Neural Networks: A Conceptual Overview[*]

Abstract

Neural networks have received much attention in both the research community and the media in recent years. Neural networks have been shown to elegantly and powerfully realize solutions to problems in pattern recognition, associative memory, and database retrieval. The Tactical Technology Office of the U.S. Defense Advanced Research Projects Agency (DARPA) concluded a study of neural networks research. They concluded that neural networks hold a lot of unrealized potential, that neural network theory needs to be developed, and that true neural network hardware must be developed.

This overview of neural networks is intended to provide the reader with a general understanding of what a neural network is and what they can be used for. Care is given to provide sufficient details for completeness, while avoiding many of the theoretical complexities. This report contains a brief review of the history of neural networks, followed by a detailed description of a few major neural network architectures. These architectures can be broken down into two main classes: feedforward and feedback. A few neural networks are discussed from each class, and examples of each are introduced. One general, architecture independent learning law is also discussed.

[*]Peter J. W. Melsa, Tellabs Research Center, Mishawaka, IN 46545, TRC-89-08

1. Introduction

The term neural network is used to describe various topologies of highly interconnected simple processing elements that offer an alternative to traditional approaches to computing. The topic of neural networks has received a lot of attention during the past ten years, and will likely receive more attention in the future. Much work remains before neural networks will become widely used and accepted as computing structures. The Tactical Technology Office of the U.S. Defense Advanced Research Projects Agency (DARPA) recently completed a study of neural networks. DARPA was interested in determining the current status and potential usefulness of neural networks, and assessing the worth of investing more time, energy and money. The DARPA study reached the following conclusions.

- Neural networks offer important new approaches to information processing because of their adaptability and ability to learn as well as their massive parallelism.[7, page 30]
- Neural network research has matured greatly since the perceptron of the 1950's. This maturation has three resources: (a) advancement in mathematical theories, (b) development of new computer tools, and (c) increased understanding of neurobiology. There is a bridge forming between the biologically — oriented neural modelers and the artificial neural network modelers, and substantial progress will result from this alliance. [7, page 30]
- Although the computer revolution of the last 20 years has played a critical role in the revival of neural network research by making it possible to undertake much work in mathematical theory, the limits of today's computing devices — inadequacies in storage, speed, and user flexibility — have restricted present neural network efforts. [7, page 35]
- There have been significant demonstrations of neural network capabilities in vision, speech, signal processing, and robotics, perhaps not to the scale which might be desired — due to a lack of funding of research — but the variety of problems addressed by neural networks is impressive. [7, page 51]
- Hardware capabilities are limiting the development of important neural network applications. It is clear that if researchers are provided with improved simulation and implementation capabilities, the field of neural networks will once again drift off into the wilderness. [7, page 52]

The goal of this report is to provide a structural, functional and conceptual overview of the basic and popular neural network architectures. The report will not delve too deeply into the theoretical considerations of any one network, but will concentrate on the mechanics of their operation. Suitable credibility will be supplied to demonstrate that the architectures are viable and functional. Future reports will more carefully consider the theory behind selected individual networks. Numerous references are given at the end of this report for the interested reader.

It is worth noting that the topic of neural networks is given various names by different authors. Some of the names given to this topic are: artificial neural systems, connectionist models, parallel distributed processing models, layered self–adaptive systems, self–organizing systems, neurocomputers, neuromorphic systems, and cyberware. There are various subtle differences between the definitions the authors use, but for this discussion they are all roughly synonymous. This report will simply refer to them as neural networks.

2. Basics of Neural Networks

2.1 History and Biophysiology

Neural networks are not a new concept. The idea of a neural network was originally conceived as an attempt to model the biophysiology of the brain, i.e. to understand and explain how the brain operates and functions. The goal was to create a model capable of human thought processes. Much of the early work in neural networks was done by biophysicists and scientific psychologists, not engineers.

> The outlines of network systems were discussed in the nineteenth century. What is different today is the ability, thanks to the computer and general familiarity with mathematical analysis, to take such theoretical networks and actually construct them and use them. Our command of the detail, techniques, and application of modeling is now very much greater. However, the basic insights of William James into the operation of the mind have not been fundamentally altered. [1, page3]

James in 1890 made the following statement:

> The amount of activity at any given point in the brain cortex is the sum of the tendencies of all other points to discharge into it, such tendencies being proportionate (1) to the number of times the excitement other points may have accompanied that of the point in question; (2) to the intensities of such excitements; and (3) to the absence of any rival point functionally disconnected with the first point, into which the discharges might be diverted. [21, page 257]

If "point in the brain cortex'' is replaced with neuron, neural activity is modeled as the sum of the neuron's weighted inputs. The neuron's activity will depend upon: the connection strengths as determined by the past correlations of the neuron's activity with the activity of the other neurons from which it receives inputs (statement 1), the current excitement of the other neurons providing inputs (statement 2), and an inhibitatory term (statement 3). In 1943, McCulloch and Pitts [33] formalized a description of these neurons. The McCulloch–Pitts neuron will be discussed in Section 2.3. They adopted the statements that James and other biologists had made, and developed a model for the nervous activity in the brain. The McCulloch–Pitts neuron is still the major building block of virtually every neural network being developed. Current neural networks combine these basic neurons in different architectures to achieve a variety of computational capabilities. Research has largely centered on the architectures in which the neurons are combined, and the methodologies by which the interconnection strengths are determined and adjusted.

Today there are two different, not always disjoint, groups of people exploring neural networks. The first group is composed of biologists, physicists and scientific psychologists who work toward developing a neural model that accurately mimics the behavior of the brain. The second group consists of engineers who are concerned with how ''artificial'' neurons can be interconnected to form networks with interesting and powerful computational capabilities. This second group treats the biological models as a functional starting point for research, and this is the approach emphasized in this report.

"The brain is not constructed to think abstractly — it is constructed to ensure survival in the world. It has many of the characteristics of a good engineering solution applied to mental operation: do as good a job as you can, cheaply, and with what you can obtain easily." [1, page 1] The goal of neural network research is not to generate machines which crunch numbers more quickly than traditional computers, but rather to create machines that excel in areas where humans easily outperform traditional computers. Neural networks are meant to supplement not replace traditional computers. In fact, some people invision neural network devices that can operate as pseudo co–processors for traditional computers. For example, traditional computers have great difficulties recognizing objects given partial information, or quickly seeing a good, not always optimal, solution to a problem. Because neural networks are capable of generating efficient and fast solutions to the same problems the brain solves quickly, the separation between engineers and scientists on the approach to neural networks often narrows. Scientist say, "Lets make it work like the brain" Engineers say, "Lets make it do problems that humans do better than machines." The difference can be viewed as a matter of syntax, not content.

In 1969, engineering research into neural networks effectively came to a halt with the publication of the book *Perceptrons*, by Minsky and Papert. It is often incorrectly stated that this death was due entirely to Minsky and Papert; a more accurate statement is that they simply "nailed the lid on the coffin." During the 1960's there were a number of events which served to dampen the enthusiasm for neural networks. First, there was a bad feeling about building an artificial "chunk of brain." Second, there was a lot of hype surrounding neural networks. Many scientists were irritated by the extravagant claims made by other researchers in the media. Third, journalists often misrepresented neural networks to the readers. A 1960's Oklahoma newspaper featured the headline "Frankenstein Monster Designed by Navy — Robot That Thinks." [1, page 157] Minsky and Papert showed that the single–layer perceptron network, a model that will be introduced in Section 2.3, is capable of any mapping that is linearly separable. They pointed out that the logical XOR function is not linearly separable and could not be realized by perceptrons. At that time it was important that computational devices be capable of being used to construct computers, a task that single–layer perceptrons would not be able to fulfill. *Perceptrons* has been touted as a brilliant and insightful analysis of the computational capacities of single–layer perceptron networks. Unfortunately, Minsky and Papert stated in their conclusion their belief that the limitations of single–layer perceptron networks would hold true in multilayer perceptron networks. In fact, Minsky and Papert state, "we consider it to be an important research problem to elucidate ... our intuitive judgement is that the extension [to multilayer systems] is sterile." [35, page 232] Neural network research had died. Fortunately, Minsky and Papert's judgement has been disproved; all nonlinearly separable problems can be solved by multilayer perceptron networks given certain conditions, see [3,6,13,20]. If Minsky and Papert had not been so convincing in their condemnation of perceptrons, the neural network field might be more advanced than it is today.

Neural networks became a hot research topic again during the mid to late 1970's. The development of more powerful computer simulation capabilities and VLSI technology have made possible many of the neural network results achieved today. Recently, many neural network architectures have been proposed and simulation results have demonstrated their potential. In addition to further advances in VLSI technology, the tools and knowledge to study neural networks from a theoretical standpoint remain to be developed. Researchers have studied small networks theoretically, but the extensions to reasonable sized networks have been nearly impossible. Networks on the order of thousands of neurons and millions of interconnects are necessary to achieve interesting and sophisticated mappings and transformations.

2.2 Neural Networks vs. Traditional Computers

Robert Hecht–Nielson, of The Hecht–Nielson Neurocomputer Corporation, defines a neural network as "a dynamical system with the topology of a directed flow graph that can carry out information processing by means of its state response to continuous or episodic inputs." [14, page 168] A neural network is fundamentally different than a traditional computer. Neural networks

- do not sequentially execute instructions nor do they contain memory for storing operation instructions or data
- respond in parallel to a set of inputs
- are more concerned with transformations than algorithms and procedures
- do not contain only one or a few complicated computational devices, but are comprised of a large number of simple devices often doing little more than computing weighted sums.

Information in neural networks is not stored in explicit memory locations, but is represented by the interconnection strengths and the current output of the neurons. Neural networks are not replacements for traditional computers, but are an entirely different class of computational devices capable of qualitatively different tasks. Neural networks provide a completely new and unique way to look at information processing.

> The transformations of neurocomputing cannot be understood using the equipment and methodologies of algorithms and procedural thought any more than the behavior of atoms can be understood in the framework of Newtonian physics. A new set of conceptual tools that allow us to characterize neurocomputing transformations in non–algorithmic and non–procedural terms and understand their development during training will be required. In the end, a radically new, broader, view of information processing will emerge. [14, page 167]

To fully appreciate and understand neural networks they must not be cast into the traditional view of computing machines. They are vastly different, and must be viewed as such.

Also, while many neural networks are currently being simulated on computers, neural networks should not be considered a new programming technique. Neural network applications are often explored via computer simulations, but these simulations are only tools for numerical analysis of networks. Given the complexity of neural network architectures, computer simulations are inefficient ways to implement neural networks. Neural network hardware will have to be developed before neural networks are efficiently implemented.

2.3 Basic Elements and Terminology

Figure 1 shows the biological neuron model upon which neural networks are based. Each neuron in the brain is composed of a *body*, one *axon*, and a multitude of *dendrites*. The dendrites form a very fine "filamentary brush" surrounding the body of the neuron. The axon can be thought of a long thin tube which splits into branches terminating in little *endbulbs* almost touching the dendrites of other cells. The small gap between an endbulb and a dendrite is called a *synapse*. The axon of a single neuron forms synaptic connections with many other neurons. Figure 1 shows the body and axon of one neuron interacting with the dendrites of a second neuron. Impulses propagate down the axon of a neuron and impinge upon the synapses, sending signals of various strengths down the dendrites of other neurons. The strength of a given signal is determined by the *efficiency* of the synaptic transmission. A particular neuron will send an impulse down its axon if sufficient signals from other neurons impinge upon its dendrites in a short period of time, called the *period of latent summation*. The signal impinging upon a dendrite may be either *inhibitatory* or *excitatory*. A neuron will fire, i.e. send an impulse down its axon, if its excitation exceeds its inhibition by a critical amount, the *threshold* of the neuron.

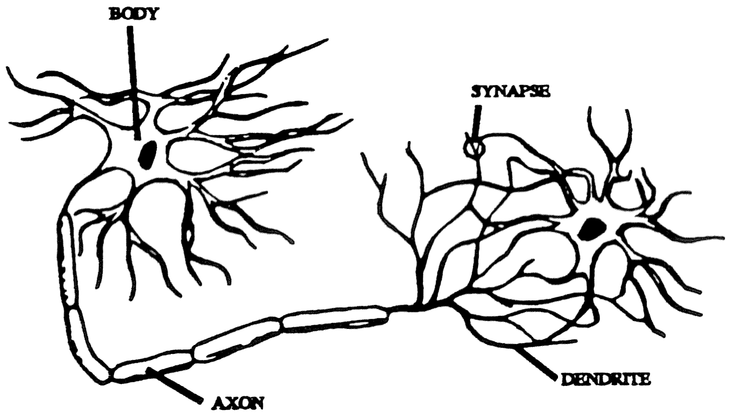


Figure 1: A Biological Neuron

Figure 2 shows an "artificial" neuron, referred to hereafter simply as a neuron, which models the behavior of the biological neuron. The body of the jth neuron will often be represented by a weighted linear sum, z_j, of the input signals followed by a linear or a nonlinear function, $y_j = f(z_j)$. The function $f(z_j)$ is called an activation function, a function which uses the input value to determine the output activity of the neuron. This neuron description is termed a McCulloch–Pitts neuron. One typical case uses a binary threshold function

$$y_j = f(z_j) = \begin{cases} \text{ON} & \text{if } z_j > \text{threshold} \\ \text{OFF} & \text{if } z_j < \text{threshold} \end{cases} . \qquad (1)$$

A McCulloch–Pitts type neuron using this binary threshold is referred to in the literature as a perceptron. The perceptron is the basic element in many neural network architectures. Different neural networks use different $f(z_j)$ functions, but the internal structure of the neuron, i.e. linear sum followed by a function $f(z_j)$, is common to most networks. Examples of functions, $f()$, are linear functions, non–linear sigmoidal functions, and threshold functions. The synaptic efficiencies will be represented by the *interconnection weights*, w_{ij}, from the ith neuron to the jth neuron. Resistors are often used for the interconnection weights in neural network hardware. The weights can be either positive (excitatory) or negative (inhibitatory). Later, when the Hopfield model is discussed, one solution for the need for negative resistive values will be shown. The weights and the functions, $f()$, dictate the operation of the network. Usually for a given network architecture the functions, $f()$, will be fixed, so varying the weights will allow the network to carry out different computations. The outputs of the neurons determine the current state of the network.

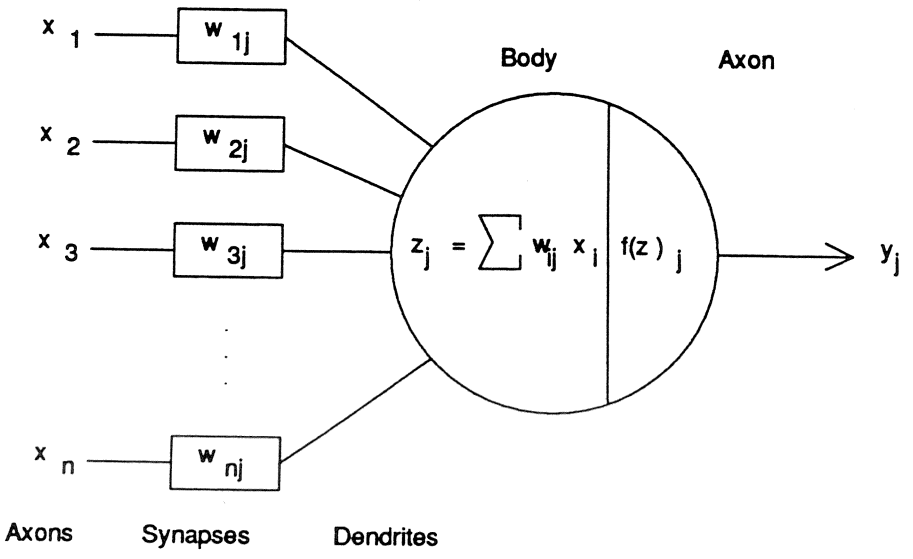


Figure 2: The McCulloch–Pitts Neuron

Neural networks will either have fixed weights or adaptable weights. Networks with adaptable weights use learning laws to adjust the values of the interconnection strengths. If the neural network is to use fixed weights, then the task to be accomplished must be well defined *a priori*. The weights will be determined explicitly from the description of the problem. Adaptable weights are essential if it is not known *a priori* what the correct weights should be. There are two types of learning: supervised and unsupervised. *Supervised learning* occurs when the network is supplied with both the input values and the correct output values, and the network adjusts its weights based upon the error of the computed output. *Unsupervised learning* occurs when the network is only provided with the input values, and the network adjusts the interconnection strengths based solely on the input values and the current network output. *Learning laws* determine how the network is to adjust its weights using an error function or some other criteria. The learning law used is determined by the desired characteristics of the network.

Adaptable neural networks may have two phases to their operation. The first phase is the *training* of the network. The user provides the network with a "suitable" number of input patterns, and output patterns if required, and the network adjusts the interconnection strengths until the output is "close enough" to the correct output. There are some applications, using either supervised or unsupervised learning, where there will be no explicit training phase. The second phase is the *recall* or computation phase. The network is presented an input pattern and the network simply computes its output. When a network uses unsupervised learning, it will sometimes continue to adjust the interconnection strengths during the recall phase. Usually, supervised networks will not do any learning during recall since that would require that the user know *a priori* the correct answer.

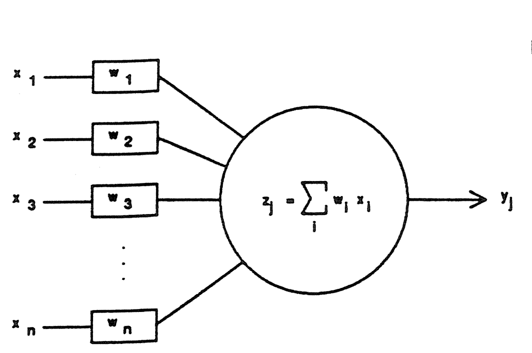
Neural networks operate in either discrete time or continuous time depending upon the network architecture used. Some networks use neurons which compute at synchronous and discrete intervals, others use asynchronous computations in continuous time. Typically, feed–forward networks use synchronous and discrete computations, while feed–back networks use asynchronous and continuous computations.

3. Feed–Forward Neural Networks

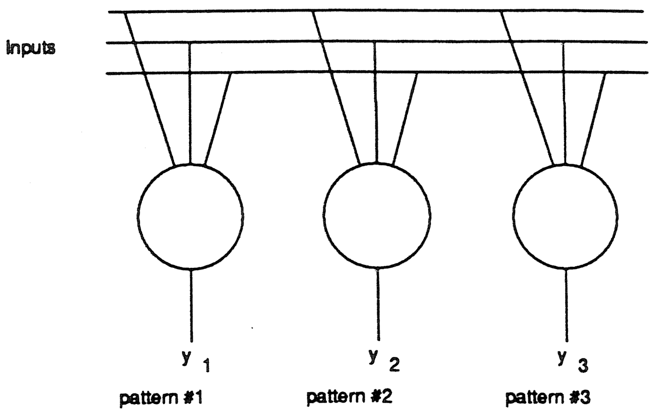
Neural network architectures can be classified in two groups based upon their operation during the recall phase. In a feedforward network the output of any given neuron can not be fed back to itself directly or indirectly or through other neurons, and so its present output does not influence future outputs. In these networks computations are completed in a single pass. When an input pattern is presented to the input terminals of a network the neurons in the first layer compute output values and pass these values onto the next layer. Each layer in turn receives input values from the previous layer, computes output values and passes these values onto the next layer. When the output values of the final layer are determined the computation ends. This rule can be broken only during the training phase or the learning portion of the recall phase, when the output of a neuron can be used to adjust its weights, thus influencing future outputs of the neuron. An example of this type of structure is the ADALINE discussed in section 3.2. Another popular feedforward network, backpropagation, uses feedback between layers during training. Remember, that the operations during the computational portion of the recall phase are what determine the classification of the network. In a feedback network any given neuron is allowed to influence its own output directly through self feedback or indirectly through the influence it has on other neurons from which it receives inputs. Feedback networks are discussed in Section 4.

3.1 Grandmother Cells

One of the simplest neural networks possible is a Grandmother cell network, see Figure 3(a). Grandmother cell networks are comprised of neurons with no interconnections. The input weights are fixed, and each neuron simply computes a linear sum of the weighted external inputs. Each neuron receives the same input but has different weights. If the input pattern and the weights are viewed as vectors, then the grandmother cell computes as its output the dot product of the input vector with the weight vector. Recalling that the dot product computes the projection of one vector onto another, the output of the cell will indicate how ''closely'' the input vector matches the weight vector. If the input vectors and weight vectors are normalized to a length of one, then the output lies between +1 and -1, +1 corresponding to an exact match and 0 indicating that the vectors are orthogonal. The weights for the grandmother cell are chosen based upon the pattern that the cell should respond to. If grandmother cells are combined as in Figure 3(b), then the network can be used to sort input patterns into various classes. An input pattern is presented to the network, each cell computes its output, and the cell with the largest output indicates which stored pattern the input pattern most closely resembles. However simple the grandmother cell architecture seems, it is actually quite powerful. Each input pattern is simultaneously compared with numerous test patterns. This concept is crucial to the use of neural networks for problems such as pattern recognition. As will be shown, other feedforward neural networks operate like the grandmother cell during recall.



(a) Individual Cells



(b) Bin Sorting Example

Figure 3: Grandmother Cells

3.2 ADALINE

One of the most popular neural network learning laws is the *Delta Rule* or normalized *Least Mean Square (LMS)* training law proposed by Widrow and Hoff [49, 50, 51]. Widrow and Hoff used an ADALINE to associate an input pattern with an output value. An ADALINE, which stands for ADAptive LINear Element, consists of a single neuron of the McCulloch–Pitts type with $f(z_j) = z_j$ and with weights determined by the normalized LMS learning law. Supervised learning is used, so, during training, the neuron knows the input and the correct output. During training, the neuron computes an error signal, the difference between the desired output and the computed output, and adjusts the weights so that the computed output is closer to the desired output. During the recall phase an ADALINE functions in the same fashion as a grandmother cell. Since an ADALINE is a linear device, hierarchical layers of ADALINEs will not increase computational capability. Any linear combination of linear units can be accomplished with the use of a single linear unit.

The normalized LMS algorithm attempts to minimize the mean squared error signal. Given some statistics for the input signal, there is a mean squared error value associated with every weight vector. The set of these values form a performance surface in *weight space*. It is not known where the minima of the surface lie, but it is possible to estimate the gradient of the surface, and thus to use a gradient descent method to search for a minimum. In a gradient descent method updates are made in the direction of the locally steepest descent. Usually, this means computing the derivative of the error surface at the current location. However, Widrow and Hoff showed that an estimate of this derivative is proportional to the error, and the normalized LMS update is based on this gradient estimate. The ADALINE will always adapt to produce the minimum mean squared error, although when operating in the presence of noise this minimum error value is greater than zero.

Figure 4 shows the structure of a single ADALINE. An input pattern is presented to the neuron, the neuron computes a weighted sum of the inputs and then computes the error signal. The error signal is used to adjust the weights using the Delta rule. Representing the weights and the inputs by vectors, with n elements in each, the Delta rule for a nonzero input vector can be written as

$$W_{new} - W_{old} = \frac{\beta E X}{\| X \|^2} \tag{2}$$

where **W** is the weight vector, **X** is the input vector, β is a learning gain, $E = d - y$ is the error, d is the desired output, and y is the actual output. When the input vector is zero no update is computed. See Appendix A for a proof showing that the delta rule is an estimated gradient descent method.

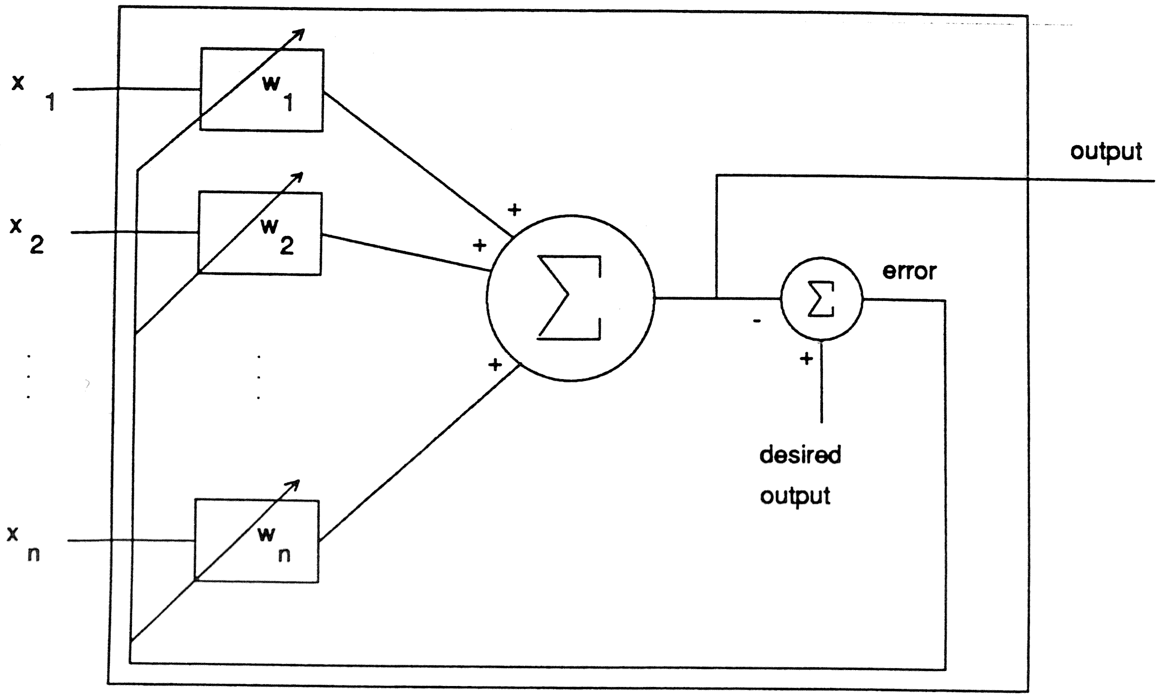


Figure 4: Adaline

The choice of the constant β will influence the size of the steps the weight vector takes while adapting, and thus the rate at which the element learns. This choice is crucial to the correct operation of the training procedure. The first criterion on β is that it be positive; a negative β would force the weights to change away from the minimum. Also, β must be less than two for the update to be stable. Most authors suggest that a typical range of values is $0.1 < \beta < 1$. The choice for the value of β will ultimately be dictated by the particular problem, how small the mean squared error should be in the presence of noise, and how long the convergence is allowed to take. For further information on the choice of β, the reader is referred to other sources, e.g. [52].

In practice, an ADALINE is usually used to make binary decisions, so the output is sent through a binary threshold function, see Figure 5. This new device is an example of a perceptron, see section 2.3, and is only capable of performing binary transformations that are linearly separable. A problem is linearly separable if the input patterns plotted in input space can be grouped as desired by bisecting the input space with a single hyperplane, e.g. a single straight line in two dimensional space. The classic example of a mapping that is not linearly separable is the XOR function. Assuming that each input is either +1 or -1 Figure 6 shows the locations of the four input patterns in input space and a symbol corresponding to the output of the XOR function for each. There is no way to draw a single straight line so that the circles are on one side of the line and the squares are on the other, i.e. the groups can not be linearly separated.

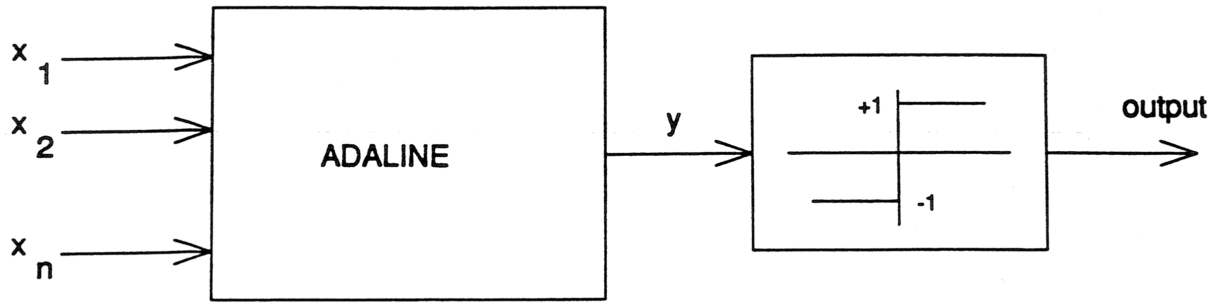


Figure 5: Perceptron

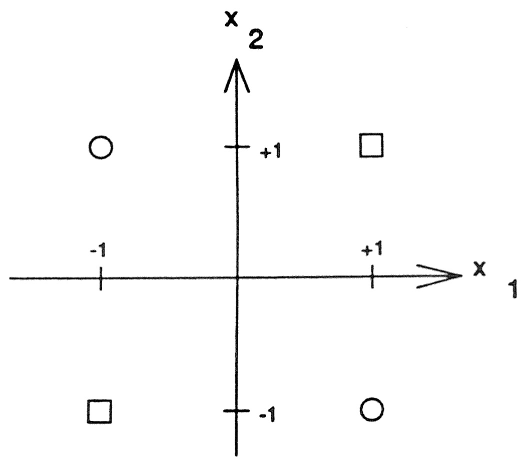


Figure 6: The XOR Function

Consider a perceptron with two inputs, x_1 and x_2, and assume that the threshold condition for which the output first switches from -1 to +1 is

$$y = x_1 w_1 + x_2 w_2 = 0$$

$$\Rightarrow x_2 = -\frac{w_1}{w_2} x_1 . \tag{3}$$

This linear equation defines the transformation realized by the perceptron. The weights, w_1 and w_2, determine the slope of the line, so adjusting the weights is equivalent to adjusting the transformation. The perceptron can be generalized with the addition of a weight, w_0, connected to a bias always having a value of +1. The threshold condition for the perceptron then becomes

$$y = w_0 + x_1 w_1 + x_2 w_2 = 0$$

$$\Rightarrow x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}. \qquad (4)$$

Now the separating line is no longer required to pass through the origin, allowing more flexible transformations. Figure 7 shows one possible use of a perceptron with a bias to implement the logical OR function.

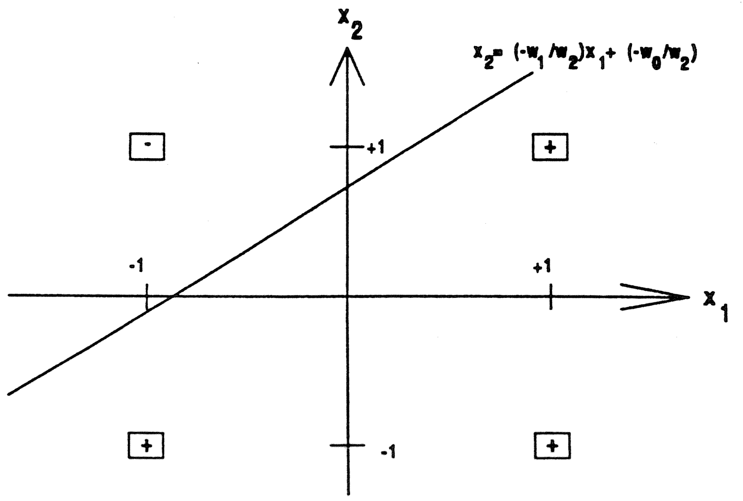
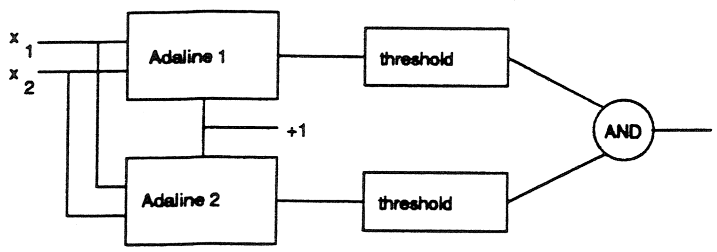
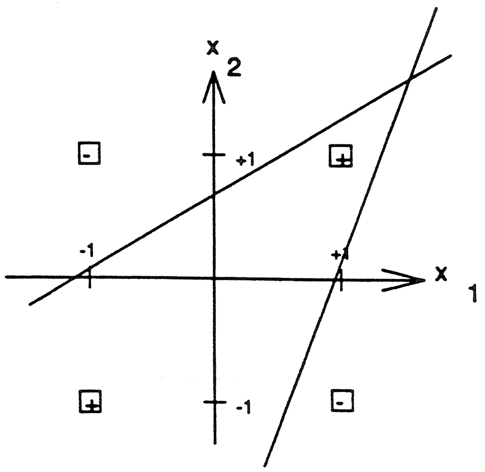


Figure 7: Logical OR with a Perceptron

The next logical extension is to place perceptrons in a hierarchical structure. A layer of perceptrons may be connected to fixed logic gates to form solutions to nonlinearly separable problems. This type of network is termed MADALINE (Many ADALINEs). In a MADALINE only the weights of the ADALINEs are adjusted. There are methods available to adjust the weights of the ADALINEs to solve specific problems using a MADALINE network, but these will not be discussed in the report, see [37, 41, 49, 53]. Figure 8(a) shows a network with two perceptrons connected to a logical AND gate. When suitable weights are chosen this network is capable of implementing the XOR function, see Figure 8(b). MADALINEs can be built with many more inputs and perceptrons, and with fixed logical gates such as OR, AND, and Majority, etc. to produce more complicated mappings.



(a) MADALINE



(b) The XOR Function

Figure 8:

3.3 Backpropagation

The backpropagation network is an extension of the MADALINE structure involving the addition of a nonlinear $f()$ on the output of each neuron, and the use of multiple adaptable layers. No longer are only the weights in one layer adaptable, but the weights in all layers are adaptable. It can be shown that the backpropagation network is capable of approximating as closely as desired any mapping between any vector of dimension m and any other vector of dimension n, see [3, 6, 13, 20]. Some other discussions of the mapping representability can be found in [19, 37, 38, 54].

The neurons used in the backpropagation network will be McCulloch–Pitts neurons using a sigmoidal function for $f(z_j)$, i.e. a function which is continuous, "S–shaped", monotonically increasing, continuously differentiable, and asymptotically approaches fixed finite values as the input approaches plus or minus infinity. The fixed asymptotic values are generally +1 and zero, or +1 and -1. One example of a sigmoidal function is

$$f(x) = \frac{1}{1 + e^{-(x+T)}} \tag{5}$$

where T is a threshold and x is the sum of the weighted inputs for the neuron. Since the activation can never attain its asymptotic value, in practice if the sum is above some prescribed value the activation is usually assigned the value of the asymptote. Figure 9 shows a graph of the above sigmoidal function for $T = 0$. In general the basic properties of the sigmoidal function outlined above, i.e. continuous, monotonically increasing, etc., are more inportant than the specific sigmoidal implementation chosen.

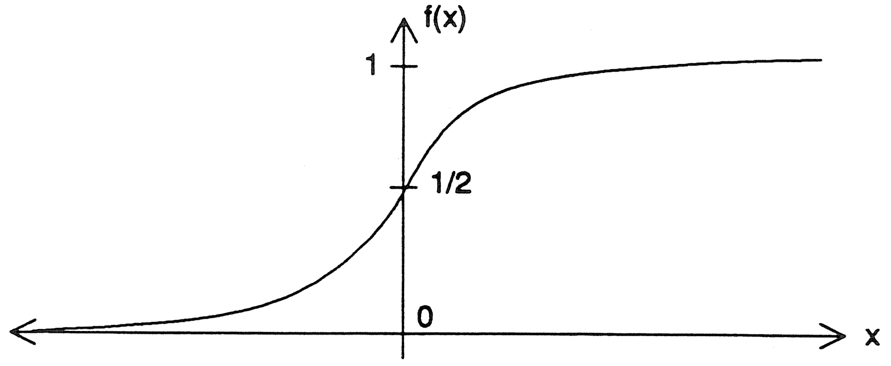


Figure 9: A Sample Sigmoidal Function

Technically, backpropagation is a specific learning law. The term is often used however to refer to a hierarchical network architecture that uses the backpropagation update algorithm for updating the weights of each layer based on the error present at the network output. A backpropagation network always consists of at least three layers; an input layer, a hidden layer, and an output layer. For some applications more than one hidden layer is used, but this report discusses a network with only one hidden layer. The number of neurons in the input layer is equal to the number of inputs in each input pattern, and each of these neurons receives one of the inputs. The output of the neurons in the output layer are the outputs of the network. The number of neurons in the hidden layer is up to the discretion of the network designer. Too many neurons will tend to make the network operate similiar to a grandmother cell network, resulting in a network that has difficulties dealing with new types of input patterns, e.g. making generalizations. Too few neurons will not allow the network to make sufficient internal representations so that accurate maps from the input patterns to the output patterns can be made. Some researchers have proposed some general guidelines for chosing the number of hidden neurons, but no one has given strict rules for how many neurons to use, see [28, 44].

In a backpropagation network there are no interconnections between neurons in the same layer. However, each neuron on a layer provides an input to each and every neuron on the next layer, e.g each input neuron will send its output to every neuron in the hidden layer. Figure 10 shows a sample network with four input neurons, two hidden neurons and three output neurons. The input layer neurons will each have a single input and a single output, and will simply pass the value on their input to their output, without using the sigmoidal function. It is a valid remark that the input layer serves no functional purpose and could be omitted. Traditionally the input layer has been included in networks discussed in the literature. Generally, the weights between the layers are initially small random values. This allows the network to train and function well. It is important that the weights not start with the same value, so that internal representations requiring nonsymmetric weights can be obtained.

There are two distinct operations which take place during the training phase: the feedforward computation and the update of the weights based upon the error of the network output. The backpropagation network uses supervised learning so the input/output patterns to be associated

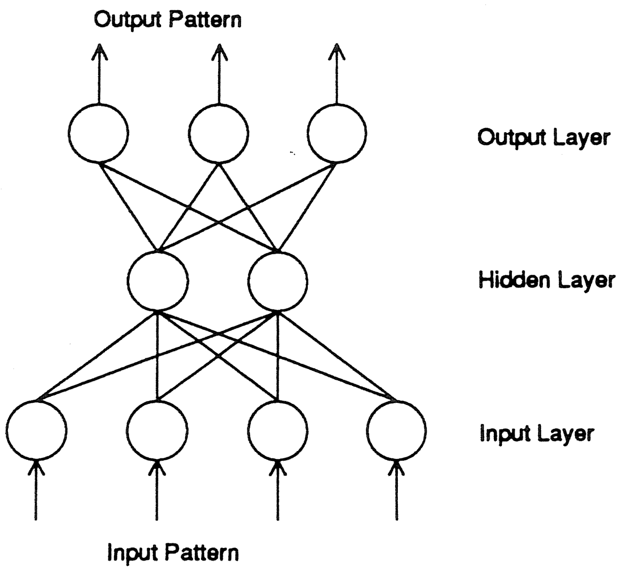


Figure 10: A Sample Backpropagation Network

are both known. During recall only the feedforward computations, consisting of a singlepass of calculations, take place.

The feedforward process involves presenting an input pattern to the input layer neurons which pass the values onto the hidden layer. Each of the hidden layer neurons computes a weighted sum of its inputs, passes the sum through its activation function and presents the activation value to the output layer. Each neuron in the output layer also computes a weighted sum of its inputs and passes the sum through its activation function, resulting in one of the output values for the network. Typically each neuron in the network will use the same activation function and threshold value, although this is not necessary. Some researchers are experimenting with using different threshold and activation functions for each of the different layers to get improved capabilities from the backpropagation network.

The following backpropagation update algorithm insures that the network follows an estimated gradient descent on a quadratic function of the output error with respect to the weights, for a proof of this fact see Appendix B. Once the output values have been computed, they are combined with the desired outputs to generate error values with which the weights of the network can be adjusted. The error for the jth output layer neuron is

$$E_j = (d_j - y_j) f_j'(net_j) \tag{6}$$

where d_j is the desired output, y_j is the computed output, $f_j'()$ is the derivative of the activation function, and netj is the net input to the neuron. The net input is the sum of the weighted inputs. The update equation for the jth output layer neuron is

$$W_j^{new} - W_j^{old} = \frac{\beta E_j X}{\| X \|^2} \tag{7}$$

where $\mathbf{W}_j$ is the weight vector for the jth output layer neuron, $\mathbf{X}$ is the input vector from the hidden layer to the output layer, and β is the learning gain. For the ith hidden layer neuron the update equation is

$$W_j^{new} - W_j^{old} = \frac{\beta\, [f_i\,'(net_i)\sum_j (w_{ij}\, E_j)\,]Y}{\| Y \|^2} \tag{8}$$

where $\mathbf{W}_i$ is the weight vector for the ith hidden layer neuron, $\mathbf{Y}$ is the input vector from the input layer to the hidden layer, w_{ij} is the weight from the ith hidden layer neuron to the jth output layer neuron, E_j is the error of the output of the jth output layer neuron, $f_j'(\,)$ is defined above, and the sum is over the neurons in the output layer. For the common sigmoidal function given in equation (5)

$$f\,'(x) = f(x)\,[\,1 - f(x)\,]\,, \tag{9}$$

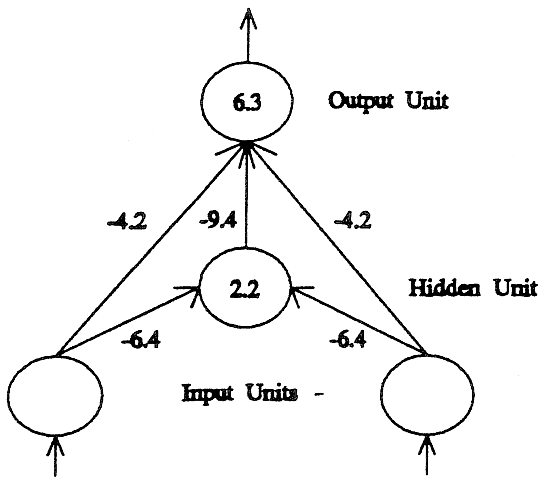
so the calculation of the derivative is easy. For the case where the outputs lie between zero and one, the derivative will have its greatest value for $x = 1/2$, e.g. when a given neuron is not yet committed to being either on or off. The above equations are also referred to as the generalized Delta rule.

Some authors indicate that the backpropagation network can converge to local rather than global minima, see [43]. In those situations it is suggested by some that one add a momentum term to avoid local minima and thus improve the performance of the Generalized Delta rule. The Generalized Delta rule for the output layer, for nonzero input, with the momentum term added is

$$W_j^{new} - W_j^{old} = \frac{\beta\, E_j X}{\| X \|^2} + \alpha\,(W_j^{new} - W_j^{old})_{prev}\,. \tag{10}$$

The second term on the right hand side of the equation is the momentum term. This term adds a portion of the most recent weight change when computing the new weight change. The constant α determines how heavily the past weight change affects the current update. An analogy for understanding momentum is a ball rolling down a hill. When a ball rolls down a hill it has momentum, and when it encounters a slight uphill segment continues to roll if the momentum is stronger than gravity, thus escaping the dip. The momentum term is supposed to give the neuron momentum in weight space, enabling it to pass through local minima. The motivation for momentum is based upon intuition rather than a firm mathematical or theoretical foundation.

As a demonstration of the backpropagation network consider the XOR function, a function that was shown to be impossible to implement with the ADALINE. Figure 11 shows one possible implementation using a backpropagation network with two input neurons, one hidden neuron, and one output neuron. This network has a slightly different architecture than has been discussed so far. Direct connections from the input layer to the output layer have been used. The above equations are still valid for this architecture. All one needs to do to verify this fact is to stick a dummy hidden layer neurons in for those links. A nonzero threshold, T, has been used. The numbers inside the neurons indicate the value of the weight connecting the bias unit with the



Input 1	Input 2	Hidden Input	Output Input
0.0	0.0	2.2	-2.9
1.0	0.0	-4.2	2.1
0.0	1.0	-4.2	2.1
1.0	1.0	-10.6	-2.1

Figure 11: Logical XOR with Backpropagation

neuron, or identically the threshold. Both the hidden and the output neurons are effectively either on or off, i.e. they either have an output of nearly one or zero. This does not imply that the neurons use threshold functions. The hidden neuron performs a logical NOR function, i.e. it is on only if both input neurons are off. The output neuron is off if either both inputs are on or the hidden neuron is on (i.e. both inputs are off). Thus, as the truth table in Figure 11 indicates, the network performs the XOR function. Higher numbers of hidden layer neurons could be used. It has been shown that the time to learn the XOR function declines linearly with the logarithm of the number of hidden units, see [43]. These examples demonstrate the capabilities of the backpropagation network to form internal representations that allow the network to solve nonlinearly separable problems. For the above XOR function, the network used the hidden layer neuron to indicate whether one or two of the inputs was on. The output layer then used that information in combination with the inputs themselves to realize the XOR function. The hidden layer provides an internal representation of the inputs. If a network is not able to form internal representations then it must rely on the external world to associate similiar input patterns with similiar output patterns. The XOR function has been used to demonstrate the ability of the backpropagation network to realize nonlinearly separable problems for simplifying and historical reasons. The backpropagation network has been used to solve much more complicated problems. Other demonstrations of the backpropagation algorithm include converting bit–map letters to ASCII characters, parity computations, binary encoding, determining whether a pattern is symmetric are not, addition, playing tic–tac–toe, pattern recognition [43, pp. 330-352], and noise filtering [22]. Backpropation has great generalizing capabilities and is very powerful, however it takes a long time to train a backpropagation network. Some problems require enormous amounts of training data before the network can learn sufficiently.

4 Feed–Back Neural Networks

4.1 Grossberg Learning

The Grossberg learning law is an attempt to explain a biological phenomenon. The Grossberg learning law is not dependent upon a specific network architecture, but is generally applicable. Besides being interesting from a biological standpoint, Grossberg learning has been shown to have interesting and useful computational and organizational properties, see [25, 32]. In this report the learning law itself will be developed, but computational implications will not be discussed. Hebbian synapses, synapses which employ Grossberg learning principles, are discussed frequently in the literature so knowledge of the principles behind them is important. As one example of the use of Grossberg learning, a simple network using Grossberg learning will be discussed later in this section.

The Grossberg learning law is an attempt to mathematically formulate Hebb's Law, which in turn attempts to explain the classical conditioning behaviors discovered by Pavlov. Pavlov's famous experiment involves the conditioning of dogs to salivate when a bell is rung. The dogs would salivate when shown a plate of food, the food being the unconditioned stimulus and the salivation being an unconditioned response. If the experimenter rings a bell at the same time that the food is presented, eventually the dogs begin to salivate when the bell is rung, even if the plate of food is not presented. The bell becomes a conditioned stimulus and the salivation, the conditioned response. Donald Hebb, in 1949, attempted to explain this phenomenon from a neurological standpoint.

Hebb's Law : If a neuron, A, is repeatedly stimulated by another neuron, B, at times when neuron A is active (for whatever reason), then neuron A will become more sensitive to stimuli from neuron B; the synaptic connection from B to A will be more efficient. Thus, B will find it easier to stimulate A to produce an output in the future.

We can relate this law directly back to Pavlov's experiment by considering the analogy that a neuron, B, will be active whenever the bell is rung, and that a neuron, A , will be active whenever the dog salivates in response to seeing the plate of food. Over time the connection from B to A strengthens so that eventually activating B will be sufficient to activate A in the absence of the plate of food. Unfortunately, Hebb's Law is not a mathematical statement that directly applies to our work with neural networks. It was Grossberg who came up with a mathematically plausible set of equations to explain Hebb's Law. The following equations are only one possible set of equations describing Hebb's Law, others could be proposed. Grossberg's learning law equations are not proposed as "the" correct or optimal equations, but rather as a plausible description of a biological phenomenon.

The neurons proposed by Grossberg use a different set of equations to describe their output than have been presented thus far, i.e. the neurons are not simply McCulloch–Pitts neurons. Consider a neuron which has a number of inputs coming from other neurons in the network as well as an external input coming from outside the network. The following equation describes the dynamics of the jth neuron

$$\frac{\partial y_j(t)}{\partial t} = -A\,y_j(t) + I_j(t) + \sum_{i=1}^{n} w_{ij}\,y_i(t)\,, \tag{11}$$

where y_j (t) is the output of the neuron, I_j (t) is an external input to the neuron, w_{ij} is the weight connecting the output of the ith neuron to the input of the jth neuron, and A is a positive constant controlling the decay of the output in the absence of any other inputs. Going back to the analogy of Pavlov's dog, the plate of food can be viewed as the external input, the bell ringing can be the input from another neuron, and the output of the neuron will be active if the dog is salivating. The second term on the right side of equation (11) indicates that the dog will salivate when the plate of food is presented to the dog. The last term indicates the effect of the bell ringing on the dog's salivation. The first term indicates that the dog will eventually discontinue salivating if it is not sufficiently stimulated by either the plate of food or the bell. Notice that equation (11) contains the capability for self feedback if $w_{ii} \neq 0$. In order to make the equation (11) more accurately represent the biological phenomenon, a time delay is often introduced into the last term on the right hand side. This time delay accounts for the fact that it takes a finite amount of time for the output of a neuron to reach the input of another neuron. Sometimes a threshold is added to the last term to reduce the effects of noise. Only if the output of a neuron is above the threshold will its output contribute to the input of another neuron. This is equivalent to saying that a neuron must be sufficiently active before it will affect any other neurons. These two subtle points will be excluded in the discussion in this report.

Next consider the equation which controls the learning of the network. The weight from the ith neuron to the jth neuron is adapted according to

$$\frac{\partial w_{ij}(t)}{\partial t} = -F\,w_{ij}(t) + G\,y_j(t)\,y_i(t)\,, \tag{12}$$

where F and G are positive constants, and y_j(t) is the output of the jth neuron. The first term on the right hand side of equation (12) allows the neuron to forget a relationship over time, with F controlling the forgetting rate. Typically this constant has a much smaller value than A. The second term is a simple way to implement Hebb's Law, with G controlling the learning rate. If both the input signal and the output signal are large then the weight change will be large. If either signal is small then the weight change will be small. As with the output equation, terms could be added to account for delays and thresholds.

It is important to choose the values of the various constants appropriately. If the forgetting constant, F, is too large the neuron will never be able to learn. If the constant, A, controlling the gain of the output decay is too small the network will have problems becoming inactive in a sufficiently short time interval. Most applications use values for A on the order of milliseconds, while F will be on the order of seconds [5, Part V, November 1988]. Usually the output values are restricted to the range $0 < y_i(t) < +1$. G is chosen to be positive since a negative value would lead to changes opposite of the intended effect. As mentioned above the learning rate is proportional to G, which typically takes on values in the range $0 < G < 1$. It is often advantageous to pick F and G such that if the stimulus to a neuron is high and its output is low the tendency of the weights to decay will overcome any small gain in the learning term.

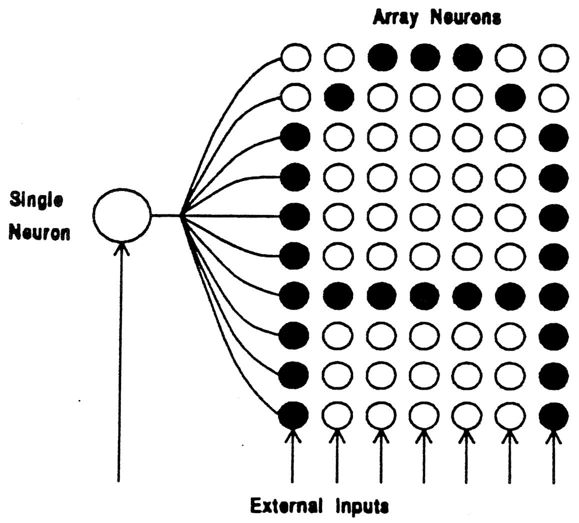


Figure 12: Network with Hebbian Learning

Consider the network in Figure 12 as example of Grossberg learning in action. The network consists of 70 neurons in a 7 x 10 array, and a single neuron not in the array. The single neuron's output is connected as an input to each of the array neurons, but none of the array neurons receive inputs from the other array neurons. Each of the 71 neurons in the network also has a unique external input such that whenever the external input is active, the output of the neuron will also be active. Instead of operating asynchronously in continuous time the network will operate synchronously in discrete time, an assumption which does not change the qualitative operation of the network. The network can be visualized as 70 independent two neuron networks using the Grossberg learning law. Begin the network with small random weights. During the training phase the external inputs are used to stimulate the single neuron and to stimulate the array with the pattern shown in Figure12. The array neurons will adjust their output and weight, from the single neuron, using the Grossberg learning law equations. The array neurons which are not stimulated by an external input have a small output and their weight slowly decays, if G and F are picked appropriately. This causes the array neuron to learn not to respond to the single neuron's stimulation. Those array neurons stimulated by an external input have large outputs and increase the strength of their weight, learning to respond to the stimulus of the single neuron. If the above external stimulations are repeated a sufficient number of times, the array neurons eventually learn to produce the pattern when only the single neuron stimulates them. The network is trained to respond to the conditioned stimulus in the absence of the unconditioned stimulus, i.e. the single neuron in the absence of the external stimuli. However, this network is not capable of much as it currently stands. By adding additional single neurons, and retraining the array neurons to respond with a distinct pattern when a different single neuron is activated the network is made more useful. When training a new pattern the weights to the prior single neurons will remain virtually unchanged if the values of G and F are chosen appropriately.

The Grossberg Learning laws have been criticized by biologist for not including enough factors to make it truly match the biological phenomenon. Never the less, Grossberg Learning laws have been used quite successfully in neural networks. John Hopfield used equations similiar to Grossberg's to determine the weights necessary to implement a content addressable memory using the Hopfield model. The Hopfield model is discussed in section 4.3.

4.2 Kohonen

The Kohonen self–organizing network is interesting since it uses unsupervised learning and organizes itself to topologically represent characteristics of the input patterns. The Kohonen network is very powerful and fast, however, it is tricky to apply to problems. The following discussion will not seek to fully explain all the intricacies involved in self–organizing networks, but rather explain the operation of Kohonen self–organizing networks and give some simple examples to illustrate their overall qualitative operation. See [23, 24, 26] for more detailed information on the Kohonen self–organizing network. This section will only discuss the one–dimensional Kohonen network but, the description generalizes to larger dimensional systems.

A Kohonen network is a fully interconnected array of neurons, i.e. the output of each neuron is an input to all of the other inputs in the network, including itself. Any of the the interconnection strengths could be zero. Each neuron has two sets of weights: one set used to compute the sum of weighted external inputs, and another to control the interactions between neurons in the network. The weights on the input pattern are adjustable, while the weights between neurons are fixed. Each neuron receives an identical input pattern, A, consisting of n elements. A diagram of a simple

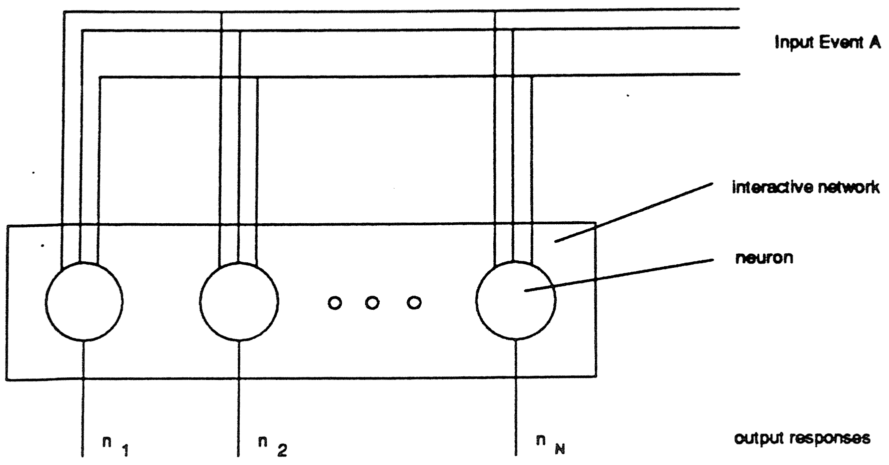


Figure 13: Kohonen Network

Kohonen network with N neurons is shown in Figure 13. During each presentation, the complete input pattern is presented to each neuron and each neuron computes its output as a sigmoidal function on the sum of its weighted inputs. The input pattern is then removed and the neurons interact with each other to locate the neuron with the largest output and to allow only that neuron to output. The interaction also isolates a neighborhood of that neuron, and allows only that neuron and its neighbors to adjust their input weights. The adjustment is done such that the neurons adjusting their input weights would have a larger output if the exact same input pattern was presented again. Consider the simple example of a Kohonen network where the input pattern has only one component. The weights for the external input to each neuron, w_i for $i = 0, \dots, N$, are initially random numbers between zero and one with a uniform distribution. A sequence of input patterns is presented to the network. During any one presentation only one neuron is allowed to output. As will be shown later the value of the output is not as important as the location of the neuron in the network. The inputs for this example are values between zero and one with a uniform random distribution. It can be shown that after a sufficient number of input presentations the network adjusts the weight of each neuron such that either $w_i < w_j \; \forall \; i < j$, or $w_i < w_j \; \forall \; i < j$, where $1 \leq i,j \leq N$, see [23]. More interesting is that the weights are uniformly distributed throughout the input range, as shown in the simulation results presented in Figure 14 . The graph shown in Figure 14 is rotated such that the dependent axis is the horizontal axis. Each vertical line represents one neuron. The horizontal position of the line indicates the value of the weight while the height of the line corresponds to the the neuron's number, i.e. the index i. As is shown, the neuron weights have adjusted to order themselves within the array. Fundamental to this ordering is the fact that physical neighbors to the neuron with the largest output respond similarly, which results from the strategy whereby only neighbors of the neuron with the largest output are allowed to update. Even though the network began with random weights, the network has self–organized in a fashion to represent the probability distribution of the input patterns. To further illustrate the point, if the input pattern had been gaussianly distributed rather than uniform the

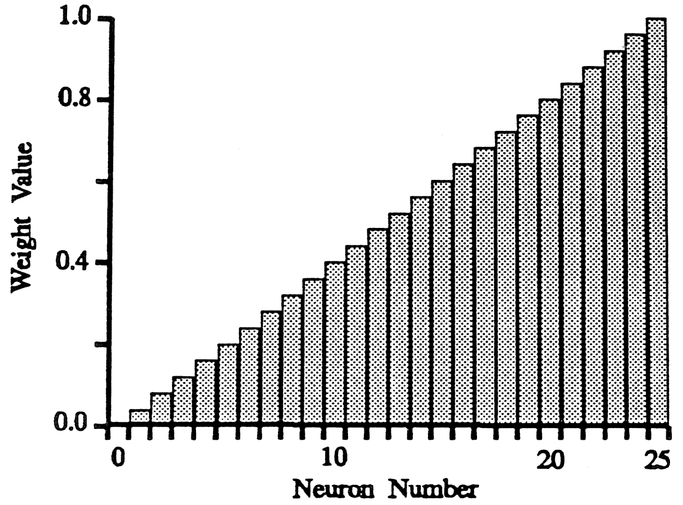


Figure 14: Kohonen Self–Organization Example 1

weight values would have been distributed as in Figure 15. The network has developed the weights such that the spacing now matches the gaussian distribution, i.e. more weights are located in regions where the inputs are most likely to occur. These two examples show the ability of a Kohonen network to develop the values of the weights to match a characteristic feature of the input patterns, e.g. the probability distribution. These examples demonstrate what is termed a topological mapping, i.e. that the order among the input patterns is directly related to the order within the network. This type of Kohonen network has been used as a vector quantizer, see [36]. The performance of this type of vector quantizer is very comparable to the LBG alogorithm for vector quantization.

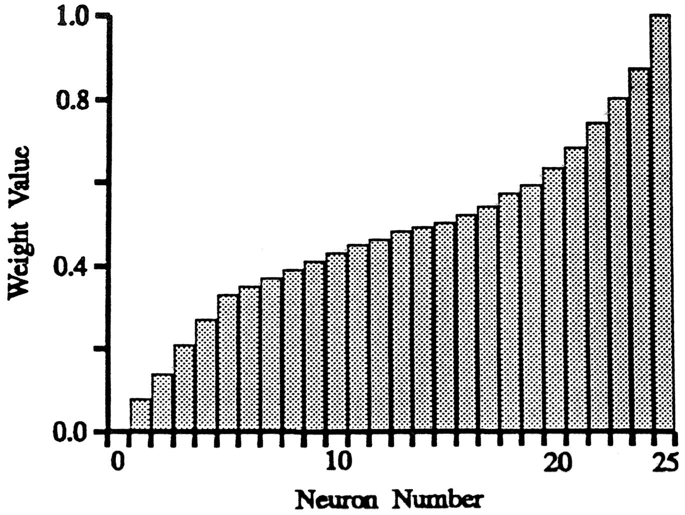


Figure 15: Kohonen Self–Organization Example 2

Now consider again the operations of the interactive network that are crucial to the correct operation of the Kohonen network beginning with the isolation, or clustering, of the neuron with the largest output and its neighbors. Clustering can be defined as an operation yielding a state in which the neuron with the largest output and neurons in some neighborhood of that neuron have been excited and all others neurons in the network have been inhibited. A neuron is said to be excited if its output is near its high asymptotic value, and inhibited if its output is close to its low asymptotic value. Clustering is accomplished using a lateral interaction procedure whereby each neuron excites itself and its neighbors and inhibits all others. The level of excitement and inhibition imparted by a given neuron is proportional to its current output. The principle behind the lateral interaction procedure was mentioned briefly above, that physical neighbors in the network should respond similarly to an input pattern. To explore lateral interaction consider Figure 16 which shows a detailed picture of the interactive network of Figure 13. The output of each neuron is fed back as inputs to each of the other neurons either in an inhibitatory or excitatory fashion, i.e. with negative or positive weights. The magnitude and sign of the feed–back weights are determined by a "Mexican Hat" function, an example of which is shown in Figure 17. These weights are not subject to any learning mechanism. This type of interaction allows each neuron to excite itself and its "close" neighbors and inhibit those neurons that are further away. This process can also be

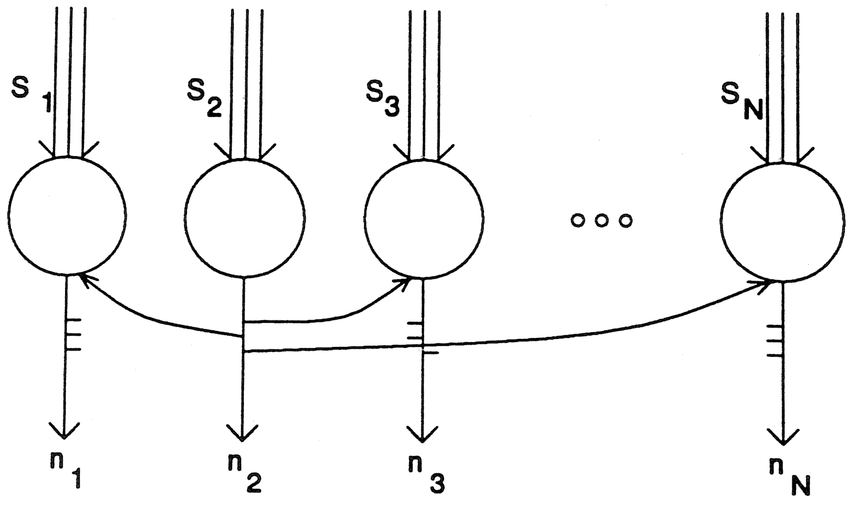


Figure 16: Interactive Network

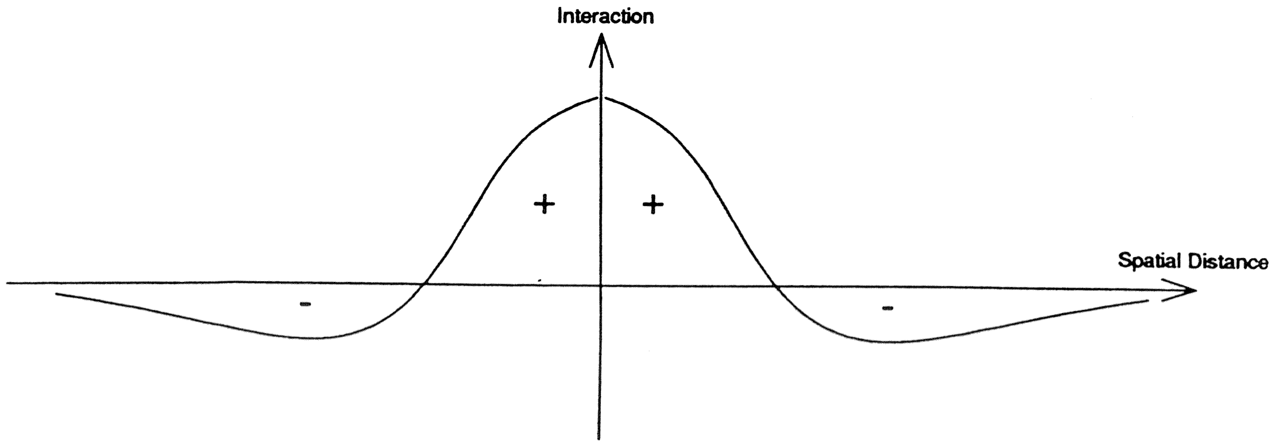


Figure 17: "Mexican Hat" Function

thought of as a competition among the neurons, with each neuron competing to have the largest output. The output of the ith neuron can be described by

$$n_i(t) = \sigma\left[\sum_{k=1}^{N} \gamma(k-i)\, n_k(t-t_0)\right] \tag{13}$$

where N is the number of neurons in the network,$\gamma(k)$ is the "Mexican hat" function, t_0 is the small transmission delay in the feed–back links, and $\sigma()$ is a sigmoidal function. Figure 18 shows an example of the functioning of lateral interaction. The curve for $t = 0$ indicates the initial outputs of the neurons before lateral interaction has begun. The other curves show the outputs of the neurons as lateral interaction proceeds. As shown, a clustering of the neurons occurs as desired. The neuron that originally has the largest output will continue to have the largest output throughout the entire clustering operation. The above clustering procedure is stable, and without any other factors included the output values will never be able to decay and the network will never be able to relax and react to a new input pattern, see [23]. "Fatigue" effects, or temporary effects to weaken the connections under persistent activity can be included in equation (13) to accomplish this decay.

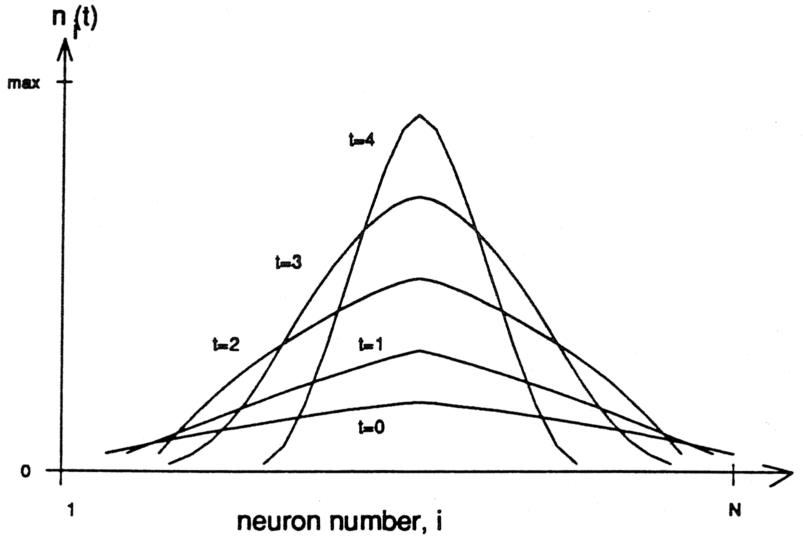


Figure 18: Lateral Interaction Example

Another issue is the width of the neighborhood. In practice, the neighborhoods begin fairly large and then shrink over time. Therefore, while the interconnection weights are not adjusted via a learning mechanism, they do in general change in time to reflect the changing size of the ''Mexican Hat''. Since the neurons start off with random weights a sufficiently large neighborhood must be used initially to allow the network to order the weights properly. If the neighborhood is not made smaller over time, then the weights of the neurons will not be able to converge to their final values. In the worst case where all neurons are always in the neighborhood, the weights of all neurons will converge to the same values. Figure 19 shows the gaussian input example discussed earlier when the original neighborhood size is not large enough. The weights are chosen correctly, but proper ordering in the network is only partially achieved. For networks of higher dimensions, the neighborhood shape is also an issue. The shape will ultimately dictate how the neurons represent the characteristic features of the inputs. Different shaped neighborhoods will results in networks with slightly different weights, i.e. different topological mappings result.

A number of simplifications can be used when simulating lateral interaction. The particular function used to control the degree of interaction is relatively unimportant. There are a wide range of functions for which the clustering is virtually identical. For computer simulations, the algorithm can simply pick the neuron with the largest output and take all neurons within a prescribed neighborhood. This procedure allows the necessary clustering to occur, and network development occurs in the same fashion as if a more complicated function had been used.

The second operation, the adjustment of the weights, is more straightforward than lateral interaction. Since we want the neuron with the largest output and its neighbors to react more

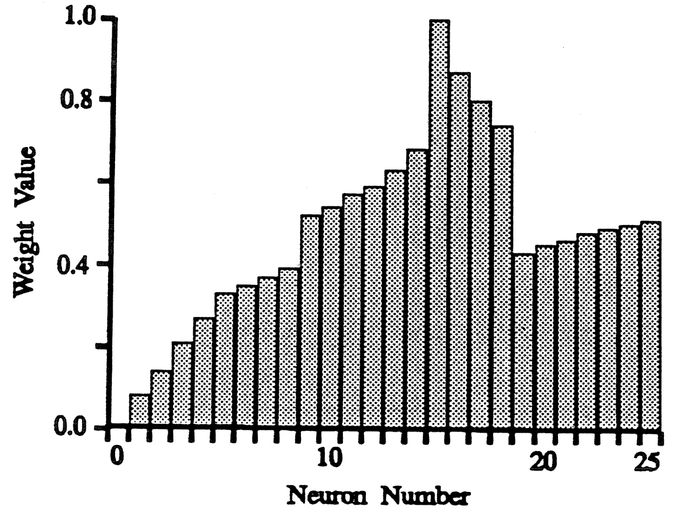


Figure 19: Kohonen Self–Organization Example 3

strongly to the presented input, the following weight adjustment is used for the ith external input to the jth neuron

$$w_i^{\,j}(t_{l+1}) = w_i^{\,j}(t_l) + \alpha(t_l)\,[x_i(t_l) - w_i^{\,j}(t_l)]\,, \tag{14}$$

where x_i is the ith component of input pattern A and $\alpha\ (t_l\,)$ is the learning gain. Each of the neurons in the cluster uses the above equation to adjust its weights. This update equation moves the weights of the updating neurons closer to the current input pattern. Thus, if the same input pattern is presented again, each of those neurons reacts more strongly to the input. The learning gain is usually a slowly decreasing function of time. Practically, any decreasing function will work and often a linear function of time is used.

Proper choice of the neighborhood size and the learning gain are often determined through experience. The network development consists of two phases; initial formation of the correct order, and final convergence of the map. Early in the network development neighborhood size tends to be more important than learning gain. This is because early on it is important that the network correctly order itself. Later in the network development a smaller gain is useful to allow the weights to converge to their asymptotic values.

A more general Kohonen network is shown in Figure 20. This network has added the use of a preprocessor to the network. The neurons no longer receive the same input pattern. The preprocessor takes each external input pattern, A, and generates a separate internal input pattern, S_i, for each of the neurons in the network. The only requirement regarding these internal input patterns, S_i, is that they be correlated with each other. This requirement of correlation ensures that the relationships among the external input patterns, A, are not distorted. If these relationships are distorted the resulting topological map will not accurately represent the characteristic features of the external inputs. This more general Kohonen network is equivalent to the network shown in Figure 13, if the preprocessor simply passes the external input pattern, A, onto each and every neuron, i.e. $S_i = A\ \forall\ i$. The Kohonen self–organizing network has some capabilities which can be extremely useful. One possible application mentioned above is vector quantization. The network can also be used to do dimension reduction and feature extraction. Often large dimensional data needs to be analyzed to extract the characteristic features. A Kohonen network will reduce the data into a topological map representing those features. This application is referred to as principle component analysis, see [32]. The Kohonen network can also be used to represent complicated hierarchical relations of high–dimensional data in a lower dimensional display, see[23].

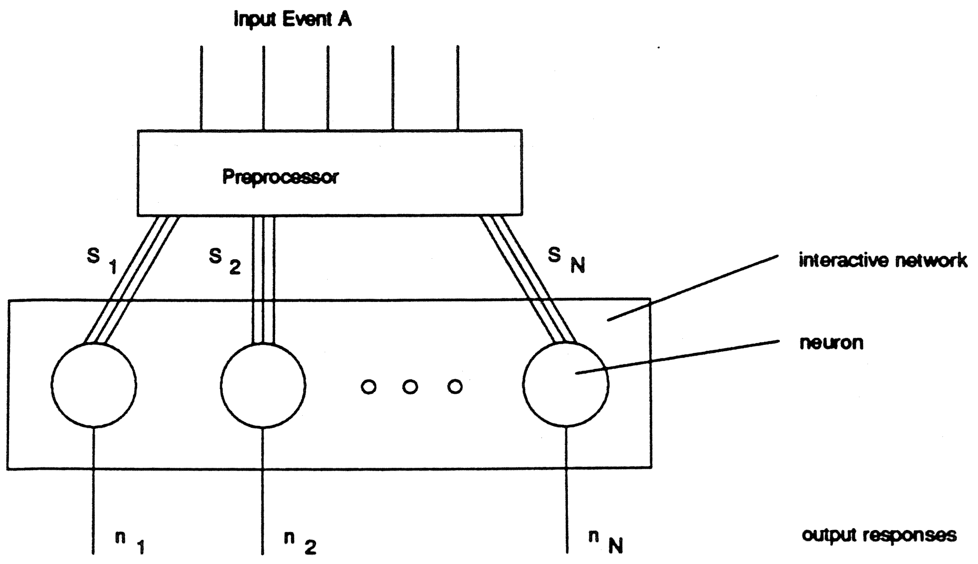


Figure 20: General Kohonen Network

4.3 Hopfield Model

The general structure of the continuous deterministic Hopfield model is shown in Figure 21. This model was first proposed by Hopfield in 1984; see [16]. Previously in 1982 Hopfield had proposed a discrete stochastic model; see [15]. Hopfield has shown that many important characteristics of the continuous deterministic model and discrete stochastic model are closely related; see[16]. Therefore for computer simulations the simpler discrete stochastic model is often used. This report discusses only the continuous deterministic model since it is the model that is likely to be implemented in actual hardware. From this point forward, the term Hopfield model will refer to the continuous deterministic Hopfield model.

A set of nonlinear dynamical equations describe the output of each of the amplifiers, i.e neurons, in the network. The equation describing the input to the ith amplifier, u_i, is

$$C_i\left(\frac{d\,u_i}{d\,t}\right) = \sum_j T_{ij} v_j - \frac{u_i}{\tau_i} + I_i \quad \text{with} \quad v_i = g_i(u_i) \tag{15}$$

where C_i is the total input capacitance of the amplifier, T_{ij} is the value of the connection from the output of the jth amplifier to the input to the ith amplifier, v_j is the output of the jth amplifier, τ_i is a resistance value defined below, $g_i()$ is the sigmoidal transfer function of the ith amplifier assuming a negligible response time, and I_i is the external input to the ith amplifier. Note that previously in this report, T_{ij} was the connection strength between the output of the ith amplifier and the input of the jth amplifier. In this section it refers to the connection strength between the output of the jth amplifier and the input of the ith amplifier. While this change of notation can be confusing, it is the notation used in much of the literature. This point is unimportant for Hopfield, since he assumes symmetric interconnections. In order to allow T_{ij} to have both negative and positive values, both a non–inverting and an inverting amplifier are used. When a negative T_{ij} is needed the output of the inverting amplifier is used, and when T_{ij} is positive the output of the noninverting amplifier is used. From this point forward the term amplifier will refer to either the non–inverting amplifier or the inverting amplifier depending on the sign of T_{ij}. The magnitude of T_{ij} will be $1/R_{ij}$ where R_{ij} is the value of the resistor connecting the output of the jth amplifier with the input of the ith amplifier. The value of τ_i is given by

$$\frac{1}{\tau_i} = \frac{1}{\rho_i} + \sum_j \frac{1}{R_{ij}}, \tag{16}$$

where ρ_i is the input resistance of the amplifier.

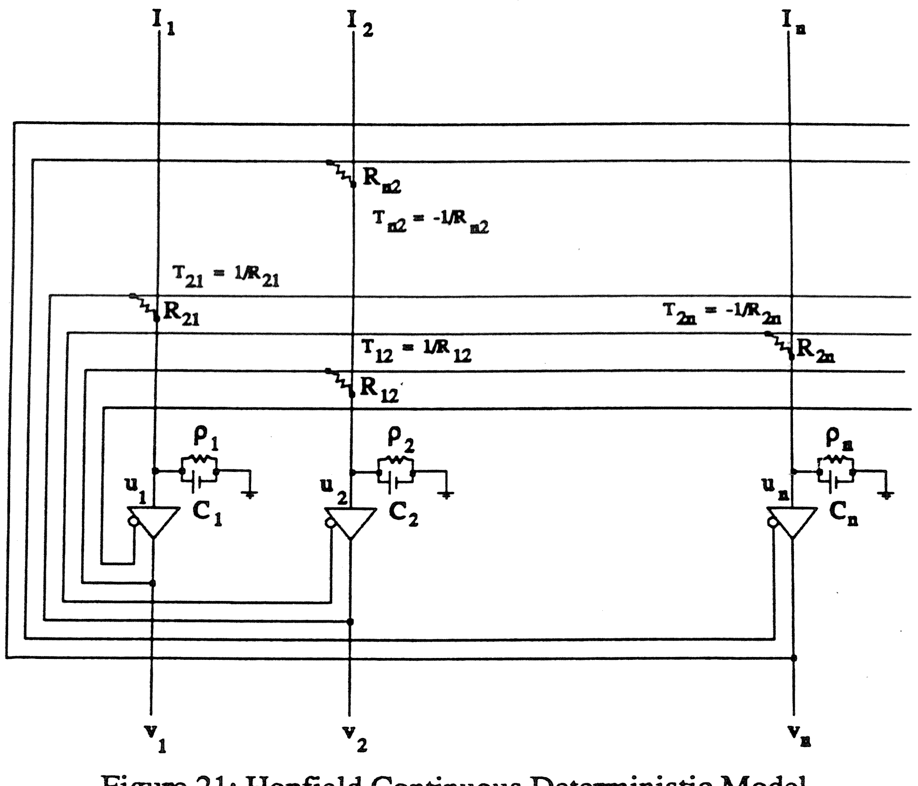


Figure 21: Hopfield Continuous Deterministic Model

The Hopfield network operates in a manner which is qualitatively different than the other networks discussed so far. To understand the operation of the Hopfield model consider the output value of the ith amplifier as the ith component of an N dimensional state vector, where is the number of neurons in the network. The state vector and equation (15) together deterministically describe the network's operation. The state of the network can be thought of as moving in state space, with equation (15) determining the direction of motion. The user picks values of the interconnection strengths to specify the state space flow, and therefore the operational characteristics of the network. In general, these values are fixed prior to the operation of the network. Consider the following positive--definite Lyapunov function for the system described in equation (15)

$$E = -\frac{1}{2}\sum_i \sum_j T_{ij} v_j v_i + \sum_i \frac{1}{\tau_i} \int_0^{v_i} g_i^{-1}(V)\, dV + \sum_i I_i v_i . \qquad (17)$$

Assuming that the matrix $T = [T_{ij}]$ is symmetric, then the time derivative of E is

$$\frac{dE}{dt} = -\sum_i \frac{dv_i}{dt} \left(\sum_j T_{ij} v_j - \frac{u_i}{\tau_i} + I_i \right) = -\sum_i \left(\frac{dv_i}{dt}\right) \left[C_i \left(\frac{du_i}{dt} \right) \right] . \qquad (18)$$

Recalling that $v_i = g_i(u_i)$,

$$\frac{dE}{dt} = -\sum_i C_i \dot{g}_i^{-1}(v_i) \left(\frac{dv_i}{dt} \right)^2 , \qquad (19)$$

where $\dot{g}_i^{-1}()$ is the first derivative of $g_i^{-1}()$with respect to v_i. Since $g_i()$ is a sigmoidal function, $g_i^{-1}()$ is a monotone increasing function so each term in the sum in equation (19) is nonnegative. Therefore $dE / dt \leq 0$, and $dE / dt = 0$ only when $dv_i/dt = 0 \ \forall$ i. The points where $dv_i / dt = 0$ are asymptotically stable equilibrium points for the E function in state space and represent the set of possible final outputs of the network. Since E is a bounded function, the above equation shows that the network always is attracted to a local minimum of the E function. One of the questions that remain is to pick the T_{ij} to arbitrarily place equilibrium points anywhere desired in the state space.

Before tackling that question, first consider the following example. The standard type of memory used in conventional computing is address addressable memory, where the complete address **must** be supplied in order for the correct memory to be recalled. Consider the use of a Hopfield network to implement a content addressable memory. A content addressable memory (CAM) is a memory device that uses portions of the content of the stored memory to recall the complete memory. For example a CAM could contain the entire bibliography for this report. The network could be designed to recall one bibliographic entry given the author's last name and the year of publication. Assume that all these entries have been stored in a Hopfield network and that no extra spurious states have been stored. An input to the network would consist of those two items with all the rest of the fields given as don't cares. This input would be the initial state of the network. The network is then allowed to change its state, according to equation (15), until an equilibrium point is reached. This final state will represent the entire bibliographical entry. One question that arises is, "Will this always work?" The answer is no. Notice that there are two items written by Hecht–Nielson in 1987. Giving the network only that much information would not allow the network to distinguish between the two entries. Starting with that information would usually result in one of the entries being recalled, and the other would only be retrievable if more information is given, e.g. a portion of the title. Now consider Hopfield's method for determining the T_{ij} to implement the CAM, see [15, 18]. For a CAM it is desired to store a set of stable states $\{X^{s} : s = 1, 2, \dots ,m.\}$ that correspond to the equilibria of an E function in state space, where output values are +1 or -1. Since equation (17) is positive definite a good choice for E is

$$E = -\frac{1}{2}\sum_{s=1}^{m}(X^{s}\cdot V)^{2}, \tag{20}$$

where **V** is a vector representing the current state of the network. If the current state vector, **V**, of the network is a random vector which is not close to any of the stored stable states, then each term in the sum should be small. If the current state vector is near one of the stored stable states, $\mathbf{X}^{s}$, then one term in the sum has a value approximately equal to N^2, where N is the number of neurons in the network, and the other terms will be the small or zero, assuming the stored stable states are uncorrelated. Thus, E will have a local minimum of approximately $-1/2N^2$ at each of the stored stable states, see [18]. If the slope of the sigmoidal transfer function of the amplifiers is large, then the second term of equation (17) is negligible, and equation (20) can be rewritten into the form of equation (17) if

$$T_{ij} = \sum_{s=1}^{m} X_i^{\,s} X_j^{\,s}, \tag{21}$$

with $I_i = 0\ \forall$ i, where X_i^s is the ith component of the stable state $\mathbf{X}^s$. This exemplifies one method of mapping a given problem description onto the Hopfield network model. Other problems which have been solved, either optimally or suboptimally, using this method are the traveling salesperson problem [17,18], an A/D converter [48], signal decompostion [48], linear programming [48], a network communications routing algorithm [40,55], and various combinatorial optimization problems [29,46]. There are some difficulties however, with this method of mapping the problem description onto a function E. The first is that the problem must be describable by a positive–definite function. The second difficulty is that this approach does not guarantee that all the desired stable states have actually been stored in the network. Two nearby desired stable states might be sufficiently close that they have merged into one stable state. The third difficulty is that the approach does not control the number of spurious unplanned stable states also stored in the network. These spurious stable states can yield outputs which do not match any of the desired stable states. The symmetry constraint used earlier yields for every desired stored stable state another symmetric stored stable state, i.e. one for which every value has been reversed — every one is now a negative one, and every negative one is now a one. The fourth difficulty is that the symmetry constraint is difficult to realize in VLSI where it is impossible to manufacture two resistors with exactly the same values. The fifth difficulty is the number of stable states that a network can support. Hopfield has estimated this number at 15% of the number of neurons in the network, a number that has been experimentally confirmed by others. This number unfortunately also includes unwanted states.

Some current research on the Hopfield model has attempted to ease some of these constraints by removing the no self–feedback constraint, increasing the number of storable states, ensuring that all desirable states can be stored, limiting the number of unwanted states, and removing the symmetry constraint. See [8, 10, 30, 31, 34].

5 Conclusions

Neural networks can be traced back to the nineteenth century. Many of the concepts behind neural networks have been taken from the work of biophysiologists and scientists who attempt to accurately model the operation of the human brain. Today's neural networks use those concepts in addition to other nonbiological concepts to create powerful computational architectures.

Broadly speaking, neural networks process inputs to produce outputs. Based on their processing topology neural networks can be divided into two classes: feedforward and feedback. The processing topology of feedforward networks is uni–directional, i.e. there are no loops around which signals propagate, while in feedback networks such loops do exist. The feedforward networks discussed are: Grandmother Cells, ADALINE, MADALINE and Backpropagation networks. Grandmother cell networks use unconnected individual neurons with fixed weights, and contain no ability to learn or adapt. The ADALINE (ADAptive LINear Element) is an extension of the grandmother cell using a normalized LMS (Least Mean Squares) algorithm to learn and adapt. The ADALINE network is capable of realizing only linearly separable problems. One way to realize nonlinearly separable problems is to use multiple ADALINEs in conjunction with logic gates, forming MADALINE(Many ADALINEs) networks. Backpropagation networks are another extension of ADALINE networks, consisting of multiple layers of ADALINEs and the addition of a nonlinear sigmoidal function. The backpropagation network uses a generalized LMS algorithm to adjust its weights, and can realize any nonlinearly separable problem, given that the inputs satisfy some conditions.

In a feedback network any given neuron is allowed to influence its own output directly through self feedback or indirectly through the influence it has on other neurons from which is receives inputs. The feedback networks discussed are: Kohonen Self–Organizing, and Hopfield networks. The Kohonen Self–Organizing network uses unsupervized learning and organizes itself to topologically represent characteristics of the input patterns. The Hopfield network operates by changing its current state in order to minimize the overall energy of the network. The Hopfield network is often employed to perform optimization problems. A good solution to the traveling salesman problem has been found using the Hopfield network.

Grossberg Learning, discussed since it arises periodically in the literature, is a mathematical formulation of Hebb's law which explains the phenomena of Pavlov's dog. Grossberg learning is not dependent upon a given network architecture, but is generally applicable.

The hardware, i.e. VLSI technology, has yet to keep pace with the simulation based neural network research. Researchers are proposing network architectures which are well beyond the current hardware capabilities. Much research has been done on neural network research, but to date no significant break throughs have been made. The number of neurons available to date on a chip is well below the number required for interesting and intricate problems. The two largest problems facing VLSI designers are the enormous number of interconnections desired, and the ability to realize adjustable weights with out requiring vast amounts of silicon real estate.

Appendix A Delta Rule as a Gradient Descent Method

For the following derivation consider presenting a set of input and true output pairs to a neuron. The neuron computes the dot product of the current input pattern with the current weight vector to produce the estimated output value. This estimate is subtracted from the true output value to generate an error value. The weights are then adjusted to reduce the error. The rule for adjusting the weights can be written as

$$\Delta w_j = \beta \left(d(p) - y(p) \right) I_j(p) = \beta \, \delta(p) \, I_j(p) , \tag{22}$$

where *d(p)* is the desired output for the pth input pattern, *y(p)* is the estimated output for the pth input pattern, $I_j(p)$ is the jth component of the pth input pattern, β is the learning gain, and Δw_j is the change to the jth weight.

It will be shown that the Delta rule minimizes the square of the differences between the estimated output and the true output. In order to prove that the Delta rule is a gradient descent method it will be shown that the derivative of the error measure with respect to each weight is negatively proportional to the weight change in the Delta rule. Let

$$E(p) = \frac{1}{2} [\, d(p) - y(p) \,]^2 \tag{23}$$

be the measure of the error for the pth pattern and define the overall error as $E = \sum_p E(p)$. Using the chain rule, the derivative of *E(p)* with respect to the jth weight is

$$\frac{\partial E(p)}{\partial w_j} = \frac{\partial E(p)}{\partial y(p)} \frac{\partial y(p)}{\partial w_j} . \tag{24}$$

The first term on the right hand side is the change in the error with respect to the estimated output, the second term is the change in the estimated output with respect to the jth weight. From equation (23)

$$\frac{\partial E(p)}{\partial y(p)} = - \, [\, d(p) - y(p) \,] = - \, \delta(p) . \tag{25}$$

Since

$$y(p) = \sum_j w_j I_j(p) , \tag{26}$$

the second term of equation (24) is

$$\frac{\partial y(p)}{\partial w_j} = I_j(p) . \tag{27}$$

Substituting equations (25) and equation (27) back into equation (24)

$$-\frac{\partial E(p)}{\partial w_j} = \delta(p)\, I_j(p)\,. \tag{28}$$

Comparing this with equation (22) it has been shown that the delta rule changes the weights along the negative derivative of the squared error with respect to each weight, as desired. Combining equation (28) with the observation that

$$\frac{\partial E}{\partial w_j} = \sum_p \frac{\partial E(p)}{\partial w_j} \tag{29}$$

shows that the net change in w_j after one complete cycle of pattern presentations is proportional to this derivative and hence that the Delta rule implements a gradient descent on E.

Appendix B The Generalized Delta Rule

For the following derivation the backpropagation network presented in section 3.3 will be assumed. However, the network will be allowed to have multiple hidden layers. Consider presenting a set of input and true output pairs to the network. For each input pattern presented an estimated output vector will be computed by a forward pass through the activation functions of each successive layer. Each of the neurons in the network will compute the following weighted input

$$net_j(p) = \sum_i w_{ij} y_i(p) , \tag{30}$$

where $net_j(p)$ is the net input to the jth neuron in the current layer for the pth pattern, wij is the interconnection weight from the ith neuron in the previous layer to the jth neuron in the current layer, and $y_i(p)$ is the output of the ith neuron in the previous layer for the pth pattern. If neuron i is an input unit then $y_i(p) = I_i(p)$, where $I_i(p)$ is the ith component of the pth input pattern. The output of the neuron will be

$$y_j(p) = f_j(net_j(p)) , \tag{31}$$

where f is a sigmoidal function. From Appendix A, the generalized delta rule will be an estimated gradient descent method if

$$\Delta w_{ij} \ \alpha \ - \frac{\partial E(p)}{\partial w_{ij}} , \tag{32}$$

where E and $E(p)$ are the same error functions defined in Appendix A. Using the chain rule, the derivative of $E(p)$ with respect to the weights is

$$\frac{\partial E(p)}{\partial w_{ij}} = \frac{\partial E(p)}{\partial net_j(p)} \frac{\partial net_j(p)}{\partial w_{ij}} . \tag{33}$$

The last term of equation (33) is

$$\frac{\partial net_j(p)}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k w_{kj} y_k(p) = y_i(p) . \tag{34}$$

Define

$$\delta_j(p) = - \frac{\partial E(p)}{\partial net_j(p)} .$$

Equation (33) can be rewritten as

$$- \frac{\partial E(p)}{\partial w_{ij}} = \delta_j(p)\, y_i(p)\,, \tag{35}$$

which corresponds to the Delta rule. However, it has not been determined how to calculate $\delta_j(p)$ for each neuron in the network. To compute $\delta_j(p)$ apply the chain rule to yield

$$\delta_j(p) = - \frac{\partial E(p)}{\partial net_j(p)} = - \frac{\partial E(p)}{\partial y_j(p)} \frac{\partial y_j(p)}{\partial net_j(p)}\,. \tag{36}$$

Using equation (31), the second part of equation (36) is

$$\frac{\partial y_j(p)}{\partial net_j(p)} = f_j\,'(\, net_j(p)\,)\,. \tag{37}$$

Consider two cases for the first term. First assume the neuron is an output neuron, so that

$$\frac{\partial E(p)}{\partial y_j(p)} = - \,(\, d_j(p) - y_j(p)\,)\,. \tag{38}$$

Substituting equations (37) and (38) into equation (36) we get

$$\delta_j(p) = (\, d_j(p) - y_j(p)\,) f\,'(\, net_j(p)\,) \tag{39}$$

for any output neuron. Second, assume the neuron is not an output neuron, then using the chain rule

$$\begin{aligned} &\sum_k \frac{\partial E(p)}{\partial net_k(p)} \frac{\partial net_k(p)}{\partial y_j(p)} \\ &\qquad = \sum_k \frac{\partial E(p)}{\partial net_k(p)} \frac{\partial}{\partial y_j(p)} \sum_i w_{ik}\, y_i(p) \\ &\qquad = \sum_k \frac{\partial E(p)}{\partial net_k(p)} w_{jk} \\ &\qquad - \sum_k \delta_k(p)\, w_{jk}\,, \end{aligned} \tag{40}$$

where w_{jk} is the weight connecting the jth neuron in the current layer to the kth neuron in the next layer, and the sum is over all the neurons in the next layer. Substituting equations (37) and (40) into equation (36) we get

$$\delta_j(p) = f_j\,'(\, net_j(p)\,) \sum_k \delta_k(p)\, w_{jk}\,, \tag{41}$$

for any neuron which is not an output neuron. If the above equations are used for $\delta_j(p)$ then the generalized delta rule approximates a gradient descent method.

References

[1] James Anderson and Edward Rosenfeld. *Neurocomputing: Foundations of Research.* The MIT Press, Cambridge, 1989.

[2] Michael Arbib. *Brains, Machines and Mathematics.* Springer-Verlag Inc, New York, 2nd. edition, 1987.

[3] Eric Baum. *On the Capabilities of Multilayer Perceptrons.* Jet Propulsion Laboratory, California Institute of Technology.

[4] Gail Carpenter and Stephen Grossberg. "The ART of Adaptive Pattern Recognition by a Self–Organizing Neural Network". *Computer,* 77–88, March 1988.

[5] Maureen Caudill. "Neural Networks Primer: Parts i – vii." *AI Expert,* 1987 – 1989.

[6] G Cybenko. *Approximation by Superpositions of a Sigmoidal Function.* Center for Supercomputing Research and Development, University of Illinois, Urbana, 1988.

[7] U.S. Defense Advanced Research Projects Agency (DARPA). *DARPA Neural Networks Study.* AFCEA International Press, Fairfax, 1988.

[8] Jay Farrell. *Analysis and Synthesis Techniques for Two Classes of Nonlinear Dynamical Systems: Digital Controllers and Neural Networks.* PhD thesis, University of Notre Dame, May 1989.

[9] Feldman, Fanty, and Goddard. "Computing with structured Neural Networks." *Computer,* 91–103, March 1988.

[10] Donald Gray, Anthony Michel, and Wolfgang Porod. *A Neural Network Design Procedure for Sorting Problems.* Technical Report, University of Notre Dame, Department of Electrical Engineering.

[11] Donald Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[12] Robert Hecht–Nielson. "Counterpropagation Networks." *Neural Networks,* 1987.

[13] Robert Hecht–Nielson. "Kolmogorov's Mapping Neural Network Existence Theorem." In *Proceedings IEEE International Conference on Neural Networks,* 1987.

[14] Robert Hecht–Nielson. "Neurocomputer Applications." In Eckmiller, Rolf, and von~der~Malsburg, editors, *Neural Computers,* Springer–Verlag, 1988.

[15] John Hopfield. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities." *Proceedings of the National Academy of Science,* 79: 2554–2558, April 1982.

[16] John Hopfield. "Neurons with Graded Response have Collective Computational Properties Like Those of Two–State Neurons." *Proceedings of the National Academy of Science,* 81:3088–3092, May 1984.

[17] John Hopfield and David Tank. "Computing with Neural Circuits: A Model." *Science,* 233:625–633, 1986.

[18] John Hopfield and David Tank. " 'Neural' Computation of Decisions in Optimization Problems." *Biological Cybernetics*, 52:141–152, 1085.

[19] W Huang and R Lippman. *Neural Net and Traditional Classifiers*. Technical Report, MIT Lincoln Laboratory, 1987.

[20] Bunpei Irie and Sei Miyake. "Capabilities of Three-layered Perceptrons." In *Proceedings IEEE International Conference on Neural Networks*, 1988.

[21] William James. *Psychology (Briefer Course)*. Holt, New York, 1890.

[22] Casey Klimasauskas. "Neural Nets and Noise Filtering." *Dr. Dobb's Journal of Software Tools*, 32, January 1989.

[23] T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, New York, 2 edition, 1988.

[24] Teuvo Kohonen. "Automatic Formation of Topological Maps of Patterns in a Self–organizing System." In *Proceedings Second Scandinavian Conference on Image Analysis*, pages 214–220, 1981.

[25] Teuvo Kohonen. "An Introduction to Neural Computing." *Neural Networks*, 1 (1): 3–16, 1988.

[26] Teuvo Kohonen. "Self–Organized Formation of Topologically Correct Feature Maps." *Biological Cybernetics*, 43:59–69, 1982.

[27] Kuczewski, Myers, and Crawford. "Exploration of Backpropagation as a Self–Organizational Structure." In *Proceedings of the International Neural Network Society*, 1987.

[28] S Kung and J Hwang. *An Algebraic Projection Analysis for Optimal Hidden Units Size and Learning Rates in Backpropagation Learning*. Technical Report, Department of Electrical Engineering, Princeton University, 1987.

[29] Bernard Levy and Milton Adams. "Global Optimization with Stochastic Neural Networks." In *Proceedings of the International Neural Network Society*, 1987.

[30] Jian Li, Anthony Michel, and Wolfgang Porod. *Analysis and Synthesis of a Class of Neural Networks: Variable Structure Systems with Infinite Gains*. Technical Report, University of Notre Dame, Department of Electrical Engineering.

[31] Jian Li, Anthony Michel, and Wolfgang Porod. "Qualitative Analysis and Synthesis of a Class of Neural Networks." *IEEE Transactions on Circuits and Systems*, 976–985, August 1988.

[32] Ralph Linsker. "Self–Organization in a Perceptual Network." *Computer*, 105–117, March 1988.

[33] Warren McCulloch and Walter Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity." *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[34] Anthony Michel, Jay Farrell, and Wolfgang Porod. "Qualitative Analysis of Neural Networks." *IEEE Transactions on Circuits and Systems", 229–243, February 1989.*

[35] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, Cambridge, 1969.

[36] Nasser Nasrabadi and Yushu Feng. ''Vector Quantization of Images Based upon the Kohonen Self–Organizing Feature Maps.'' *Proceedings IEEE International Conference on Neural Networks*, 1988.

[37] N Nilsson. *Learning Machines*. McGraw Hill, New York, 1965.

[38] Sverrir Olafsson and Yaser Abu-Mostafa. ''The Capacity of Multilevel Threshold Functions.'' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(2): 277–281, March 1988.

[39] D Parker. *Learning Logic*. Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, April 1985.

[40] Herbert Rauch and Theo Winarske. ''Neural Networks for Routing Communication Traffic.'' *IEEE Control Systems Magazine*, 26–30, April 1988.

[41] W Ridgway. *An Adaptive Logic System with Generalizing Properties*. PhD thesis, Stanford University, April 1962.

[42] Frank Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1961.

[43] David Rumelhart, G.E. Hinton, and R.J. Williams. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. The MIT Press, Cambridge, 1986.

[44] J Sietsma and R Dow. ''Neural Net pruning — Why and How.'' In *Proceedings IEEE International Conference on Neural Networks*, 1988.

[45] F.W.~\ Smith. ''A Trainable Nonlinear Function Generator.'' *IEEE Transactions Automatic Control*, 11(2): 212–218, April 1986.

[46] Gene Tagliarini and Edward Page. ''Solving Constraint Satisfaction Problems with Neural Networks.'' In *Proceedings of the International Neural Network Society*, 1987.

[47] David Tank and John Hopfield. ''Collective Computation in Neuron Like Circuits.'' *Scientific American*, 104–114.

[48] David Tank and John Hopfield. ''Simple 'Neural' Optimization Networks: an A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit.'' *IEEE Transactions on Circuits and Systems*, 33(5): 533–541, 1986.

[49] Bernard Widrow. ''Adaline and Madaline — 1963.'' *IEEE–ICNN*, 1963.

[50] Bernard Widrow and Marcian Hoff. ''Adaptive Switching Circuits.'' In *Convention Record, Part 4*., pages 96–104, Institute of Radio Engineers, Western Electronic Show and Convention, IRE, New York, 1960.

[51] Bernard Widrow and Marcian Hoff. ''Generalization and Information Storage in Networks of Adaline 'Neurons'.'' In M.C. Yovitz, G.T. Jacobi, and G. Goldstein, editors, *Self–Organizing Systems*, Spartan Books, Washington, DC, 1962.

[52] Bernard Widrow and Samuel Stearns. *Adaptive Signal Processing*. Prentice Hall, Englewood, 1985.

[53] Bernard Widrow and R. Winter. ''Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition.'' *Computer*, 25–39, March 1988.

[54] A Wieland and R Leighton. "Geometric Analysis of Neural Network Capabilities.'' In *Proceedings IEEE International Conference on Neural Networks*, 1987.

[55] P. Melsa, J. Kenney and C. Rohrs. ''A Neural Network Solution for Routing in Three Stage Interconnection Networks.'' In *Proceedings of the 1990 Internation Symposium on Circuits and Systems*, May 1990.