the accuracy of biological neurons, and back-propagation was not, it is perhaps not too surprising that back-propagation will not work with this circuit. It also points out a fundamental conflict in neural network research. Those with a biological background tend to insist that the simulated networks have properties like those found in nature. But those with more background in computers see no reason why a good algorithm should be "natural," and model abstract elements which may be very different from biological neurons.

Although this circuit may not be able to perform back-propagation, there is no reason why it cannot perform learning algorithms. The weights are external to the circuit, and thus are easily changed. But an algorithm must take into account the limitations of biological neurons in order to work on this circuit.

CONCLUSIONS

At the start of neural network development, researchers saw many parallels between the computer and the brain. Both are made of basic switching elements — the neuron in the brain, and the vacuum tube or electromechanical relay in early computers. However, important differences appeared. The brain does not seem to be organized as simply as the computer.

Some of the important developments in neural network research have been described. From these it is apparent that neural network research has diverged from simply modelling biological networks, although they may have some of the desirable characteristics found in the brain. From the hardware implementations described, it is apparent that straightforward neural network implementations are generally not feasible, and that some method for saving chip area is required.

For this reason, a unique time-multiplexed neural network implementation was developed, and a sixteen-neuron breadboard version was tested. This circuit is one method for realizing a large number of neurons in a small area.

The Hopfield associative memory worked well on this circuit, and demonstrated the robustness of this algorithm. Its implementation pointed out some of the characteristics this algorithm has in common with biological networks, particularly its noise insensitivity. Another advantage is its distributed representation of information. Like the brain, if a few of the neural connections are destroyed, the network may still be functional.

The results of the simulations of back-propagation were unexpected, but they reveal some important limitations of the algorithm. Like biological neurons, in this circuit, the range of the weights is limited and the analog signals are inaccurate. These factors destroy the performance of back-propagation. Indeed, although any implementation must have limited weights, almost no algorithms take this into account.[4]

In the future, some extensions could be made to this time-multiplexed neural network circuit implementation which would make it more useful. At the present time, its speed is limited by the controlling microprocessor. A faster specialized system could be developed to control the circuit more efficiently, and it could be interfaced with a standard microcomputer. Aside from hardware improvements, better neural network

[4] An exception is Kohonen (1988 b).

algorithms may be developed which combine the speed and learning ability of back-propagation with the robustness of the Hopfield algorithm.

BIBLIOGRAPHY

Ackley, D.H., Hinton, G.E., and Sejnowski, T.J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science* **9**:1, 147-169.

Anderson, J.A., and Rosenfeld, E. (1988). *Neurocomputing: Foundations of Research.* MIT Press (Cambridge, MA).

Bailey, J., and Hammerstrom, D. (1988). Why VLSI implementations of associative VLCN's require connection multiplexing. *IEEE Second Annual Neural Network Conference*, **2**, 173-180.

Ballard, D.H., Hinton, G.E., and Sejnowski, T.J. (1983). Parallel visual computation. *Nature* **306**: 3, 21-26.

Bernstein, J. (1981). Profiles: AI. *New Yorker* (December 14, 1981), 50-126.

Bullock, T.H., Orkland, R., and Grinnell, A. (1977). *Introduction to Nervous Systems*. W.H. Freeman and Company (San Francisco, CA).

Cleary, J.G. (1987). A simple VLSI connectionist architecture. *IEEE First Annual Neural Network Conference*, **3**, 419-426.

Cowan, J.D., and Sharp, D.H. (1988). Neural nets and artificial intelligence. *Dædalus* **117**:1, 85-121.

Graf, H.P, Hubbard, W., Jackel, L.D., and deVegvar, P.G.N. (1987). A CMOS Associative Memory Chip. *IEEE First Annual Neural Network Conference*, **3**, 461-467.

Graf, H.P,. Jackel, L.D., and Hubbard, W.E. (1988). VLSI Implementation of a Neural Network Model. *Computer* **21**:3, 41-49.

Gregorian, R., and Temes, G.C. (1986) *Analog MOS Integrated Circuits for Signal Processing*. John Wiley and Sons (New York, NY).

Hansen, J.E., Skelton, J.K., and Allstot,D.J. (1989). A time-multiplexed switched-capacitor circuit for neural network applications. *Proceedings of the 1989 IEEE International Symposium on Circuits and Systems,* Portland, OR.

Hebb, D.O. (1949). *The Organization of Behavior.* John Wiley & Sons (New York).

Hinton, G.E., and Sejnowski, T.J. (1983). Optimal perceptual inference. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 448-453.

Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA* **79**, 2554-2558.

Hopfield, J.J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences, USA* **81**, 3088-3092.

Hopfield, J.J., and Tank, D.W. (1985). "Neural" computation of decisions in optimization problems. *Biological Cybernetics* **52**, 141-152.

Hopfield, J.J., and Tank, D.W. (1986). Computing with neural circuits: A model. *Science* **233**, 625-633.

Hopfield, J.J., Feinstein, D.I., and Palmer, R.G. (1983). "Unlearning" has a stabilizing effect in collective memories. *Nature* **304**:14, 158-159.

Kohonen, T. (1972). Correlation matrix memories. *IEEE Transactions on Computers* **C-21**, 353-359.

Kohonen, T. (1988 a). An introduction to neural computing. *Neural Networks* **1**: 1, 3-16.

Kohonen, T. (1988 b). *Self-Organization and Associative Memory,* 2nd ed. Springer-Verlag (Berlin).

Kung, S.Y., and Hwang, J.N. (1987) Systolic designs for state space models: Kalman filtering and neural network. *Proceedings of the 26th IEEE Conference on Decision and Control* **3**, 1461-1467.

Levy, D., and Newborn, M. (1982). *All About Computers and Chess.* Computer Science Press (Rockville, Maryland).

McCulloch, W.S., and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **5**, 115-133.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry.* MIT Press (Cambridge, MA).

Rosenblatt, F.(1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* **65**: 6, 386-408.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). Learning representations by back-propagating errors. *Nature* **323**: 9, 533-536.

Rumelhart, D.E., McClelland, J.L., and the PDP Research Group (1986). *Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Volume 1: Foundations.* MIT Press (Cambridge, MA).

Short, K.L. (1981). *Microprocessors and Programmed Logic.* Prentice-Hall (Englewood Cliffs, NJ).

Tank, D.W., and Hopfield, J.J. (1987). Collective computation in neuronlike circuits. *Scientific American* **257**, 104-108+.

Vander, A.J., Sherman, J.H., and Luciano, D.S. (1985). *Human Physiology,* 4th ed. McGraw-Hill (New York).

Von Neumann, J. (1956). Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies,* C.E. Shannon and J. McCarthy, ed. Princeton University Press (Princeton, NJ), 43-116.

Welsh, D.E. (1984). *Computer Chess.* William C. Brown (Dubuque, Iowa).

Widrow, B., and Hoff, M.E. (1960). Adaptive switching circuits. *IRE Convention Record,* **4**: part 4, Los Angeles, 96-104.

APPENDICES

APPENDIX A: TEST RESULTS

The system was tested by programming the controlling microprocessor to run the Hopfield associative memory algorithm, with sixteen neurons. Two randomly generated memories were stored in the system, and each neuron was initialized to a random starting state of zero or one. Each of the neurons was then updated once. Some of the test results are shown below. The numbers in parentheses after the memories indicate the number of differences between the memories and the initial state of the neurons, giving a measure of how close the starting state is to each of the memories.

Test 1

memory 1:	0100 1100 1101 0110 (8)
memory 2:	0110 1101 1000 0000 (8)
initial:	1111 0001 0101 0010
simulation:	0000 0010 0010 1101
	1001 0010 0111 1011
	1001 0010 0111 1111
circuit:	1001 0010 0111 1111

The initial state differs from both the memories in eight of the sixteen neurons, so it is not surprising that the system converges to neither memory. The simulation of the ideal network takes three updates of all the neurons. The circuit output after one update is the same as the final simulated output, perhaps because the capacitors store the unthresholded states, and thus the effect of the already updated neurons on later updates is amplified.

Test 2

memory 1:	1001 0001 0010 0010 (4)
memory 2:	1110 1101 1011 1110 (7)
initial:	1010 0001 0010 1011
simulation:	1110 1101 1011 1110
circuit:	1110 1101 1011 1110

In this case both the simulation and the circuit converge immediately to memory 2, even though memory 1 is closer to the initial state.

Test 3

memory 1:	1010 0011 0010 1111 (9)
memory 2:	1001 1000 1111 1110 (8)
initial:	0011 0100 1100 0111
simulation:	1000 1000 0011 1000
	1001 1000 1111 1110
circuit:	1001 1000 1111 1110

The simulation converges after two updates, the circuit after one. Both converge to memory 2.

Test 4
memory 1: 0110 0011 1100 1110 (8)
memory 2: 1000 0011 0010 0001 (8)
initial: 0000 1010 0001 1111
simulation: 0111 0100 1100 1110
0111 1100 1101 1110
circuit: 0111 1100 1101 1110

As in Test 1, both the circuit and the simulation converge to the opposite of one of the memories.

Test 5
memory 1: 0001 0111 1010 0100 (9)
memory 2: 1101 1111 0100 0101 (6)
initial: 1110 1001 0010 0101
simulation: 1101 1110 0100 0001
1101 1111 0100 0101
circuit: 1101 1111 0100 0101

Both the circuit and the simulation converge to memory 2.

Test 6
memory 1: 0110 0101 1110 1001 (7)
memory 2: 1110 0010 0000 0001 (7)
initial: 1100 0101 1001 0000
simulation: 0000 0101 1110 1000
circuit: 0001 1101 1111 1110

In this case, because of the differences between the circuit and the Hopfield model, the simulation converges to a different output than the circuit.

Test 7
memory 1: 1100 1111 0110 0111 (8)
memory 2: 0100 0001 1000 1111 (8)
initial: 1001 1001 1011 0011
simulation: 1100 1110 0110 0100
1100 1111 0110 0111
circuit: 1100 1111 0110 0111

Both the circuit and the simulation converge to memory 1.

Test 8
memory 1: 1101 0011 0111 1111 (7)
memory 2: 1010 0001 0110 0111 (11)
initial: 1101 1111 1100 1100
simulation: 0101 0010 0011 1011
1101 0011 0111 1111

circuit: 1101 0011 0111 1111

The circuit and the simulation converge to memory 1.

Test 9
memory 1: 0000 0000 0110 1101 (5)
memory 2: 1100 1111 0111 1100 (5)
initial: 1110 0010 0111 1101
simulation: 1100 1111 0111 1100
circuit: 1100 1111 0110 1100

The circuit differed in one output from the simulation. Taking into account the saturation of the op-amp, the circuit should give the same output as the simulation. However, when the neurons are sampled, some of the charge on the neuron capacitors leaks off due to the parasitic capacitance on the neuron switches. In this case, by the time neuron 11 is updated, some of the neurons have changed from one to zero because of this leakage, so the circuit output for neuron 11 is zero instead of one. This problem occurs rarely, however, and the parasitic capacitances should not be as bad for the integrated version of this circuit as for the breadboard.

APPENDIX B: DERIVATIONS

LINEAR SEPARABILITY

For a logical function to be realizable with a single threshold element, it must be linearly separable. Assume a threshold element has two inputs, x_0 and x_1. The four possible inputs may be thought of as vertices of a square in 2 dimensional space, as shown in figure A 1.

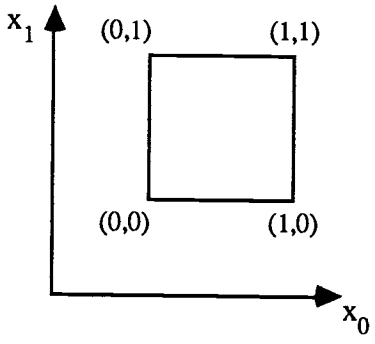

Figure A 1. Inputs to Threshold Element. The vertices of the square correspond to inputs to the threshold element.

The output of the element, y, depends of the inputs, the weights, and on the threshold θ:

$$\text{if } x_0 w_0 + x_1 w_1 < \theta, \text{ then } y = 0,$$
$$\text{if } x_0 w_0 + x_1 w_1 > \theta, \text{ then } y = 1.$$

Assume that the sum is equal to the threshold:

$$x_0 w_0 + x_1 w_1 = \theta,$$

which is an equation for a line in 2 dimensions. At every point above the line, the output is 1, and at every point below the line the output is zero. Thus the function to be realized must be such that all the points for which the function is one lie on one side or another of the line. Otherwise, the function cannot be realized by a single threshold element.

One function which is not linearly separable is the exclusive-or (XOR) function. In this case, the points for which the function is one — (0,1) and (1,0) — lie on opposite vertices of the square, and no single line can separate them from both the other vertices.

When there are n inputs, the square becomes an n-dimensional cube, and the line separating the vertices becomes a plane in n dimensions. Then, for a function to be

linearly separable, all the outputs for which it is one must lie on one side of a separating plane.

HOPFIELD'S ENERGY DECREASE

As given above, the update rule for a binary neuron is:

$$x_i = \begin{cases} 0 & \text{if } \sum_j x_j w_{ij} < \theta_i; \\ 1 & \text{if } \sum_j x_j w_{ij} > \theta_i. \end{cases} \quad [1]$$

The energy of a network is:

$$E = -\frac{1}{2}\sum_i \sum_j x_i x_j w_{ij} + \sum_i x_i \theta_i. \quad [2]$$

For a Hopfield network, the following are assumed:

$$w_{ij} = w_{ji} \quad [3]$$

$$w_{ij} = 0 \text{ if } i = j. \quad [4]$$

To see the effect of a change in one of the neurons x_k, factor out all the terms containing x_k in the energy equation:

$$E = -\frac{1}{2}\sum_i x_i x_k w_{ik} - \frac{1}{2}\sum_j x_k x_j w_{kj} + x_k \theta_k - \frac{1}{2}\sum_{i\neq k}\sum_{j\neq k} x_i x_j w_{ij} + \sum_{i\neq k} x_i \theta_i.$$

But the first two terms may be combined since $w_{ij} = w_{ji}$, so

$$E = -\sum_i x_i x_k w_{ik} + x_k \theta_k - \frac{1}{2}\sum_{i\neq k}\sum_{j\neq k} x_i x_j w_{ij} + \sum_{i\neq k} x_i \theta_i,$$

$$E = x_k\left(\theta_k - \sum_i x_i w_{ik}\right) - \frac{1}{2}\sum_{i\neq k}\sum_{j\neq k} x_i x_j w_{ij} + \sum_{i\neq k} x_i \theta_i. \quad [5]$$

Evaluating E for $x_k = 0$ and $x_k = 1$ gives

$$E(x_k = 0) = -\frac{1}{2}\sum_{i\neq k}\sum_{j\neq k} x_i x_j w_{ij} + \sum_{i\neq k} x_i \theta_i, \quad [6]$$

$$E(x_k = 1) = \theta_k - \sum_i x_i w_{ik} - \frac{1}{2}\sum_{i\neq k}\sum_{j\neq k} x_i x_j w_{ij} + \sum_{i\neq k} x_i \theta_i. \quad [7]$$

Assume x_k changes from 0 to 1. Then the change in energy may be found by subtracting equation [6] from equation [7], to give

$$\Delta E = \theta_k - \sum_i x_i w_{ik}.$$

But if $x_k = 1$, then the second inequality of equation [1] must hold:

$$\sum_i x_i w_{ik} > \theta_k,$$

$$0 > \theta_k - \sum_i x_i w_{ik},$$

$$0 > \Delta E.$$

Now assume x_k changes from 1 to 0. Then the change in energy may be found by subtracting equation [7] from equation [6], to give

$$\Delta E = \sum_i x_i w_{ik} - \theta_k.$$

But if $x_k = 0$, then the first inequality of equation [1] must hold:

$$\sum_i x_i w_{ik} < \theta_k,$$

$$\sum_i x_i w_{ik} - \theta_k < 0,$$

$$\Delta E < 0.$$

Therefore, any change in one of the neurons results in a decrease in the energy of the system.

BACK-PROPAGATION DELTA RULE

The delta rule minimizes the square of the differences between the outputs of the network and the target values of those outputs. First, some definitions are necessary. The error for a particular pattern p is

$$\Gamma_p = \frac{1}{2} \sum_i (t_{i(p)} - x_{i(p)})^2 \qquad [8]$$

where $t_{i(p)}$ is the desired output of that unit for the stimulus pattern p, and $x_{i(p)}$ is the actual output. The total input to the unit x_i is

$$\sigma_i = \sum_j x_j w_{ij}, \qquad [9]$$

and the output of the unit obtained through the threshold function

$$x_i = \frac{1}{1 + e^{-\sigma_i}}, \qquad [10]$$

although other threshold functions may be used. Solving equation [10] for $e^{-\sigma_i}$ gives:

$$e^{-\sigma_i} = \frac{1}{x_i} - 1. \qquad [11]$$

The goal of the delta rule is to change the weights so that the change implements a gradient descent of the error function. Therefore, the weights must be changed by an amount:

$$\Delta w_{ji} \propto - \frac{\partial \Gamma}{\partial w_{ji}},$$

$$\Delta w_{ji} = - \eta \frac{\partial \Gamma}{\partial w_{ji}},$$

where η is a constant between zero and one.

First we must find the change in error due to a change in the weight on an input:

$$\frac{\partial \Gamma}{\partial w_{ji}} = \frac{\partial \Gamma}{\partial x_i} \frac{\partial x_i}{\partial w_{ji}} \qquad [12]$$

Differentiating equation [8] gives the change in the error with respect to output i (suppressing the p subscript):

$$\frac{\partial \Gamma}{\partial x_i} = -(t_i - x_i).$$

Using the chain rule again gives:

$$\frac{\partial x_i}{\partial w_{ji}} = \frac{\partial x_i}{\partial \sigma_i} \frac{\partial \sigma_i}{\partial w_{ji}}.$$

Differentiating equation [10], and substituting equation [11] back in gives

$$\frac{d x_i}{d \sigma_i} = \frac{e^{-\sigma_i}}{(1 + e^{-\sigma_i})^2},$$

$$\frac{d x_i}{d \sigma_i} = x_i (1 - x_i).$$

Now δ_i may be defined as

$$\delta_i \equiv -\frac{\partial \Gamma}{\partial \sigma_i} = (t_i - x_i)\, x_i\, (1 - x_i). \qquad [13]$$

Differentiating equation [9] gives

$$\frac{\partial \sigma_i}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_k x_k\, w_{ki} = x_j.$$

Then the change in the weights should be proportional to the change in the sum-squared error due to the change in the weights:

$$\Delta w_{ji} = -\eta \frac{\partial \Gamma}{\partial \sigma_i} \frac{\partial \sigma_i}{\partial w_{ji}} = \eta\, \delta_i\, x_j. \qquad [14]$$

Equation [13] defines δ_i for an output unit, one for which the desired value is known. However, the network may contain hidden units, for which the desired value is unknown. Then

$$\delta_i = -\frac{\partial \Gamma}{\partial \sigma_i} = -\frac{\partial \Gamma}{\partial x_i} \frac{d x_i}{d \sigma_i},$$

$$\delta_i = -x_i (1 - x_i) \frac{\partial \Gamma}{\partial x_i}.$$

$$\frac{\partial \Gamma}{\partial x_i} = \sum_k \frac{\partial \Gamma}{\partial \sigma_k} \frac{\partial \sigma_k}{\partial x_i} = \sum_k \frac{\partial \Gamma}{\partial \sigma_k} \frac{\partial}{\partial x_i} \left(\sum_j x_j\, w_{jk} \right)$$

$$\frac{\partial \Gamma}{\partial x_i} = \sum_k \frac{\partial \Gamma}{\partial \sigma_k} w_{ik} = -\sum_k \delta_k\, w_{ik}.$$

So,

$$\delta_i = x_i (1 - x_i) \sum_k \delta_k w_{ik}. \qquad [15]$$

Together, equations [13], [14], and [15] define the delta rule for back-propagation.

SWITCHED-CAPACITOR CIRCUIT ANALYSIS

Figure A 2 shows an equivalent circuit for the analysis of the switched-capacitor circuit structure. For an ideal op-amp, the input resistance is infinite so there is no current into the op-amp at node A, and the voltage difference between the two inputs to the op-amp is zero. Thus, node A is at virtual ground. The sign of the signal added to the voltage across the capacitor C_I is determined by the timing of the switches W and ϕ_X. If W is closed on ϕ_1, and ϕ_X is closed on ϕ_2, the sign is positive. If W is closed on ϕ_2, and ϕ_X is closed on ϕ_1, the sign is negative.

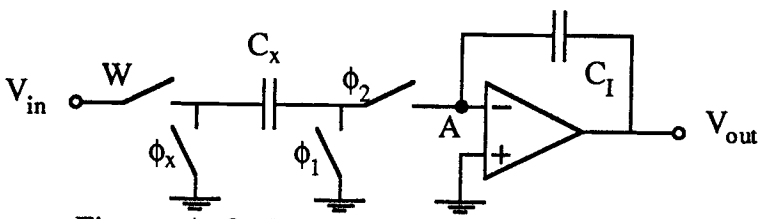

Figure A 2. Equivalent Switched-Capacitor Circuit.

Figure A 3 shows the equivalent circuits for the positive case. During clock phase ϕ_1, capacitor C_X is charged to V_{in}. During clock phase ϕ_2, summing the currents at node A gives:

$$s C_X (V_A + V_{in}) + s C_I [(V_A - (V_{out} - V_I)],$$

but node A is virtual ground, so V_A is zero. Solving for V_{out} gives:

$$V_{out} = V_I + \frac{C_X}{C_I} V_{in}.$$

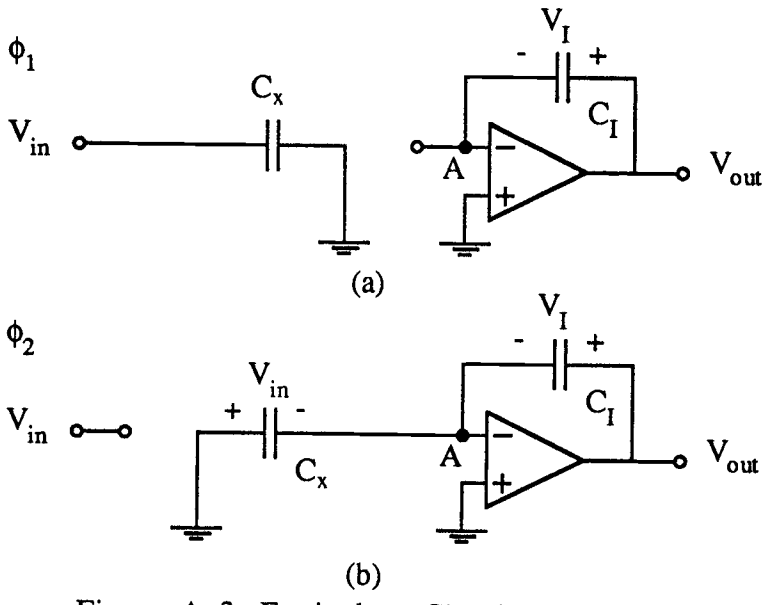

Figure A 3. Equivalent Circuits for Positive Multiplication. (a) Equivalent circuit for clock phase ϕ_1. (b) Equivalent circuit for clock phase ϕ_2.

Figure A 4 shows the equivalent circuits for the negative case. During clock phase ϕ_1, capacitor C_x is discharged. During clock phase ϕ_2, summing the currents at node A gives

$$s\,C_x\,(V_A - V_{in}) + s\,C_I\,[(V_A - (V_{out} - V_I)],$$

but node A is virtual ground, so V_A is zero. Solving for V_{out} gives

$$V_{out} = V_I - \frac{C_x}{C_I} V_{in}.$$

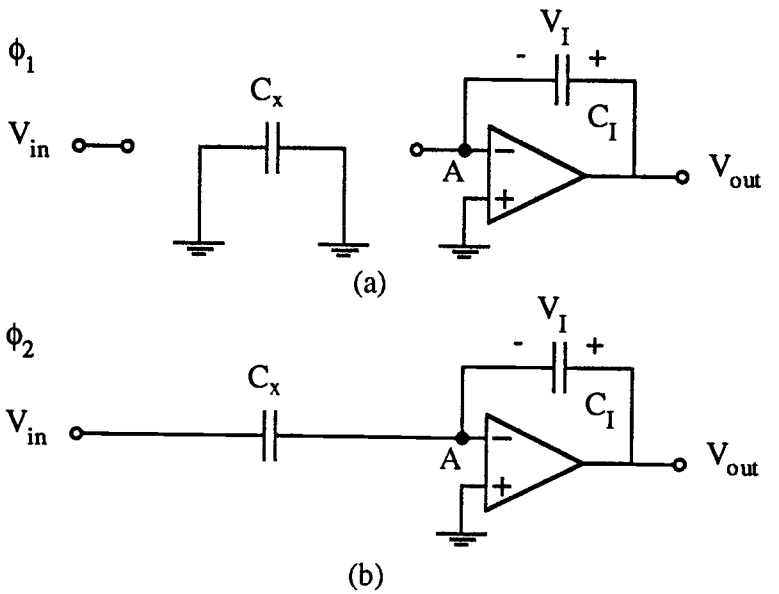

Figure A 3. Equivalent Circuits for Negative Multiplication. (a) Equivalent circuit for clock phase ϕ_1. (b) Equivalent circuit for clock phase ϕ_2.

Thus the voltage across capacitor C_I is changed by an amount $\pm (C_x/C_I)\, V_{in}$. The capacitor C_x is the parallel combination of the capacitors in the MDAC, the value of which may be changed by setting the weight switches. The total value of the capacitors is

$$C_x = W_0\, C_u + W_1\, 2\, C_u + W_2\, 4\, C_u + \ldots + W_{k-1}\, 2^{k-1}\, C_u = C_u \sum_{i=0}^{k-1} W_i\, 2^i.$$

Then if

$$C_I = 2^k\, C_u,$$

the change in the signal across C_I will be

$$\pm \sum_{i=0}^{k-1} W_i\, 2^{i-k}.$$

APPENDIX C: ASSEMBLY PROGRAM FOR GENERATING CONTROL WORDS

Following is an assembly language program for the Intel 8085A to generate the control words for the switched-capacitor circuit.

```
TABLE:  DS      110H    ; FIRST 16 BYTES
                        ; = INITIAL COND.
                        ; NEXT 256 BYTES
                        ; = WEIGHT ARRAY IN
                        ; ROW MAJOR FORM
INIT1   EQU     C000H   ; INIT. CONSTANTS
INIT2   EQU     2000H   ;
UPDT1   EQU     A040H   ; UPDATE CONSTANTS
UPDT2   EQU     2040H   ;
UPDT3   EQU     4040H   ;
DSCHG   EQU     6000H   ; DISCHARGE CONSTANT
PORTH   EQU     1000H   ; ADDRESS OF HIGH BYTE
                        ; OF OUTPUT PORT
PORTL   EQU     1000H   ; ADDRESS OF LOW BYTE
                        ; OF OUTPUT PORT
N       EQU     10H     ; NUMBER OF NEURONS
                        ;
                        ;       INIT. PHASE
                        ;
START:  LXI     HL,TABLE ;(3)   HL: TABLE ADDR.
        MVI     B,00H   ;(2)    B: NEUR. CNTER
                        ;
LOOP1:  LXI     DE,INIT1 ;(3)   1ST CLK PHASE
        MOV     A,D     ;(1)    HIGH BYTE
        OUT     PORTH   ;(3)
        MOV     A,E     ;(1)    OUTPUT LOW BYTE
        OUT     PORTL   ;(3)
                        ;
        LXI     DE,INIT2 ;(3)   2ND CLK PHASE
        MOV     A,D     ;(1)    HIGH BYTE
        ORA     B       ;(1)    SELECT NEURON
        OUT     PORTH   ;(3)    OUTPUT HIGH
        INR     B       ;(1)
        MOV     A,E     ;(1)    LOW BYTE
        ORA     M       ;(2)    SELECT WEIGHT
                        ;       IN MEMORY
        OUT     PORTL   ;(3)    OUTPUT LOW BYTE
        INX     HL      ;(1)
                        ;
        MOV     A,B     ;(1)
        CPI     N       ;(2)    COMPUTE B-N
        JNZ     LOOP1   ;(2/3)  LOOP TERMINATES
                        ;       ON B=N
                        ;
                        ;       UPDATE PHASE
                        ;
        MVI     B,00H   ;(2)    B CONTAINS
                        ;       NUMBER OF
                        ;       UPDATED NEURON
LOOP2:  MVI     C,00H   ;(2)    C CONTAINS
                        ;       NUMBER OF
                        ;       SAMPLED NEURON
LOOP3:  MVI     A,00H   ;(2)
        CMP     M       ;(2)    COMPUTE -W FOR
                        ;       EACH WEIGHT
        JP      NEG     ;(2/3)  NEGATIVE WEIGHT
        JZ      ZERO    ;(2/3)  ZERO WEIGHT --
                        ;       DO NOTHING
                        ;
POS:    LXI     DE,UPDT1 ;(3)   1ST CLOCK PHASE
        MOV     A,D     ;(1)    HIGH BYTE
        ORA     C       ;(1)    SAMPLED NEURON
        OUT     PORTH   ;(3)    OUTPUT HIGH
        MOV     A,E     ;(1)    LOW BYTE
        ORA     M       ;(2)    WEIGHT FROM
                        ;       SAMPLED NEURON
        OUT     PORTL   ;(3)    OUTPUT LOW BYTE
                        ;
        LXI     UPDT2   ;(3)    2ND CLOCK PHASE
        MOV     A,D     ;(1)    HIGH BYTE
        ORA     B       ;(1)    UPDATED NEURON
        OUT     PORTH   ;(3)    OUTPUT HIGH
        MOV     A,E     ;(1)    LOW BYTE
        OUT     PORTL   ;(3)    OUTPUT LOW BYTE
                        ;
        LXI     UPDT3   ;(3)    3RD CLOCK PHASE
        MOV     A,D     ;(1)    HIGH BYTE
        OUT     PORTH   ;(3)    OUTPUT HIGH
        MOV     A,E     ;(1)    LOW BYTE
        OUT     PORTL   ;(3)    OUTPUT LOW BYTE
        JMP     ZERO    ;(3)
                        ;
NEG:    LXI     DE,UPDT1 ;(3)   1ST CLOCK PHASE
        MOV     A,D     ;(1)    HIGH BYTE
        ORA     C       ;(1)    SAMPLED NEURON
        OUT     PORTH   ;(3)    OUTPUT HIGH
        MOV     A,E     ;(1)    LOW BYTE
        OUT     PORTL   ;(3)    OUTPUT LOW BYTE
                        ;
        LXI     UPDT2   ;(3)    2ND CLOCK PHASE
        MOV     A,D     ;(1)    HIGH BYTE
        ORA     B       ;(1)    UPDATED NEURON
        OUT     PORTH   ;(3)    OUTPUT HIGH
        MVI     A,00H   ;(2)    LOW BYTE
        SUB     M       ;(2)    ABSOLUTE VALUE
                        ;       OF WEIGHT IN A
        ORA     E       ;(1)    LOW BYTE
                        ;       CONTROL WORD
        OUT     PORTL   ;(3)    OUTPUT LOW BYTE
                        ;
        LXI     UPDT3   ;(3)    3RD CLOCK PHASE
        MOV     A,D     ;(1)    HIGH BYTE
        OUT     PORTH   ;(3)    OUTPUT HIGH
        MOV     A,E     ;(1)    LOW BYTE
        OUT     PORTL   ;(3)    OUTPUT LOW BYTE
                        ;
ZERO:   INR     C       ;(1)    INCR. COUNTERS
        INX     HL      ;(1)    INCREMENT
                        ;       MEMORY INDEX
        MOV     A,C     ;(1)
        CPI     N       ;(2)    COMPUTE C-N
        JNZ     LOOP3   ;(3/2)  JUMP IF C<N
                        ;
        LXI     DE,UPDT4 ;(3)   SEND LATCH SIG.
        MOV     A,D     ;(1)    HIGH BYTE
        OUT     PORTH   ;(3)    OUTPUT HIGH
        MOV     A,E     ;(1)    LOW BYTE
        OUT     PORTL   ;(3)    OUTPUT LOW BYTE
                        ;
        INR     B       ;(1)
        MOV     A,B     ;(1)
        CPI     N       ;(2)    COMPUTE B-N
        JNZ     LOOP2   ;(3/2)  JUMP IF B<N
                        ;
                        ;       POSSIBLY
                        ;       PERFORM READ-IN
                        ;       AND CHANGE
                        ;       WEIGHTS
                        ;
                        ;
                        ;       DISCHARGE PHASE
                        ;
        LXI     DE,DSCHG        ;(3)
DISCHARGE
                        ;       CONTROL WORD
        MOV     A,D     ;(1)    HIGH BYTE
```

```
        OUT   PORTH   ;(3)    OUTPUT HIGH
        MOV   A,E     ;(1)    LOW BYTE
        ORI   40H     ;(2)    SET SELECT BIT
        OUT   PORTL   ;(3)    OUTPUT LOW BYTE
        MVI   B,01H   ;(2)    COUNTER
              ;
LOOP4:  MOV   A,D     ;(1)    HIGH BYTE
        XRI   10H     ;(2)    CHANGE CLK BIT
        MOV   D,A     ;(1)
        ORA   B       ;(1)    SELECT NEURON
        OUT   PORTH   ;(3)    OUTPUT HIGH
        MOV   A,E     ;(1)    LOW BYTE
        OUT   PORTL   ;(3)    OUTPUT LOW BYTE
        INR   B       ;(1)
        MOV   A,B     ;(1)
        CPI   N       ;(2)    COMPUTE B-N
        JNZ   LOOP4   ;(3/2)  JUMP IF B<N
        JMP   START   ;(3)
              ;
              ;       TOTAL TIME =
              ;       58 N^2 + 30 N
              ;       + 19
              ;
HLT
END
```

In contrast, a program to perform the same operations in software takes more time. Following is a program for the same machine which uses Booth's algorithm (Short, 1981) to multiply signed numbers.

```
NEUR:   DS    10H     ; INITIAL STATES OF
                      ; NEURONS
WTS:    DS    100H    ; WEIGHT ARRAY
N       EQU   10H     ; NUMBER OF NEURONS
                      ;
                      ;
                      ;
START:  LXI   HL,WTS  ;(3)
        PUSH  HL      ;(3)
        LXI   HL,NEUR ;(3)
        MVI   B,00H   ;(2)    B = COUNTER
              ;
LOOP1:  MVI   E,00H   ;(2)    E = SUM
        MVI   C,00H   ;(2)    C = COUNTER
              ;
LOOP2   MOV   D,M     ;(2)    D = NEUR C
        INX   HL      ;(1)
        XTHL          ;(5)    (HL) = WEIGHT
        MOV   A,M     ;(2)    A = WEIGHT
        INX   HL      ;(1)
        XTHL          ;(5)    (HL) = NEUR
        PUSH  BC      ;(3)
        PUSH  DE      ;(3)
        MOV   C,A     ;(1)    C = WEIGHT
              ;
              ;       MULTIPLY
              ;       C * D
              ;
SMULT:  MVI   B,00H   ;(2)
        MVI   E,08H   ;(2)    # BITS
        XRA   A       ;(1)
        MOV   A,C     ;(1)
LLA:    JC    LLB     ;(2/3)
        RRC           ;(1)
        MOV   A,B     ;(1)
        JNC   LLC     ;(2/3)
        SUB   D       ;(1)
        JMP   LLC     ;(3)
LLB:    RRC           ;(1)
        MOV   A,B     ;(1)
        JC    LLC     ;(2/3)
        ADD   D       ;(1)
LLC:    MOV   B,A     ;(1)
        RAL           ;(1)
        MOV   A,B     ;(1)
        RAR           ;(1)
        MOV   B,A     ;(1)
        MOV   A,C     ;(1)
        RAR           ;(1)
        MOV   C,A     ;(1)
        DCR   E       ;(1)
        JNZ   LLA     ;(2/3)
              ;
              ;       PRODUCT IN B
              ;
        MOV   A,B     ;(1)
        POP   DE      ;(3)
        POP   BC      ;(3)
        ADD   E       ;(1)
        MOV   E,A     ;(1)
        INR   C       ;(1)
        MOV   A,C     ;(1)
        SBI   N       ;(2)
        JNZ   LOOP2   ;(2/3)
              ;
        PUSH  HL      ;(3)    STORE NEURON
        LXI   HL,NEUR ;(3)
        MOV   A,B     ;(1)
        ADD   L       ;(1)
        MOV   L,A     ;(1)
        MVI   A,00H   ;(2)
        ADC   H       ;(1)
        MOV   H,A     ;(1)
        MOV   M,E     ;(2)
        POP   HL      ;(3)
        INR   B       ;(1)
        MOV   A,B     ;(1)
        SBI   N       ;(2)
        JNZ   LOOP1   ;(2/3)
              ;
              ; TOTAL TIME =
              ; 194 N^2 + 28 N + 10
              ;
HLT
END
```

APPENDIX D: SOME OF THE VARIABLES AVAILABLE FOR INTEGRATION IN NEURAL NETWORKS

The following table, from Bullock (1977, p. 234-235), demonstrates the great variation among biological neurons, and how few properties neural networks take into account.

...[W]e emphasized the common denominators of nervous elements — nerve cells, axons, synapses, impulses, and p.s.p.'s [post synaptic potentials] — as a first approximation to understanding the neural machine. It is justifiable simplification to treat neurons as all alike, up to a point.

Here we restate some of the reasons ... that a more sophisticated view of how the nervous system works must emphasize the differences among nervous elements....

1. Spike threshold can be high or low (i.e., depolarization from resting level necessary to fire).
2. Recovery cycle through the relatively refractory period can be slow or fast.
3. Subthreshold local potential can be lower or higher in the steepness of its nonlinear response function.
4. Safety factor [ratio of spike height to threshold depolarization from resting potential] can be high or low, especially at axon branch points.
5. Accommodation [rise in threshold for slow depolarization] can be large or small; fast or slow.
6. Time constant [time required for a passive potential to decay to within $1/e$ of the resting potential] of membrane can be large or small, especially in dendrites and axon terminals.
7. Space constant [distance at which a passive potential decays to within $1/e$ of the resting potential] of membrane can be large or small, especially in dendrites and axon terminals.
8. Afterpotentials [potential occurring after spike, which may be hyperpolarized or depolarized] can be large or small in the magnitude and duration of each successive phase.
9. Iterativeness [repeated firing] can be tonic [long lasting and not adapting] or phasic [transitory, adapting] or both; fast or slow or both in succession.
10. Autorhythmicity can be large or small; the neuron spontaneously active or driven or both.
11. Pacemaker units [neurons firing rhythmically] can be invaded and reset by input or not.
12. Influence by hormones, CO_2, temperature, and other agents can vary in direction and degree.

13. Quantal miniature junction potentials [spontaneous leaks of packet of neurotransmitters across synapses, causing small potentials] can be great or few in number; large or small in amplitude.
14. Synaptic transmission can be chemical [involving neuro-transmitters] or electrical [transmission which spreads ionic currents between neurons].
15. Synaptic transmission can be excitatory or inhibitory.
16. Synaptic transmission can be polarized [transmission in only one direction] or unpolarized.
17. Synaptic transmission can be high gain or low gain.
18. Postsynaptic potential can be monophasic or biphasic.
19. Postsynaptic response can be slow or fast.
20. Postsynaptic decay can be solely passive or partly active.
21. Postsynaptic response can depend mainly on potential change or on conductance change.
22. Postsynaptic inhibition can occur with increased or with decreased conductance.
23. Postsynaptic inhibition can occur with discrete i.p.s.p.'s or i.l.d. (inhibition of long duration, up to many minutes).
24. Postsynaptic activity may or may not exert influence back on presynaptic ending.
25. Postsynaptic response can be facilitating [repeated firing results in higher p.s.p. than from temporal summation] or antifacilitating [repeated firing results in lower p.s.p.] or neither.
26. Postsynaptic response after the end of input can continue or rebound or neither.
27. Facilitation can be fast or slow or both in succession.
28. Input sequence and interval can be critical (e.g. A *then* B is critical = heterosynaptic facilitation) or not.
29. Firing rate/depolarization function can be steep or shallow.
30. Firing rate can be more or less subject to autoinhibition or some form of saturation.
31. Firing rate can be regular or patterned or bursting or irregular.
32. Proportion of signal to noise in neuronal activity can be higher or lower.
33. Electrotonic connection between cells can be low or high in resistance.
34. Electrotonic connection between cells can be low or high in capacitance.
35. Synchrony in subthreshold potentials between units can be strong or weak or at chance level.
36. Synchrony of spike firing can be strong or weak or at chance level.

37. Recognition units can have lower or higher complexity of input requirements.
38. Command units can be of greater or lesser effectiveness (i.e., one or several units command a large musculature); sustained or triggering; high or low in responsibility.
39. Redundancy of neurons can be high or low.
40. Central determinants or sensory determinants of the time course of activity can be relatively more important.
41. Set point of a control circuit can be adjustable or relatively fixed.
42. Potentiation [enhancement of response after long high-frequency stimulation] can be present or absent.
43. Recruiting response [widely distributed evoked potential in the cerebral cortex] can be present or absent.
44. Augmenting response [evoked potential in the primary sensory cortex] can be present or absent.
45. Kindling response [building of after-discharge and possibly convulsions after intermittent stimulation at long intervals] can be present or absent.
46. Evoked slow waves can be large or small, early or late, simple or complex.
47. DC and slowly fluctuating potentials across cells and masses (the EEG and related states) can form fields of larger or smaller equivalent dipoles, simple or complex form, widely different power spectrum, and varying coherence with other areas.
48. Plasticity as a result of environmental influence or experience can be high or low.