Artificial Neural Network Algorithms

By

Samer N. Ramadan

Submitted in partial fulfillment of the requirements for the degree of Master in Science in Computer Science

Project Advisor: Dr. Nash'at Mansour

NATURAL SCIENCE DIVISION

LEBANESE AMERICAN UNIVERSITY

Beirut

October 1999

LEBANESE AMERICAN UNIVERSITY

GRADUATE STUDIES

We hereby approve the Project of

Samer N. Ramadan

candidate for the Master of Science degree*.

(signed) N. Mansour Signatures Redacted
(chair)

Signatures Redacted

_____

_____

_____

_____

date Oct 99

*We also certify that written approval has been obtained for any proprietary material contained therein.

I grant to the LEBANESE AMERICAN UNIVERSITY the right to use this work, irrespective of any copyright, for the university's own purpose without cost to the university or to its students, agents and employees. I further agree that the University may reproduce and provide single copies of the work, in any format other than in or from microforms, to the public for the cost of reproduction.

ARTIFICIAL NEURAL NETWORK ALGORITHMS

ABSTRACT

by

Samer N. Ramadan

Inspired by the architecture of the biological brain, artificial neural networks were designed to provide solutions for computationally demanding problems. Neural network architectures are based on wide-scale parallel computing, a feature that promises an increased computational power. In this project, we implement a Boltzmann Machine neural network for solving the Traveling Salesperson Problem (TSP), a constrained optimization problem. We also implement a Kohonen's Self-Organizing Map for solving the Character Recognition Problem, a pattern recognition problem. The same problem is also solved by implementing an Adaptive Resonance Theory network. Experimental results show that the execution time of a Boltzmann Machine network for solving the TSP problem increases at a high rate as the number of cities increases. Moreover, penalty and bonus parameter values have shown a limited effect on the network performance as long as the penalty parameter is greater than the bonus parameter. Experiments also show that higher initial temperature values decrease the probability of the network converging to a feasible solution.

Experimental work done on Kohonen's Self-Organizing Map for character recognition shows that using problem-related initial weight vectors rather than random values improves the ability of the network to recognize characters accurately. Moreover, the topology of the cluster

units and the radius of learning also play key role in the network performance. In Adaptive Resonance Theory network, experimental results demonstrate the ability of the user to control the degree of similarity that allows patterns to be clustered on the same unit. Moreover, the order of the input patterns and the number of output cluster units also proved to have an effect on the network's output.

To my parents and to Miss Rola Tayan

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Nasha't Mansour for his guidance throughout my M.S. studies. Thanks are also due to Dr. Ramzi Haraty for being on my project committee.

I would like to express my sincere gratitude to the Lebanese American University whose financial support during my graduate studies made it all possible.

Finally, I would like to thank my friends and family for their long support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traditional, sequential, and digital computing excels in many areas, but has been less successful for other types of problems. The development of artificial neural networks began approximately 50 years ago, motivated by a desire to try both to understand the brain and to emulate some of its functions. Their introduction has added a new dimension to the capabilities of computers and moved them from machines that can infinitely perform repetitive preprogrammed tasks to machines capable of doing more complex and challenging ones.

Millions of the daily tasks performed "effortlessly" by the human brain are categorized as "highly complex" tasks when measured by computer-capability scale. Things like recognizing an old friend that has not been seen for a long time or differentiating between different handwritings or any kind of pattern classification are common things encountered almost daily. What makes it easy for a human being to do these things and more difficult for a computer to do, in spite of the fact that the switching time of the components in modern electronic computers are more than seven orders of magnitude faster than the neurons of the human brain? [Veelenturf, 95]. The answers to such a question are widely diversified and fall mainly beyond the scope of this report. However, a major part of the answer lies within the difference in the architecture of computers and that of the human brain. Where as the response time of the individual neural cells is typically in the order of tens of milliseconds, the massive parallelism and interconnectivity observed in the biological systems evidently account for the ability of the brain to perform

complex pattern recognition in a few hundred milliseconds. In many real-world applications, computers are required to perform complex pattern recognition problems, such as the ones described above. Until recently computers were based on the Von Neumann architecture. They derived their performance from one or only a few central processors which carry out long sequential programs at extremely high speed. Therefore, signal propagation time within the computer has already begun to emerge as a limiting factor to further gains in speed [Veelenturf, 95]. A way out of this limitation is to abandon the Von Neumann architecture and instead apply a large number of computational processors working in parallel. Features from the physiology of the brain are therefore used as the basis of the new processing models. Hence, the technology has come to be known as Artificial Neural Systems (ANS) or simply neural networks.

1.1 Applications of Neural Networks

The study of neural networks is an interesting field, both in their development and in their application. Regardless of the domain they are used in, neural-networks solutions are suitable when the problem to be solved can not be formulated explicitly. If no known algorithm is found to solve a problem due to a noisy environment causing extreme non-uniformity, then solving the problem by presenting it to a neural network with a problem-specific architecture and training algorithm is favorable. Many of the most complex problems in areas such as particle physics, structures of organic molecules, spacecraft control, and other advanced areas are well-handled by computers as long as the problems can be formulated. But when a problem is too elusive to be formulated explicitly, other approaches are necessary [Chester, 93].

A brief overview of some of the areas in which neural networks are currently being applied indicates their widespread use. The examples range from commercial successes to areas of active research that show promise for the future.

There are many applications of neural networks in the general area of signal processing. One of the first commercial applications is to suppress noise on a telephone line. The neural net used for this purpose is a form of ADALINE. ADALINE is discussed briefly in the next chapter. The need for adaptive echo cancelers has become more essential with the development of transcontinental satellite links for long-distance telephone circuits. The switching involved in conventional echo suppression is very disruptive with path delays of considerable length. Even in the case of wire-based telephone transmission, the repeater amplifiers introduce echoes in the signal [Aggarwal, 99].

The adaptive noise cancellation idea is quite simple. At the end of a long-distance line, the incoming signal is applied to both the telephone system component, the hybrid, and the adaptive filter (the ADALINE type of neural net). The difference between the output of the hybrid and the output of the ADALINE is the error. The error is used to adjust the weights on the ADALINE. The ADALINE is trained to remove the noise, echo, from the hybrid's output signal [Aggarwal, 99].

An example of a robotic application can be seen in robot arm control software. The problem of controlling robotic arm is not an easy task when carried out by conventional software. The complexity of the problem is evident from the fact that the placement of the robot's "hand" at a

particular location is not a single-solution problem. Various combinations of joint angles can get the hand to the same location. To train a neural network to control a robotic arm is conceptually simple. The user simply inputs an end state (the location of an object to be grasped) and any set of joint angles that will put the arm into that end state.

For an arm having n degrees of freedom, the user presents the network with $n + 1$ examples of this kind. From then on, the arm will generate the trajectory to take itself through any sequence of endpoints the user specifies. In other words, from the examples it has been given, it maps the geometry of the space [Fausett, 94].

Many interesting problems fall into the general area of pattern recognition. One specific area in which many neural network applications have been developed is the automatic recognition of handwritten characters. The large variation in sizes, positions, and styles of writing makes this a difficult problem for traditional techniques [Freeman, 91]. It is a good example, however, of the type of information processing that humans can perform relatively easily.

General-purpose multi-layer neural networks, such as the backpropagation net described in chapter 6, have been used for recognizing handwritten zip codes. Even when an application is based on a standard training algorithm, it is quite common to customize the architecture to improve the performance of the application. An alternative approach to the problem of recognizing handwritten characters is the "neocognitron" approach described briefly in the next chapter.

One of many examples of the application of neural networks to medicine was developed by Anderson in the mid 80s. It has been called the "Instant Physician." The idea behind this application is to train an auto associative memory neural network to store a large number of medical records, each of which includes information on symptoms, diagnosis, and treatment for a particular case. After training, the net can be presented with input consisting of a set of symptoms; it will then find the full stored pattern that represents the "best" diagnosis and treatment.

The net performs surprisingly well, given its simple structure. When a particular set of symptoms occurs frequently in the training set, together with a unique diagnosis and treatment, the net will usually give the same diagnosis and treatment. In cases where there are ambiguities in the training data, the net will give the most common diagnosis and treatment. In novel situations, the net will prescribe a treatment corresponding to the symptom(s) it has seen before, regardless of the other symptoms that are present [Freeman, 91].

Learning to read English text aloud is a difficult task, because the correct phonetic pronunciation of a letter depends on the context in which the letter appears. A traditional approach to the problem would typically involve constructing a set of rules for the standard pronunciation of various groups of letters, together with a look-up table for the exceptions.

One of the most widely known examples of a neural network approach to the problem of speech production is NETalk. Instead of rules and look-up tables, NETalk only require a set of examples of the written input together with the correct pronunciation of it [Freeman, 91].

Progress is being made in the area of speaker-independent recognition of speech. A number of useful systems now have a limited vocabulary or grammar or require retraining for different speakers. Several types of neural networks have been used for speech recognition, including multi-layer networks. One net that is of particular interest, both because of its level of development toward a practical system and because of its design, was developed by Kohonen using the Self-Organizing Map, discussed in chapter 4. He calls his net a "phonetic type-writer" [Ritter, 92]. The output units for a Self-Organizing Map are arranged in a two-dimensional array. The input to the net is based on short segments of the speech waveform. As the net groups similar inputs, the clusters that are formed are positioned so that different examples of the same phoneme occur on output units that are close together in the output array.

After the speech input signals are mapped to the appropriate clusters, the output units can be connected to the appropriate typewriter key to construct the phonetic typewriter [Ritter, 92].

Neural networks are being applied in a number of business settings. The mortgage assessment work by Nestor, Inc. is only one of many examples. Although it may be thought that the rules which form the basis for mortgage funding are well-understood, it is difficult to specify completely the process by which experts make decisions in some cases. The basic idea behind the neural network approach to mortgage risk assessment is to use past experience to train the net to provide more consistent and reliable evaluation of mortgage applications. Using data from several experienced mortgage evaluators, neural networks were trained to screen mortgage applicants. The training input includes information on the applicant's years of employment, number of

dependents, current income, as well as features related to the mortgage itself. The target output from the net is an "accept" or "reject" response. The neural networks achieved a high level of agreement with the human experts [Ritter, 92].

1.2 Objectives and Scope of the Report

The objective behind this work is to simulate some of the widely known neural networks with different architectures for solving real-world problems. Three programs were written to simulate a Boltzmann Machine Network, a Kohonen's Self-Organizing Map, and an Adaptive Resonance Theory Network. The programs were developed with Microsoft Visual Basic 5.0 Professional Edition, which operates under Windows95/98 environment. They have graphical user interface and have the full look and feel of a Windows environment. Each program is made up of a main form and other auxiliary forms, depending on the particular program needs.

The report falls into six chapters and an appendix. Chapter 2 discusses the Boltzmann Machine for solving the Traveling Salesperson Problem. Chapter 3 discusses Kohonen's Self-Organizing Maps for the Character Recognition Problem. The same problem is also handled in chapter 4 but using Adaptive Resonance Theory. Backpropagation neural networks are discussed in chapter 5. The Appendix, at the end of the report, contains the high-level code design of the implemented algorithms.

Chapter 2

Neural Networks

2.1. Neural-Networks Concepts

This section highlights the features of biological and artificial neural networks. It also sheds light on the common features found in both kinds of networks.

2.1.1. Biological Neural Networks

In addition to being the original inspiration for artificial networks, biological neural systems suggest features that have distinct computational advantages. There is a close analogy between the structure of a biological neuron (i.e., brain or nerve cell) and the processing element (or artificial neuron). A biological neuron has three types of components that are of particular interest in understanding an artificial neuron: Dendrites, Soma, and Axon. Figure 2.1 shows a simplified representation of a neuron. The Dendrites receive signals from other neurons. Each neuron can receive up to 10,000 signals from other neurons. The signals are electric impulses that are transmitted across a synaptic gap by means of a chemical process. The action of the chemical transmitter modifies the incoming signal (typically, by adjusting the frequency of the signals that are received) in a manner similar to the action of the weights in an artificial neural network [Ritter, 92].

The Soma, or cell body, sums the incoming signals. When sufficient input is received, the cell fires; that is, it transmits a signal over its axon to other cells. It is often supposed that a cell either fires or does not at any instant of time. Therefore, the transmitted signals can be treated as binary. The transmission of the signal from a particular neuron is accomplished by an action potential resulting from differential concentrations of ions on either side of the neuron's axon sheath (the brain's "white matter"). The ions directly involved are potassium, sodium, and chloride [Veelenturf, 95].

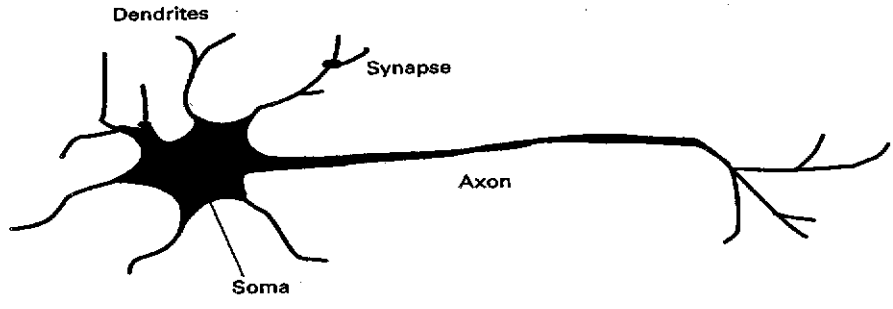


Figure 2.1: A Biological Neuron

2.1.2. Artificial Neural Networks

An artificial neural network is an information-processing system that has certain performance characteristics in common with biological neural networks. A neural net consists of a large number of simple processing elements called neurons, units, cells, or nodes. Each neuron is connected to other neurons by means of directed communication links, each with an associated weight. The weights represent information being used by the

net to solve a problem. Neural networks can be applied to a wide variety of problems, such as storing and recalling data or patterns, classifying patterns, performing general mappings from input patterns to output patterns, grouping similar patterns, or finding solutions to constrained optimization problems.

Each neuron has an internal state, called its activation or activity level, which is a function of the inputs it has received. Typically, a neuron sends its activation as a signal to several other neutrons. It is important to note that a neuron can send only one signal at a time, although that signal is broadcast to several other neurons [Chester, 93].

The neuron Y, illustrated in Figure 2.2, receives inputs from neurons X1, X2, X3, and X4. The activations (output signals) of these neurons are x1, x2, x3, and x4 respectively. The weights on the connections from X1, X2, X3, and X4 to neuron Y are w1, w2, w3, and w4, respectively. The net input, y-in, to neuron Y is the sum of the weighted signals from neurons X1, X2, X3 and X4, i.e., Y-in = w1x1 + w2x2 + w3x3 + w4x4

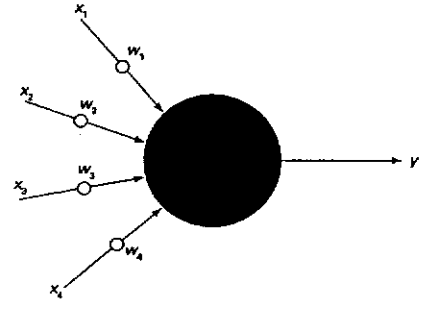


Figure 2.2: An Artificial Neuron

Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

1. Information processing occurs at many simple elements called neurons.
2. Signals are passed between neurons over connection links.
3. Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
4. Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal [Freeman, 91].

Several key features of the processing elements of artificial neural networks are suggested by the properties of biological neurons:

1. The processing element receives many signals.
2. Signals may be modified by a weight at the receiving synapses. The processing element sums the weighted inputs.
3. Under appropriate circumstances (sufficient input), the neuron transmits a single output.
4. The output from a particular neuron may go to many other neurons (the axon branches).
5. Information processing is local (although other means of transmission, such as the action of hormones, may suggest means of overall process control).
6. Memory is distributed:
   a. Long-term memory resides in the neurons' synapses or weights.
   b. Short-term memory corresponds to the signals sent by the neurons. A synapse's strength may be modified by experience.
7. Neurotransmitters for synapses may be excitatory or inhibitory [Freeman, 91].

Another important characteristic that artificial neural networks share with biological neural systems is fault tolerance. Biological neural systems are fault tolerant in two respects. First, humans are able to recognize many input signals that are somewhat different from any signal they have seen before. An example of this is our ability to recognize a person in a picture we have not seen before. Second, humans are able to tolerate damage to the neural system itself. Humans are born with as many as 100 billion neurons. Most of these are in the brain, and most are not replaced when they die. In spite of the continuous loss of neurons, we continue to learn [Chester, 93].

A neural network is characterized by (1) its pattern of connections between the neurons (called its architecture), (2) its method of determining the weights on the connections (called its training, or learning algorithm), and (3) its activation function [Veelenturf, 95].

2.2. Evolution of Neural Networks

This section presents a brief summary of the history of neural networks, in terms of the development of architectures and algorithms that are widely used today. Researchers such as McCulloch and Pitts started working in this field in the 40s and researchers continue to develop new architectures and training algorithms. Future progress in this field is certain as more research funds are being allocated for development in this field.

2.2.1. The Beginning of Neural Networks (40s)

The first ideas about neural networks date back to the decade that witnessed the birth of computers. Significant progress has been made in the early years. The credit for this progress goes back to many researchers such as McCulloch, Pitts, and Hebb.

The first artificial neural networks were designed in the 1940s by McCulloch and Walter Pitts. They recognized that combining many simple neurons into neural systems was the source of increased computational power [Ritter, 92]. The weights on McCulloch-Pitts neuron are set so that the neuron performs a particular simple logic function, with different neurons performing different functions. The neurons could be arranged to produce any output that can be represented as a combination of logic functions.

The idea of a threshold such that if the net input to a neuron is greater than the threshold then the unit fires is one feature of McCulloch-Pitts neuron that is used in many artificial neurons today. McCulloch and Pitts' subsequent work addressed issues that are still important research areas today. These include translation and rotation invariant pattern recognition [Veelenturf, 95].

Donald Hebb, a psychologist at McGill University, designed the first learning law for artificial neural networks. His premise was that if two neurons were active simultaneously, then the strength of the connection should be increased. Kohonen and Andersen subsequently made refinements to this concept.

2.2.2. The First Golden Age of Neural Networks (50s and 60s)

The introduction of the perceptrons and ADALINE in the 50s and 60s added new dimensions to artificial neural networks and led researchers to name that era the golden age of artificial neural networks.

Together with other researchers, Frank Rosenblatt introduced and developed a large class of artificial neural networks called perceptrons. The most typical perceptron consisted of an input layer (the retina) connected by paths with fixed weights to associator neurons; the weights on the connection paths were adjustable. The perceptron learning rule uses an iterative weight adjustment that is more powerful than the Hebb rule. Perceptron learning can be proved to converge to the correct weights if there are weights that will solve the problem at hand. Like the neurons developed by McCulloch and Pitts and by Hebb, perceptrons use a threshold output function [Chester, 93].

Bernard Widrow and Marcian Hoff developed a learning rule, the Delta Rule or the least mean squares, that is closely related to the perceptron learning rule. The perceptron rule adjusts the connection weights to a unit whenever the response of the unit is incorrect. The Delta Rule adjusts the weights to reduce the difference between the net input to the output unit and the desired output. This results in the smallest mean squared error. The Widrow learning rule for a single-layer network is the basis of the backpropagation rule for multi-layer networks [Veelenturf, 95]. The name ADALINE, ADAptive LINEar neurons, was often given to networks developed by Widrow and his students.

2.2.3. Continued Progress (70s)

Many of the current leaders in the field began to publish their work during the 70s. Kohonen, Andersen, and Grossberg were among the most famous researchers during the 70s.

The early work of Teuvo Kohonen, of Helsinki University of Technology, dealt with associative memory neural networks [Freeman, 91]. His most recent work has been the development of Self-Organizing Maps that use the topological structure for the cluster units. These networks have been applied to speech recognition, the solution of the Traveling Sales Man Problem, and musical composition.

James Andersen works with associative memory networks. He developed these ideas into his "Brain-State-in-a-Box", which truncates the linear output of earlier models to prevent the output from becoming too large as the net iterates to find a stable solution [Fausett, 94]. Among the areas of application for these networks are medical diagnosis and learning multiplication tables.

Together with Stephen Grossberg, Gail Carpenter developed a theory of Self-Organizing neural networks called Adaptive Resonance Theory (ART). Details about this theory will follow later in the report.

2.2.4. New Techniques

New techniques were introduced during the 80s. Multi-layer networks training algorithms were used. Backpropagation networks were the distinctive features of this era.

The 1970s are sometimes referred to as the "quiet years" due to the failure of single-layer perceptrons to be able to solve some problems and the lack of a general method of training a multi-layer net. A method for propagating information about errors at the output units back to the hidden units had been discovered in this decade. Neural networks that employed this method were called backpropagation neural networks.

Another key player in the domain of neural networks is Nobel Prize winner (in physics) John Hopfield, of the California Institute of Technology. Together with David Tank, a researcher at AT&T, Hopfield has developed a number of neural networks based on fixed weights and adaptive activations. These networks can serve as associative memory networks and can be used to solve constraint satisfaction problems such as the Traveling Salesperson Problem [Chester, 93].

Kunihiko Fukushima and his colleagues at NHK Laboratories in Tokyo have developed a series of specialized neural networks for character recognition [Chester, 93]. One example of such a net is called a neocognitron. An earlier Self-Organizing network, called the cognitron, failed to recognize position or rotation-distorted characters. This deficiency was corrected in the neocognitron.

A number of researchers have been involved in the development of non-deterministic neural network in which weights or activations are changed on the basis of probability density function. These networks incorporate classical ideas such as simulated annealing and Bayesian Decision Theory.

A major reason for the interest in neural networks is improved computational capabilities. Optical neural networks and VLSI implementations are being developed. Leon Cooper introduced one of the first multi-layer networks, the "reduced coulomb energy network". Robert Hecht-Nielsen and Todd Gutschow developed several digital neurocomputers at TRW Inc. during 1983-1985 [Veelenturf, 95].

2.3. Architectures of Artificial Neural Networks

Often, it is convenient to visualize neurons as arranged in layers. Typically, neurons in the same layer behave in the same manner. Key factors in determining the behavior of a neuron are its activation function and the pattern of weighted connections over which it sends and receives signals. Within each layer, neurons usually have the same activation function and the same pattern of connections to other neurons. In many neural networks, the neurons within a layer are either fully interconnected or not interconnected at all. If any neuron in a layer is connected to a neuron in another layer then each neuron in that layer is connected to every neuron in the other layer.

The arrangement of neurons into layer and the connection patterns within and between layers is called the net architecture [Freeman, 91]. Many neural networks have an input layer in which the activation of each unit is equal to an external input signal. Neural networks are often classified as single-layer or multi layer. In determining the number of layers, the input units are not counted as a layer, because they perform no computation. Equivalently, the number of layers in the net can be defined to be the number of layers of weighted interconnected links between the blocks of neurons. This view is motivated by the fact that the weights in a net contain extremely important information. Two classes of neural networks are the feed forward and recurrent networks. These classes could be either single-layer or multi-layer networks.

2.3.1. Single-layer Networks

A single-layer net has one layer of connection weights. Often the units can be distinguished as input units, which receive signals from the outside world, and output units from which the response of the net can be read. In the single-layer net the input units are fully connected to output units but are not connected to other input units and the output units are not connected to other output units. For pattern classification each output unit corresponds to a particular category to which an input vector may or may not belong. For pattern association the same architecture can be used, but now the overall pattern of output signals gives the response pattern associated with the input signal that caused it to be produced. The same network architecture can be used for different problems, depending on the interpretation of the response of the net. Figure 2.3 shows one possible architecture of a single-layer network.

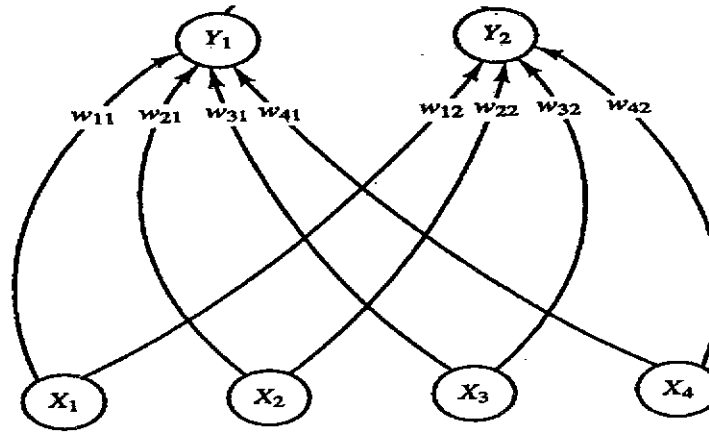


Figure 2.3: A Single-layer Network

2.3.2. Multi-layer Network

A multi-layer network is a net with one or more layers of nodes (hidden units) between the input units and the output units. Typically, there is a layer of weights between two adjacent levels of units. Multi-layer networks can solve more complicated problems than can single-layer networks. Figure 2.4 shows an example of a multi-layer network.

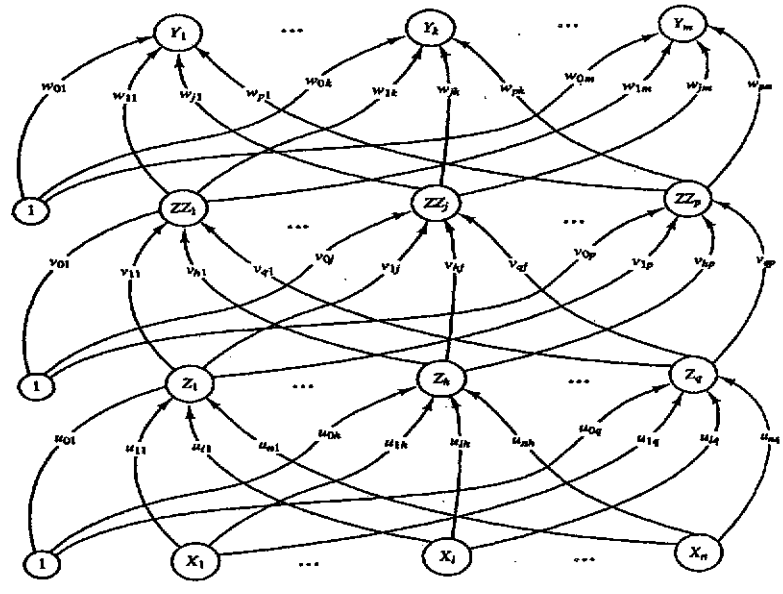


Figure 2.4: A Multi-layer Network

2.4. Training Neural Networks

The method of setting the values of weights is an important characteristic of different neural networks. Two types of training are used, supervised and unsupervised. Some networks do not need training since their weights do not change. Many of the tasks that neural networks can be trained to perform fall into the areas of mapping, clustering, and constrained optimization. The following two subsections give details about each of the training methods.

2.4.1. Supervised Training

In most of the typical neural networks' setting, training is performed by presenting a sequence of training vectors or patterns, each with an associated target output vector. The weights are then adjusted according to a learning algorithm. This process is known as supervised training [Freeman, 91].

Some of the simplest neural networks are designed to perform pattern classification, (i.e., to classify an input vector as either belonging or not belonging to a given category). In this type of neural networks, the output is binary. These networks are trained using a supervised algorithm. For more difficult classification problems, a multi-layer network such as that trained by backpropagation algorithm is more adequate. Pattern association is another special form of a mapping problem, one in which the desired output is not just a "yes" or "no", but rather a pattern. A neural network that is trained to associate a set of input vectors with a corresponding set of output vector is called an associative memory.

After training, an associative memory can recall a stored pattern when it is given an output vector that is sufficiently similar to a vector it has learned [Ritter, 92].

2.4.2. Unsupervised Training

Self-Organizing neural networks group similar input vectors together without the use of training data to specify what a typical member of each group looks like or to which group each vector belong. A sequence of input vectors is provided, but no target vectors are specified. The network modifies the weights so that the most similar input vectors are assigned to the same output or cluster unit. The network will produce an exemplar or representative vector for each cluster formed.

Chapter 3

Boltzmann Machine for the Travelling Salesperson Problem

There are many interesting variants on neural network paradigms, involving aspects such as probabilistic models and mixed architectures of various kinds that take selected elements from two or more existing paradigms. This chapter describes the use of neural networks coupled with simulated annealing paradigm. The network is used for solving the Traveling Salesperson Problem, which is a well-known constrained optimization problem.

3.1 Description of the Travelling Salesperson Problem

In the classic constrained optimization problem known as the Traveling Salesperson Problem (TSP), the salesperson is required to visit each of a given set of cities once and only once, returning to the starting city at the end of his tour. The tour of minimum distance is desired. The TSP becomes computationally vast as the number of cities increases. For n cities, there are $n!$ possible arrangements. But because the solution is a closed loop, it does not matter which city is chosen as a starting point (this divides the number of independent solutions by n), nor does the direction of traversing the loop matter (which divides the number of solutions by two). Thus the number of independent solutions becomes $n!/2n = (n\text{-}1)!/2$ [Chester, 93].

3.2 Neural-Networks' Approach to Constrained Optimization

In neural networks that are designed for constrained optimization, each unit represents a hypothesis, with the unit "on" if the hypothesis is true, or "off" if the hypothesis is false. The weights are fixed to represent both the constraints of the problem and the function to be optimized. The solution of the problem corresponds to the minimum of an energy function or the maximum of a consensus function for the network. The activity level of each unit is adjusted so that the network will find the desired maximum or minimum value.

Neural networks have several potential advantages over traditional techniques for certain types of optimization problems. They can find near optimal solutions quickly for large problems. They can also handle situations in which some constraints are weak (desirable but not absolutely required). For example, in the TSP, it is physically impossible to visit two cities simultaneously, but it may be desirable to visit each city only once. The difference in these types of constraints could be reflected by making the penalty for having two units in the same column "on" simultaneously larger than the penalty for having two units in the same row "on" simultaneously. If it is more important to visit some cities than others, these cities can be given larger self-connection weights. The idea of obtaining a solution that could be tailored according to our needs makes neural network solutions favorable ones.

3.3 Boltzmann Machine Network

For n cities, n^2 units arranged in a square array are used. A valid tour is represented by exactly one unit being "on" in each row and in each column. Two units being "on" in a row indicates that the corresponding city was visited twice; two units being "on" in a column shows that the salesperson was in two cities at the same time. The units in each row are fully interconnected; similarly, the units in each column are fully interconnected. The weights are set so that units within the same row or within the same column will tend not to be "on" at the same time. In addition, there are connections between units in adjacent columns and between units in the first and last columns, corresponding to distances between cities.

The bi-directional nature of the connection is often represented as $w_{ij} = w_{ji}$. A unit may also have a self-connection w_{ii}. The state x_i of unit X_i is either 1 ("on") or 0 ("off"). The objective of the neural net is to maximize the consensus function

$$C = \Sigma_i[\Sigma_{j<=i}\, w_{ij}\, x_i x_j].$$

The net finds this maximum (or at least a local maximum) by letting each unit attempt to change its state (from "on" to "off" or vice versa). The attempts may be made either sequentially or in parallel. The change in consensus if unit X_i were to change its state from 1 to 0 or from 0 to 1 is

$$\Delta C(i) = (1 - 2xi)*(w_{ii} + \Sigma_{i\neq j}\, w_{ij}x_i)$$

where x_i is the current state of unit X_i. The coefficient $(1 - 2x_i)$ will be + 1 if unit Xi is currently "off" and - 1 if unit X_i is currently "on" [Fausett, 94].

If unit X_i were to change its activation, the resulting change in consensus can be computed from information that is local to unit X_i (i.e., from weights on connections and activations of units to which unit X_i is connected, with $w_{ij} = 0$ if unit X_j is not connected to unit X_i).

However, unit X_i does not necessarily change its state, even if doing so would increase the consensus of the net. The probability of the net accepting a change in state for unit X_i is

$$A(i,T)= 1 / (1+e^{-\Delta C(i)/T})$$

The control parameter T (called the temperature) is gradually reduced as the net searches for a maximal consensus. Lower values of T make it more likely that the net will accept a change of state that increases its consensus and less likely that it will accept a change that reduces its consensus. The use of a probabilistic update procedure for the activations, with the control parameter decreasing as the net searches for the optimal solution to the problem represented by its weights, reduces the chances of the net getting stuck in a local maximum [Fausett, 94].

This process of gradually reducing the temperature is called simulated annealing. It is analogous to the physical annealing process used to produce a strong metal (with a regular crystalline structure). During annealing, a molten metal is cooled gradually in order to avoid imperfections in the crystalline structure of the metal due to freezing [Freeman, 91].

3.3.1 Architecture

The Boltzmann machine network consists of units arranged in a two-dimensional array. The units within each row are fully interconnected. Similarly, the units within each column are also fully interconnected. The weights on each of the connections is $-p$ (where $p > 0$). In addition, each unit has a self-connection, with weight $b > 0$. The connections shown in the figure will form a portion of the Boltzmann machine to solve the Traveling Salesperson Problem.

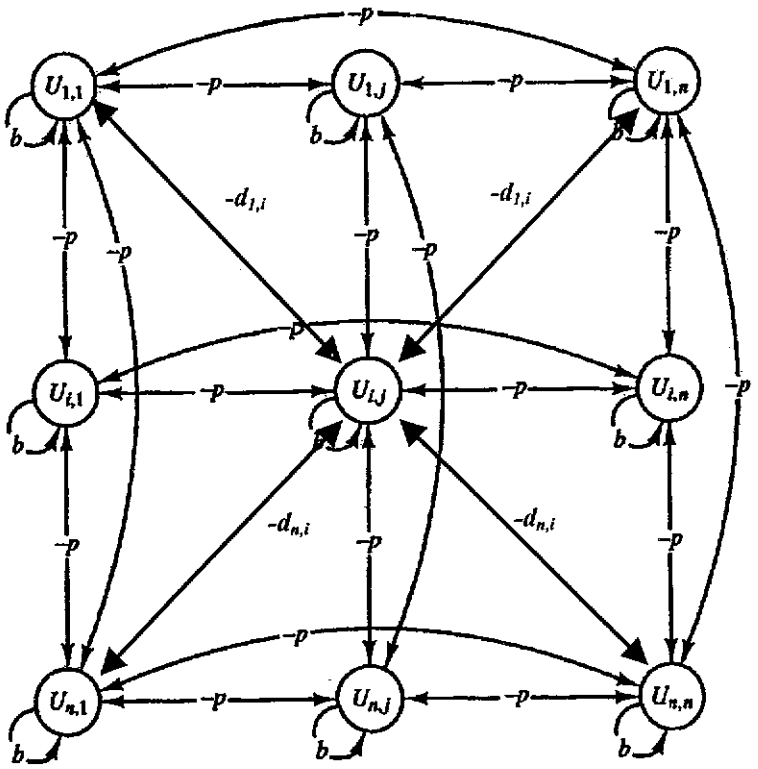


Figure 3.1: Boltzmann Neural Network for Solving TSP

3.3.2 Weight Setting

The weights for a Boltzmann machine are fixed so that the net will tend to make state transitions toward a maximum of the consensus function defined earlier. If the net is to have exactly one unit "on" in each row and in each column, the values of the weights p and b must be chosen so that improving the configuration corresponds to increasing the consensus.

Each unit is connected to every other unit in the same row with weight $-p$ ($p > 0$); similarly, each unit is connected to every other unit in the same column with weight $-p$. These weights are penalties for violating the condition that at most one unit be "on" in each row and each column. In addition, each unit has a self-connection, of weight $b > 0$. The self-connection weight is an incentive to encourage a unit to turn "on" if it can do so without causing more than one unit to be on in a row or column.

If $p > b$, the net will function as desired. The correct choice of weights to ensure that the net functions as desired can be deduced by considering the effect on the consensus of the net in the following two situations.

If unit U_{ij} is "off ", i.e., $u_{ij} = 0$, and none of the units connected to U_{ij} is on, then changing the status of U_{ij} to "on" will increase the consensus of the net by the amount b. This is a desirable change; since it corresponds to an increase in consensus. The network will be more likely to accept it than to reject it. On the other hand, if one of the units in row i or in column j (say, $U_{i,j+1}$ is already "on"), attempting to turn unit U_{ij} "on" would result in a change

of consensus by the amount $b - p$. Thus, for $b - p < 0$,i.e., $p > b$, the effect would be to decrease the consensus. The network will tend to reject this unfavorable change [Fausett, 94].

Bonus and penalty connections, with $p > b$, will be used in the network for the TSP to represent the constraints for a valid tour.

The desired neural network will be constructed in two steps. First, a neural network will be formed for which the maximum consensus occurs whenever the constraints of the problem are satisfied (i.e., when exactly one unit is "on" in each row and in each column). Second, weighted connections will be added to represent the distances between the cities. In order to treat the problem as a maximum consensus problem, the weights representing distances will be negative [Fausett, 94].

To complete the formulation of a Boltzmann neural net for the TSP, weighted connections representing distances must be included. In addition to the weights described before (which represent the constraints), a typical unit U_{ij} is connected to the units $U_{k,j-1}$ and $U_{k,j+1}$ (for all $k \neq i$) by weights that represent the distances between city *i* and city *k*. Units in the last column are connected to units in the first column by connections representing the appropriate distances. However, units in a particular column are not connected to units in columns other than those immediately adjacent to the said column.

3.3.3 Nomenclature

The following is a list of names used in the algorithm:

N: Number of cities in the tour

i: Index designating a city

j: Index designating position in tour, mod n; so if $j = n+1$ then j gets 1

U_{ij}: Unit representing the hypothesis that the i^{th} city is visited at the j^{th} step of the tour

$u_{i,j}$: Activation of unit $U_{i,j}$; $u_{i,j} = 1$ if the hypothesis is true, 0 if it is false

$d_{i,k}$: Distance between city i and city k, $k \neq i$.

d: Maximum distance between any two cities

3.3.4 Algorithm

The implemented algorithm is taken from [Fausett,94]. Its steps are as follows:

Initialize weights to represent the constraints of the problem.

Initialize the control parameter (temperature) T.

Initialize activations of units (random binary values).

While stopping condition is false, **do**

Do n^2 times. (This constitutes an epoch.)

Begin

Choose integers I and J at random between 1 and n.

(Unit U_{IJ} is the current candidate to change its state.)

Compute the change in consensus that would result:

$$\Delta C = [1 - 2u_{I,J}]*[w(I,J;I,J) + \Sigma_{ij \neq IJ} \Sigma\, w(i,j;I,J)u_{i,j}]$$

Compute the probability of acceptance of the change:

$A(T)= 1/(1+e^{(-\Delta C/T)})$

Determine whether or not to accept the change:

Let R be a random number between 0 and 1

If R<A, accept the change:

$u_{I,J} = 1 - u_{I,J}$ (This changes the state of unit $U_{I,J}$)

Else Reject the proposed Change

End {Do n^2 times}

Reduce the control parameter:

T(new)= 0.95T(old)

Test Stopping Condition

3.4 Experimental Results and Conclusions

The program that implements the above algorithm runs under Windows95/98 and has the full feel and look of a Windows environment. The user is required to deploy the cities either manually or allow the program to deploy them randomly. Initial and final temperatures should also be specified for the program. The program produces four types of output:

- A graphical and animated graph showing the order of cities the salesperson should take;
- Tabulated results of each city name and its corresponding turn in the visit;
- The distance traveled at each instance of time; and
- The time taken by the algorithm to finish execution

Table 3.1 below shows the results obtained from running the above algorithm on 10, 20, 30, 40, 50, and 90 cites. In each scenario, cities were deployed randomly in 10 different initial configurations and only the minimal path obtained is documented in the table below. The cooling schedule is such that $T(\text{new}) = T(\text{old}) * 0.9$.

Number Of Cities	**Temperature**		**Execution Time in Minutes**	**Minimum Distance Covered on the Unit Square**	**Number of Failures Before Convergence**
	Initial	**Final**			
10	20	5	0.066	3.91	0
20	20	5	0.416	11.01	0
30	20	5	2.9	13.19	0
40	20	5	41	19.21	6
50	20	5	63	26.33	6
90	20	5	306.2	56.17	11

Table 3.1: Effect of Increasing the Number of Cities on the Execution Time

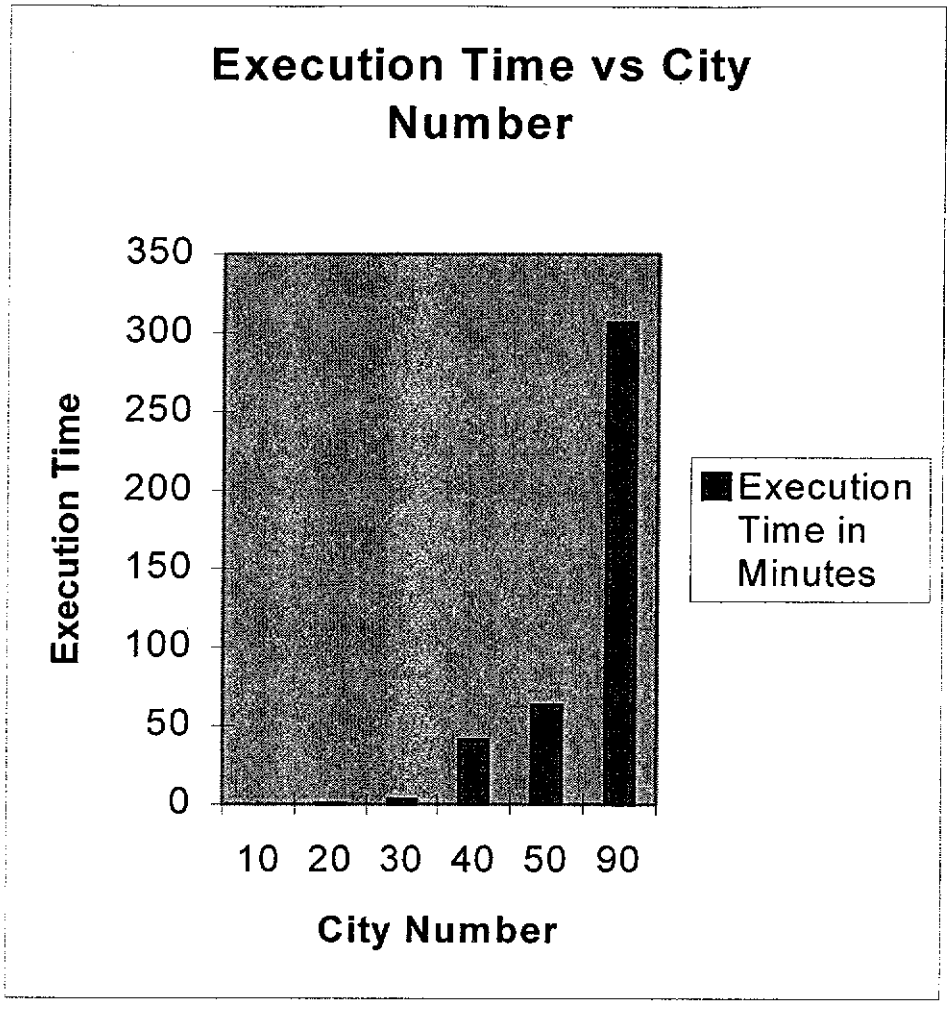


Figure 3.2: Execution Time vs City Number

Figure 3.2 shows the relationship between the number of cities and the execution time of the algorithm.

In Table 3.2 the same city configuration is generated each time for testing the effect of choosing initial values for the temperature parameter on the performance of the network. This is done by calling the Visual Basic Function *Rnd(number)* with a negative *number* just before using the *Randomize* function.

Number Of Cities	Temperature		Execution Time in Minutes	Distance Covered	Number of Failures Before Convergence
	Initial	Final			
30	20	5	2.1	12.45	0
30	21	5	2.2	15.66	0
30	22	5	2.2	15.66	0
30	23	5	12.0	14.14	6
30	24	5	2	14.90	1
30	25	5	9.5	14.29	3
30	30	5	5.00	15.47	2
30	35	5	7.1	14.17	2

Table 3.2: Effect of Changing the Value for the Initial Temperature Value

Yet another set of parameters worth considering are the constraint parameters p and b. Choosing large bonus and penalty weights (i.e., $b=60$ and $p=70$) with initial temperature $T=20$ and a cooling schedule such that T(new)= 0.9 T(old) produced valid tours for 30 different initial 30-city problem configuration. Typically, valid tours were generated in 10 or fewer epochs. An epoch consists of each unit attempting to change its state. Choosing smaller values for b and p (b=30, p=35), again 30 valid tours were found. However, none of the tours were was shorter than their corresponding ones in the previous experiment. Using still smaller values for p and b, the network failed to find valid tours.

New adaptive neural structures such as the Guilty Net presented by [Burkee, 92], utilizes a straight forward competitive learning algorithm with fixed neighbors to automatically generate valid, short tours in only a few hundred iterations. Results shown in [Burkee, 92] indicate shorter tours and shorter running time for the proposed Guilty Net algorithm when compared with other approaches. Table 3.3 below is taken as is from [Burkee, 92].

Method	**Number of Cities**			
	10	**30**	**50**	**90**
Theoretically Optimal	2.43	4.21	5.43	7.68
Simulated Annealing	2.74	4.75	6.13	8.40
Space Filling Curve	3.14	5.45	7.03	9.56
Hopfield	2.92	5.12	6.64	-
Elastic Net	2.45	4.51	5.58	8.23
Guilty Net	2.78	4.81	6.21	8.81

Table 3.3: Comparison between Different Optimization Methods for the TSP Problem

Four main conclusions may be drawn based on the above discussions and results:

1. As the number of cities to be visited increases linearly, the running time of the algorithm increases at much higher rate. This fact is justifiable since the algorithm has a polynomial complexity $O(n) = n^4$, where n is the number of cities to be visited.

2. Relatively lower initial temperatures tend to allow the network to converge to a valid solution faster. The network fails to converge at least once for initial temperature greater than 22.
3. Penalty and bonus weights (p and b) do not have direct effect on the performance of the network as long as they are greater than a certain problem-specific threshold. As explained previously, p must be greater than b so than the network functions as desired.
4. The Boltzmann Machine for solving the TSP is not the best neural network approach for such a solution. Although it offers some kind of flexibility, it proved to be computationally demanding and less efficient when compared with other competition based neural network approaches such as the Guilty Net.

Chapter 4

Kohonen's Self-Organizing Maps for Character Recognition

This chapter discusses one type of competition-based neural networks, Kohonen's Self-Organizing Maps. In this type of unsupervised training networks, clusters compete among one another and the winning cluster has the chance to update its weight vector. An algorithm to solve the Character Recognition Problem is discussed and implemented. Neural networks developed by Kohonen have been applied to an interesting variety of problems. One recent development of his is a neural network approach to computer-generated music. Angeniol, Vaubois, and Le Texier have also applied Kohonen Self-Organizing Maps to the solution of the Traveling Salesperson Problem.

4.1 Competition-Based Networks

Some neural networks that are used for pattern classification may respond by associating a certain input pattern with more than one output pattern. In circumstances in which only one of several neurons should respond, additional structure could be included in the network so that the network is forced to make a decision as to which one unit will respond. The mechanism by which this is achieved is called competition. Competition ensures that a pattern is associated with only one exemplar (or representative). The most extreme form of competition among a group of neurons is called the "Winner Take All." As the name suggests, only one neuron in the competing group will have a non-zero output signal when the competition is completed [Freeman, 91].

Neural network learning is not restricted to supervised learning, where training pairs are provided, as with the pattern classification and pattern association problems. A second major type of learning for neural networks is unsupervised learning, in which the network seeks to find patterns in the input data. Self-Organizing Maps, developed by Kohonen, groups the input data into clusters, a common use for unsupervised learning. Adaptive Resonance Theory networks, the subject of the next chapter, are also clustering networks. The term "pattern classification" is applied to these networks too, but the use of it is reserved for situations in which the learning is supervised and target classifications are provided during training.

In a clustering network, there are as many input units as an input vector has components. Since each output unit represents a cluster, the number of output units will limit the number of clusters that can be formed.

The weight vector for an output unit in a clustering network serves as a representative, or exemplar, vector for the input patterns which the network has placed on that cluster. During training, the network determines the output unit that is the best match for the current input vector; the weight vector for the winner is then adjusted in accordance with the network's learning algorithm. The training process for the Adaptive Resonance Theory networks discussed in the next chapter involves a somewhat expanded form of this basic idea.

The network discussed in this chapter uses a learning algorithm, known as Kohonen learning. In this form of learning, the units that update their weights do so by forming a new weight vector that is a linear combination of the old weight vector and the current input vector. Typically, the unit whose weight vector was closest to the input vector is allowed to learn. The weight update for output (or cluster) unit *j* is given as:

$$w_j(new) = w_j(old) + \alpha[x - w_j(old)] = \alpha x + (1 - \alpha)w_j(old)$$

where x is the input vector, w_j is the weight vector for unit *j*, and α, the learning rate, decreases as learning proceeds [Freeman, 91].

Two methods of determining the closest weight vector to a pattern vector are commonly used for Self-Organizing networks. Both are based on the assumption that the weight vector for each cluster (output) unit serves as an exemplar for the input vectors that have been assigned to that unit during learning.

The first method of determining the winner uses the squared Euclidean distance between the input vector and the weight vector and chooses the unit whose weight vector has the smallest Euclidean distance from the input vector.

The second method uses the dot product of the input vector and the weight vector. The largest dot product corresponds to the smallest angle between the input and weight vectors if they are both of unit length. The dot product can be interpreted as giving the correlation between the input and weight vectors.

4.2 Kohonen's Self-Organizing Maps

The Self-Organizing neural network described in this section, also called topology-preserving map, assume a topological structure among the cluster units. This property is observed in the brain, but is not found in other artificial neural networks. There are *m* cluster units, arranged in a one or two-dimensional array; the input signals are *n*-tuples. The weight vector for a cluster unit serves as an exemplar of the input patterns associated with that cluster. During the self-organization process, the cluster unit whose weight vector matches the input pattern most closely is chosen as the winner. The winning unit and its neighboring units, in terms of the topology of the cluster, update their weights.

4.2.1 Architecture

The architecture of the Kohonen Self-Organizing Map is shown in Figure 4.1 below. The neighborhoods of radii R = 2, 1 and 0 are shown in Figure 4.2 for a grid. The winning unit is indicated by the symbol "#" and the other units are denoted by "*".

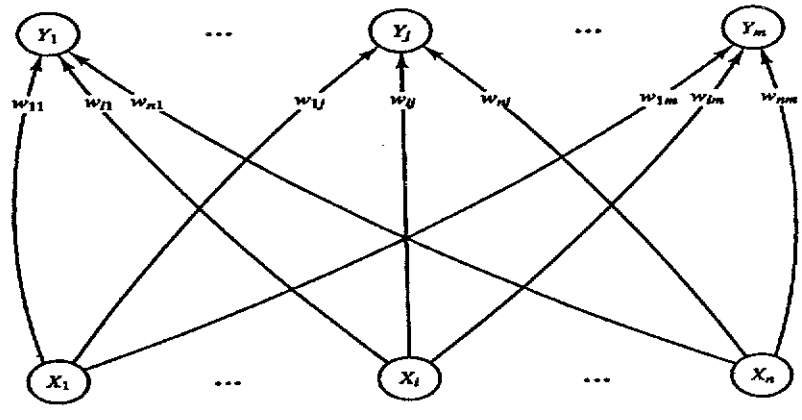


Figure 4.1: Kohonen's Self-Organizing Map

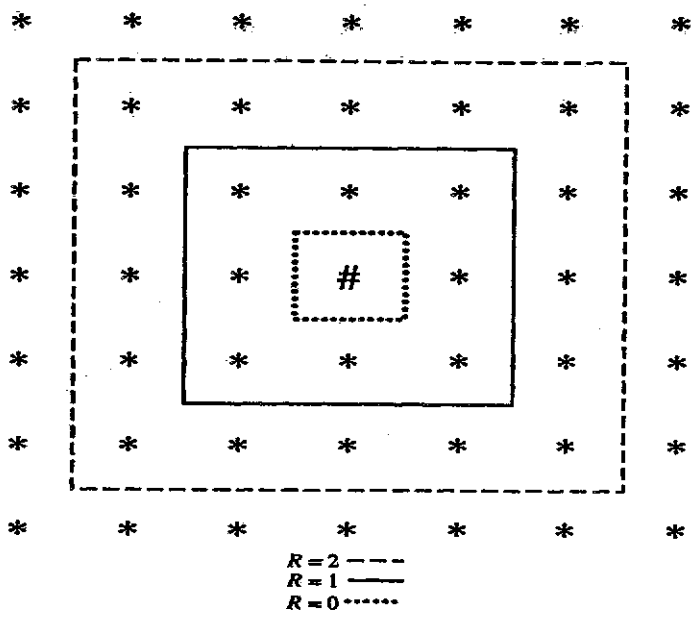


Figure 4.2 Neighborhood in a Rectangular Grid

4.2.2 Algorithm

The implemented algorithm is taken from [Fausett,94]. Its steps are as follows:

Step 0. Initialize weights w_{ij}.

Set topological neighborhood parameters.

Set learning rate parameters.

Step 1. While condition is false, do Steps 2-8.

Step 2. For each input vector x, do Steps 3-5.

Step 3 For each j, compute:

$$D(j)= \Sigma_i(w_{ij} - X_i)^2.$$

Step 4. **Find** index J such that D(J) is a minimum.

Step 5. **For** all units j within a specified neighborhood of J, and for all i:

$w_{ij}(new) = w_{ij}(old) + \alpha[x_i - w_{ij}(old)]$

Step 6. **Update** learning rate.

Step 7. **Reduce** radius of topological neighborhood at specified times.

Step 8. **Test** stopping condition.

4.2.3 Controlling Parameters

Alternative structures are possible for reducing R and α. The learning rate α is a slowly decreasing function of time (or training epochs). Kohonen indicates that a linearly decreasing function is satisfactory for practical computations; a geometric decrease would produce similar results. The radius of the neighborhood around a cluster unit also decreases as the clustering process progresses [Fausett, 94].

The formation of a map occurs in two phases: the initial formation of the correct order and the final convergence. The second phase takes much longer than the first and requires a small value for the learning rate.

4.3 Experimental Results and Conclusions

The above algorithm is used to recognize characters belonging to three different fonts. The fonts used are not real world fonts. They were created in such away that a character in one font has a different shape than the same one in the other two fonts, yet the difference was intended not to be revolutionary so that the network could still give reasonable results. Figure 4.3 shows the letter "A" written in the three different fonts. The network is supposed to cluster similar patterns together. For example, all "A" letters in the three fonts are clustered on unit 24.

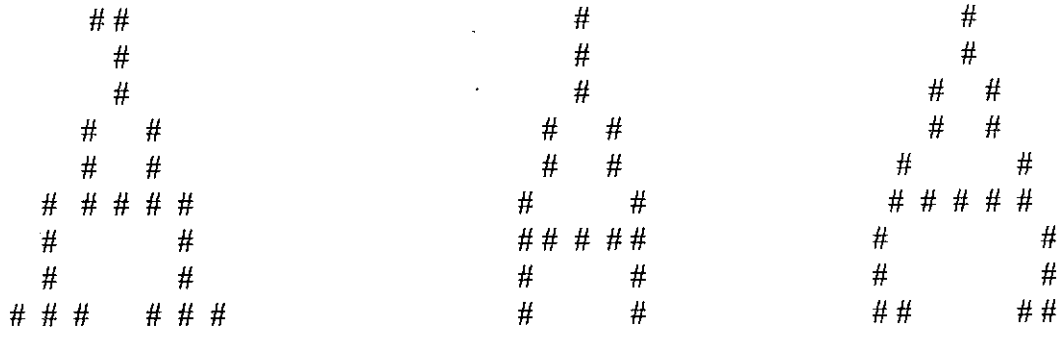


Figure 4.3: Letter "A" as Drawn in the Three Fonts

The program takes as input a bitmap file made up of disconnected uppercase letters. The file is opened using a Windows CommonDialoguebox control into a Picturebox control. The user starts the character recognition process by clicking a command button. The text equivalent for the picture input is written to a Textbox Control on the same form. The program traverses the picture box and identifies each letter individually before presenting it to the network as a binary vector.

For experimental purposes, the algorithm was run a number of times, each time using a different value for the "learning" radius while keeping the same value for the learning rate α. All experiments were run using a 36-unit grid neighborhood architecture for the cluster units. Table 4.1 shows the results of running the algorithm with the radius parameter set to 4. Table 4.2 shows the same thing but with radius set to three. This value proved to be the most suitable for this particular character recognition problem at a given learning rate α.

Radius = 4 **α = 0.8**	**Unit or** **Cluster**	**Patterns**
	1	G1, G2, G3, S1,S2, S3
	3	E2, B2
	5	I1, I3, T1, T2, T3
	8	M1, M2, M3
	9	B1
	10	B3, E1, E3
	12	Z2, Z3
	13	O1, O2 ,O3 ,Q1 , Q2 , Q3
	16	V3
	15	H1, H2, H3
	17	W1, W2, W3, V1
	18	Z1
	20	F1, F2, F3, K1, K3
	21	D1, D3, L1, L2 ,L3
	22	P1, P2, P3, R1, R2, R3
	25	C1, K2, U1, U2, U3
	26	C2, C3, J1, J2 , J3, D2, V2
	28	I2
	29	Y1, Y2, Y3
	30	X1, X2, X3
	31	N3
	33	N2, N3
	36	A1, A2, A3

Table 4.1 Clustering of Characters with Radius Set to Four & α=0.8

Radius = 3 α = 0.8	Unit or Cluster	Patterns
	1	W1, W2, W3
	3	C1, C2, C3, G1, G2, G3
	5	D2, U1, U2, U3
	8	D1, D3, R1, R2, R3, P1, P3
	10	E2, K2
	11	V2
	12	O1, O2, O3
	13	V1, V3
	14	F1, F2, F3, K1, K3, L1, L2, L3, P2
	15	J1, J2, J3
	17	I2
	20	Y1, Y2, Y3
	22	T1, T2, T3, I1, I3
	24	Q1, Q2, Q3
	25	X1, X2, X3
	27	Z1
	28	Z2, Z3
	31	S1, S2
	32	S3
	33	B1, B2, B3, E1, E3
	34	M1, M2, M3
	35	N1, N2, N3, H1, H2, H3
	36	A1, A2, A3

Table 4.2: Clustering of Characters with Radius Set to Three & α=0.8

For convenience, only the results of the other trials will be shown. Table 4.3 summarizes the results of running the program each time with a different value for the radius parameter.

Radius	Percentage of Correctly Recognized Characters	Percentage of Erroneously Recognized Characters
0 (No Architecture)	65	35
1	65	35
2	71	29
3	72	28
4	72	28
5	73	27

Table 4.3: Effect of Changing the Initial Value for the Learning Radius

The next set of experiments was conducted to observe the effect of the learning rate α on the network performance. The learning radius was fixed to three while the learning rate changed from 0.55 to 1 where α was incremented by 0.5 at each trial. Table 4.4 on the next page summarizes the obtained results.

Learning Rate α	Percentage of Correctly Recognized Characters	Percentage of Erroneously Recognized Characters
0.55	63	37
0.6	75	25
0.65	70	30
0.7	63	37
0.75	84	16
0.8	74	26
0.85	76	24
0.9	77	23
0.95	86	14
1	75	25

Table 4.4: Effect of Changing the Learning Rate

Figure 4.4 gives a graphical interpretation of Table 4.4.

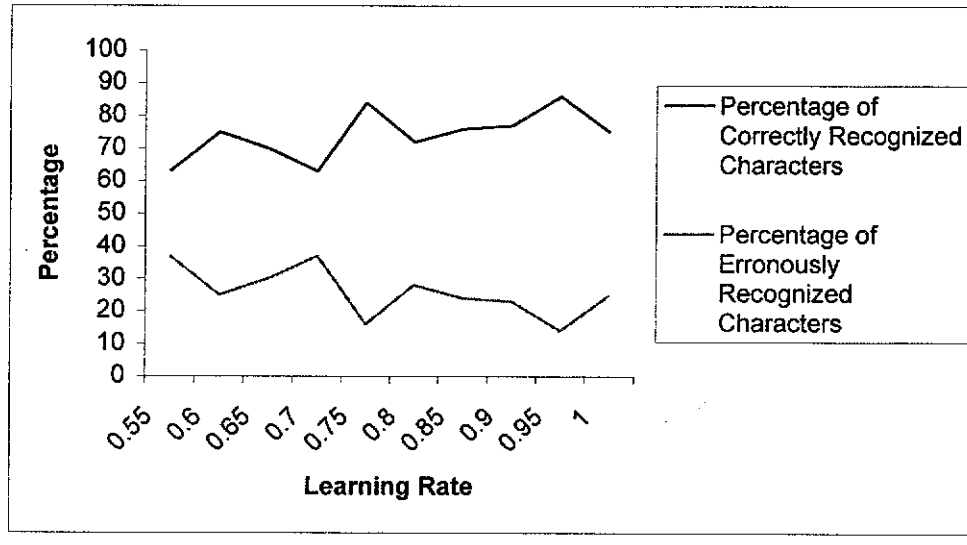


Figure 4.4: Graphical Interpretation of Table 4.4

So far the discussion has been centered on printed character recognition; a more challenging problem, however, is that of hand-written character recognition. One variation of Self-Organizing neural networks, the Adaptive-Subspace Self-Organizing Map proposed by [Zhang, 99], proved to be highly accurate in recognizing hand-written characters. A full discussion on this topic can be found in [Zhang, 99].

Based on the above results and work performed by [Fausett, 94], the following can be concluded:

1. The value of the learning radius plays a role in the ability of the network to accurately recognize characters. Lower values for the radius allows a smaller number of neighboring

clusters to learn the input pattern. Larger radius values do the opposite and improve the network performance.

2. The learning rate α is also a key parameter that affects the network performance. Relatively lower values for α produce lower correct recognition percentages. Higher values for α tend to increase the network's ability to correctly recognize characters. This is because higher α values make the weight vector of the winning cluster unit very similar to the input pattern being presented to network, thus making the network prone to clustering input patterns on the same cluster unit only if they highly "resemble" each other.
3. The architecture of the cluster units is a third factor in determining network performance. Fausett presents three different topological architectures for the output cluster units. Experiments prove that the no architecture scheme produces poor results as compared with the diamond architecture and the square grid architecture.

Chapter 5

Adaptive Resonance Networks for Character Recognition

Adaptive Resonance Theory (ART) was developed by Carpenter and Grossberg [Veelenturf, 95]. These networks cluster inputs by using unsupervised learning. Input patterns may be presented in any order. Each time a pattern is presented, an appropriate cluster unit is chosen and that cluster's weights are adjusted to let the cluster unit learn the pattern. As is often the case in clustering networks, the weights on a cluster unit may be considered to be an exemplar for the patterns placed on that cluster.

5.1 ART Networks

ART networks are designed to allow the user to control the degree of similarity of patterns placed on the same cluster. However, since input patterns may differ in their level of detail (number of components that are non-zero), the relative similarity of an input pattern to the weight vector for a cluster unit, rather than the absolute difference between the vectors, is used. As the network is trained, each training pattern may be presented several times. A pattern may be placed on one cluster unit the first time it is presented and then placed on a different cluster when it is presented later (due to changes in the weights for the first cluster if it has learned other patterns in the meantime.) A stable network will not return a pattern to a previous cluster. In other words, a pattern oscillating among different cluster units at different stages of training indicates an

unstable network. Some networks achieve stability by gradually reducing the learning rate as the same set of training patterns is presented many times. However, this does not allow the network to learn a new pattern that is presented for the first time after a number of training epochs has already taken place. The ability of a network to learn a new pattern equally well at any stage of learning is called plasticity. Adaptive resonance theory networks are designed to be both stable and plastic [Lavoie, 99].

5.1.1 Architecture

The basic architecture of an adaptive resonance neural network involves three groups of neurons: an input processing field (called the F1, layer), the cluster units (the F2 layer), and a mechanism to control the degree of similarity of patterns placed on the same cluster (a reset mechanism). The F1, layer can be considered to consist of two parts: the input portion and the interface portion. The interface portion combines signals from the input portion and the F2 layer, for use in comparing the similarity of the input signal to the weight vector for the cluster unit that has been selected as a candidate for learning. The input portion of the F1 layer is denoted as F1(a) and the interface portion as F1(b).

To control the similarity of patterns placed on the same cluster, there are two sets of connections (each with its own weights) between each unit in the interface portion of the input field and each cluster unit. The F1(b) layer is connected to the F2 layer by bottom-up weights; the bottom-up weight on the connection from the i^{th} F1(b), unit to the j^{th} F2 unit is designated b_{ij}. The F2 layer is connected to the F1(b) layer by top-down weights; the top-down weight on the connection from the j^{th} F2 unit to the i^{th} F1(b), unit is designated by t_{ij}.

The F2 layer is a competitive layer where the cluster unit with the largest network input becomes the candidate to learn the input pattern. The activations of all other F2 units are set to zero. The interface units now combine information from the input and cluster units. Whether or not this cluster unit is allowed to learn the input pattern depends on how similar its top-down weight vector is to the input vector. This decision is made by the reset unit, based on signals it receives from the input (a) and interface (b) portions of the F1, layer. If the cluster unit is not allowed to learn, it is inhibited and a new cluster unit is selected as the candidate. Each unit in the F1(a) layer is connected to the corresponding unit in the F1(b) layer. Each unit in the F1(a) and F(b) layers is connected to the reset unit, which in turn is connected to every F2 unit. Each unit in the F1(b) layer is connected to each unit in the F2 layer by two weighted pathways. The F1(b) unit X_i is connected to the F2 unit Y_j by bottom-up weights b_{ij}. Similarly, unit Y_j is connected to unit X_i by top-down weights t_{ji}.

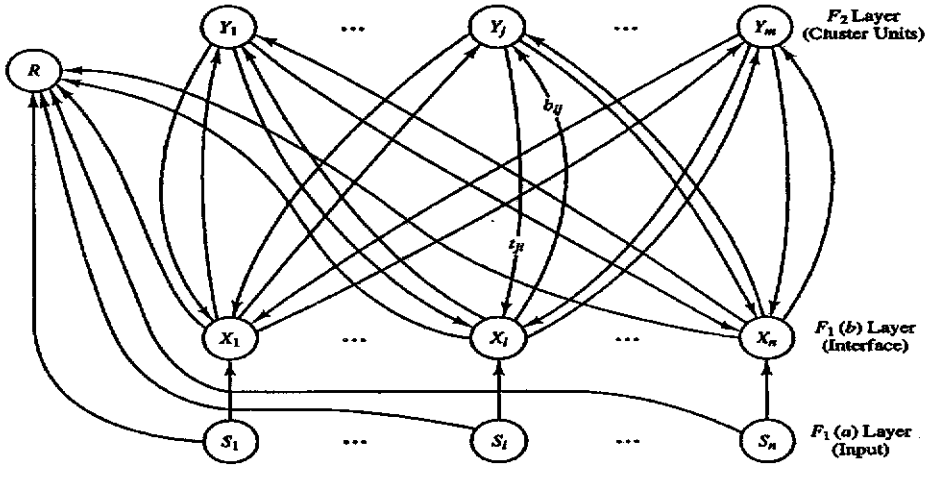


Figure 5.1: Architecture of an ART Network

5.1.2 Operation Description

It is difficult to describe even the basic architecture of ART networks without discussing the operation of the networks. A learning trial in ART consists of the presentation of one input pattern. Before the pattern is presented, the activations of all units in the network are set to zero. All F2 units are inactive. Once a pattern is presented, it continues to send its input signal until the learning trial is completed.

The degree of similarity required for patterns to be assigned to the same cluster unit is controlled by a user-specified parameter, known as the vigilance parameter. Its function is to control the state of each node in the F2 layer. At any time, an F2 node is in one of three states:

Active "on" activation = 1

Inactive "off" activation = 0, but available to participate in competition

Inhibited "off" activation = 0, and prevented from participating in any further competition during the presentation of the current input vector.

5.1.3 Algorithm

A binary input vector s is presented to the F1(a) layer, and the signals are sent to the corresponding X units. These F1(b) units then broadcast to the F2 layer over connection pathways with bottom-up weights. Each F2 unit computes its network input, and the units compete for the right to be active. The unit with the largest network input sets its activation to 1; all others have an activation of 0. This winning unit, with index J, becomes the candidate to learn the input pattern. A signal is then sent down from F2 to F1(b) (multiplied by the top-down weights). The X units, in F1(b) layer, remain "on" only if they receive non-zero signals from both the F1(a) and F2 units.

The norm of the vector x, the activation vector F1(b), gives the number of components in which the top-down weight vector for the winning F2 unit t_j and the input vector s are both 1. If the ratio of ||x|| to ||s|| is greater than or equal to the vigilance parameter, the weights (top down and bottom up) for the winning cluster unit are adjusted. However, if the ratio is less than the vigilance parameter, the candidate unit is rejected, and another candidate unit must be chosen. The current winning cluster unit becomes inhibited, so that it cannot be chosen again as a candidate on this learning trial, and the activations of the F1 units are reset to zero. The same input vector again sends its signal to the interface units, which again send this as the bottom-up signal to the F2 layer, and the competition is repeated without the participation of any inhibited units.

The process continues until either a satisfactory match is found, (i.e., a candidate is accepted) or all units are inhibited. The action to be taken if all units are inhibited must be specified by the user. It might be appropriate to reduce the value of the vigilance parameter, allowing less similar patterns to be placed on the same cluster, or to increase the number of cluster units, or simply to designate the current input pattern as a pattern that could not be clustered.

At the end of each presentation of a pattern, (i.e., after the weights have been adjusted) all cluster units are returned to inactive status but are available to participate in the next competition.

The use of the ratio ||x|| to ||s|| of the input vector in the reset calculations described above allows an ART network to respond to relative differences. This reflects the fact that a difference of one component in vectors with only a few non-zero components is much more significant than a difference of one component in vectors with many non-zero components.

The training algorithm for an ART network is taken from [Fausett, 94]. Its steps are as follows:

Step 0. **Initialize** parameters: $L > 1, 0 < p \leq 1$.

Initialize weights:

$0 < b_{ij}(0) < L/(L - 1 + n)$

$t_{ji}(0) = 1$

Step 1. **While** stopping condition is false, **do** Steps 2-13.

Step 2. **For** each training input, **do** Steps 3-12.

Step 3. **Set** activations of all F2 units to zero.

Set activations of F1(a) units to input vector **s**.

Step 4. **Compute** the norm of **s**:

$$||s|| = \Sigma_i S_i$$

Step 5. **Send** input signal from F1(a) to the F1(b) layer:

$$X_i = S_i$$

Step 6. **For** each F2 node that is not inhibited:

If $y_j \neq -1$, **then**

$$y_j = \Sigma_i b_{ij} x_i.$$

Step 7. **While** reset is true, do Steps 8-11.

Step 8. **Find** J such that $y_j \geq y_i$ for all nodes j.

If $y_i = -1$, **then** all nodes are inhibited and this pattern cannot be clustered.

Step 9. **Recompute** activation **x** of F1(b):

$$X_i = s_i\, t_{ji}.$$

Step 10. **Compute** the norm of vector **x**:

$$||\mathbf{x}|| = \Sigma_i x_i.$$

Step 11. **Test** for reset:

If $||x||/||s|| < P$, **then**

$y_j = -1$ (inhibit node J and continue, executing Step 7 again).

Else Proceed to step 12

Step 12. **Update** the weights for node J (fast learning):

biJ(new) = Lxi/(L-1 + ||x||)

tji(new) = xi.

Step 13. **Test** for stopping condition.

The stopping condition is satisfied when one of the following is satisfied:

- No weight changes,
- No units reset, or
- Maximum number of epochs reached.

There are some restrictions on the initial values of the parameters. Fausettt suggests the following initial values:

L	L>1
P	$0 < p \leq 1$
b_{ij}	$0 < b_{ij}(0) < L/(L-1+n)$
t_{ji}	$t_{ji}(0) = 1$

5.2 Learning

In ART, the changes in the activations of units and in weights are governed by coupled differential equations. The network is a continuously changing system, but the process can be simplified because the activations are assumed to change much more rapidly than the weights. Once an acceptable cluster unit has been selected for learning, the bottom-up and top-down signals are maintained for an extended period, during which time the weight changes occur. This is the resonance that gives the network its name [Fausett, 94].

Two types of learning that differ both in their theoretical assumptions and in their performance characteristics can be used for ART networks. In the fast learning mode, it is assumed that weight updates during resonance occur rapidly, relative to the length of time a pattern is presented on any particular trial. Thus, in fast learning, the weights reach equilibrium on each trial. In the slow learning mode the weight changes occur slowly relative to the duration of a learning trial; the weights do not reach equilibrium on a particular trial. Many more presentations of the patterns are required for slow learning than for fast learning, but fewer calculations occur on each learning trial in slow learning. In fast learning, the network is considered stabilized when each pattern chooses the correct cluster unit when it is presented. Because the patterns are binary, the weights associated with each cluster unit also stabilize in the fast learning mode. Also, the equilibrium weights are easy to determine, and the iterative solution of the differential equations that control the weight updates is not necessary. This is the form of learning that is typically used for ART networks [Fausett, 94].

5.3 Experimental Results and Conclusions

Like the preceding two programs, this program, which implements the above algorithm, has been developed with Visual Basic 5.0 Professional Edition. The graphical user interface resembles that described in the previous chapter. The deference is however, is in the character recognition algorithm. The experimental results presented below show the effect of the value of the vigilance on the ability of the network to recognize characters correctly. The order in which training patterns are presented to the network also plays a key role in the correct learning of the network. Another factor effecting the ability of the network to accurately recognize characters is the maximum number of output clusters allowed.

The program has been run seven times, each time with a different value for the vigilance parameter. The goal is to determine the effect the vigilance parameter has on the network output. The number of output cluster units used is 26 (one for each letter in the alphabet). The same sequence of input pattern is presented each time in this experiment:

A1, A2, A3, B1, B2, B3, C1, C2, C3, ..., Y1, Y2, Y3, Z1, Z2, Z3

Table 5.1 on the next page, shows the results obtained from a sample run of the program with the vigilance parameter set to 0.55. This value proved to be the most convenient for our particular character recognition problem. The results of the other six trials will be shown. The last column in the table is labeled NC, meaning Not Clustered.

	Cluster Unit																									
Input Pattern	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**N.C.**
A1, A2, A3	X																										
B1, B2, B3																								X			
C1, C2, C3			X																								
D1, D2, D3				X																							
E1																	X										
E2					X																						
E3																											X
F1, F2, F3										X																	
G1, G2, G3					X																						
H1																											X
H2, H3																									X		
I1, I3								X																			
I2		X																									
J1, J2, J3									X																		
K2											X																
K1, K3																X											
L1										X																	
L2, L3						X																					
M1, M2, M3												X															
N1, N2, N3													X														
O1, O2, O3														X													
P1, P3							X																				
P2										X																	
Q1, Q2, Q3															X												
R1, R2, R3																X											
S1, S2, S3																	X										
T1, T2, T3								X																			
U1, U2, U3																		X									
V1, V3																			X								
V2																				X							
W1, W2, W3																					X						
X1, X2, X3																										X	
Y1, Y2, Y3																						X					
Z1, Z2, Z3																							X				

Table 5.1: Sample Run with Vigilance = 0.55

Table 5.2 shows the percentage of the correctly recognized characters, erroneously recognized characters, and that of characters that were not clustered as the vigilance value changes in each of the seven trials.

Trial Number	Vigilance	% of Correctly Recognized Characters	% of Erroneously Recognized Characters	% of Not Clustered Characters
1	0.52	72.3	24.9	2.8
2	0.53	75.8	20.8	3.4
3	0.54	84.7	11.6	3.7
4	0.55	87.3	10.2	2.5
5	0.56	82.6	9.8	7.6
6	0.57	80.9	8.8	10.3
7	0.58	80	8.6	11.4

Table 5.2: Effect of Changing the Value for the Vigilance Parameter

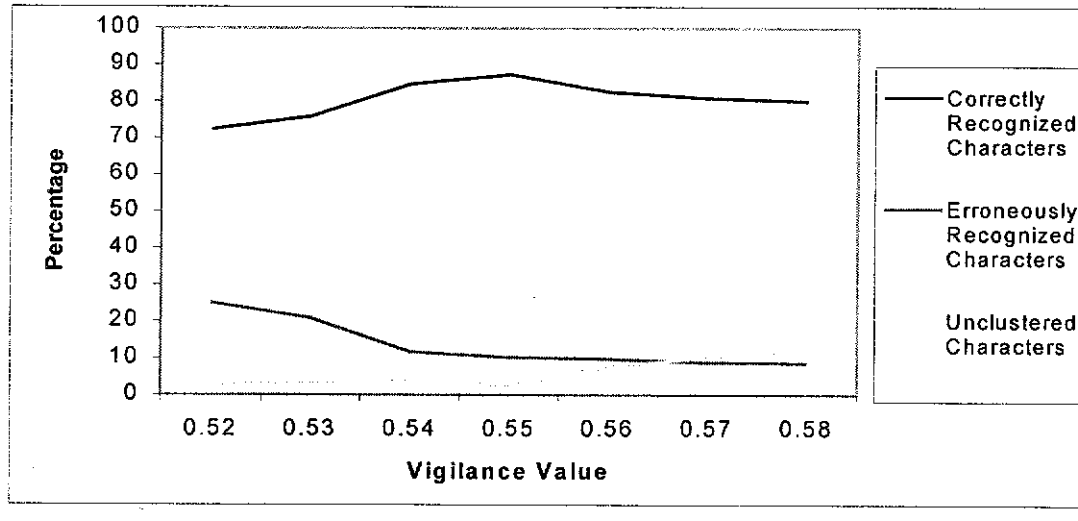


Figure 5.2: Graphical Interpretation of Table 5.2

Figure 5.1 gives a graphical interpretation of Table 5.2.

To examine the effect the input order of patterns has on the network output, the network was presented with the following input pattern sequence:

A1, B1, C1, ..., X1, Y1, Z1, A2, B2, C2, ..., X2, Y2, Z2, A3, B3, C3, ..., X3, Y3, Z3

rather than the initial input pattern sequence:

A1, A2, A3, B1, B2, B3, C1, C2, C3, ..., X1, X2, X3, Y1, Y2, Y3

For the same vigilance value (0.55) and the same number of output clusters (26), the following results were obtained:

% of Correctly Recognized Characters	**% of Erroneously Recognized Characters**	**% of Not Clustered Characters**
75.7	14.1	10.2

Table 5.3: Effect of Changing the Order of Input Pattern

Changing the number of output cluster units while keeping the other parameters constant determines the effect of the output-unit number on the performance of the network. Conducted experiments prove that having insufficient number of output units (i.e., number of units less than 26) causes the network to cluster dissimilar patterns together and depending on the vigilance value the number of unclustered pattern is determined. Having greater number of cluster units makes the network less sensitive to the order of input patterns.

Based on the above three conclusions can be drawn:

1. A relatively lower value for the vigilance parameter allows the network to cluster dissimilar clusters together and minimizes the number of unclustered pattern. A relatively higher vigilance value minimizes the possibility of clustering dissimilar patterns together, but causes the number of unclustered patterns to increase. A moderate vigilance value achieves best results. Vigilance values are problem-specific and there is no rule governing their value.

2. Contrasting the result obtained in Table 5.3 with the corresponding one in Table 5.2 (trial number 4) we find that the network output depends on the order in which input patterns are presented.

3. Insufficient number of output units causes the network to cluster dissimilar patterns together and depending on the vigilance value the number of unclustered pattern is determined. Greater number of cluster units makes the network less sensitive to the order of input patterns.

Chapter 6

Backpropagation Networks for Character Recognition

Backpropagation has been the most popular and most widely implemented of all neural networks paradigms. It is based on a multi-layered, feed forward topology, with supervised learning. This paradigm was initially developed by Paul Werbos in the early 70s, and was evolved in a series of his subsequent papers. Werbos adopted Sigmund Freud's ideas about how the brain processes information. Freud hypothesized that a chemical flow backwards from neuron to neuron takes place in a direction opposite to the forward direction of electrical excitation [Ritter, 92]. This concept triggered Werbos into looking at the backward propagation of errors through a feed forward electronic network, as a model of the learning process.

A key element in the backpropagation paradigm is the existence of a hidden layer of nodes. The network is fully connected, with every node in layer n - 1 connected to every node in layer n. A backpropagation network typically starts with random weights. Then it is exposed to a training set of input data. As it gets these inputs, it also is given the correct output that should go along with every input. The network contrasts its initial random output with what it is supposed to produce and adjusts its weights accordingly. As the training proceeds, the network's weights are incrementally adjusted until it is responding more accurately. At that point, it is ready to categorize unknown inputs in whatever application it was trained for.

Among the advantages of backpropagation are its ability to store numbers of patterns far more than its built-in vector dimensionality. On the other hand, it requires very large training sets in the learning phase, and it converges to local minima, which may or may not be global minima.

6.1 Architecture

The structure of a backpropagation network is illustrated in Figure 6.1. In the figure, a three-layered network is assumed, having an input layer, an output layer, and one hidden layer. However, a backpropagation network can contain greater numbers of hidden layers. The training of a network by backpropagation involves three stages: the feedforward of the input training pattern, the calculation and backpropagation of the associated error, and the adjustment of the weights. After training, the application of the network involves only the computations of the feedforward phase. The next section describes the training algorithm in more detail.

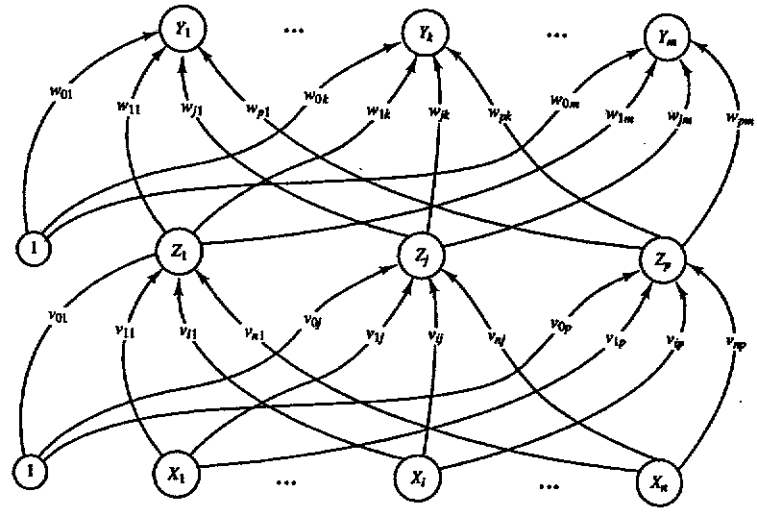


Figure: 6.1 Backpropagation Network

6.2 Training Algorithm

During feedforward, each input unit X_i receives an input signal and broadcasts this signal to the each of the hidden units $Z_i, \ldots, Z_p$. Each hidden unit then computes its activation and sends its signal z_j to each output unit. Each output unit Y_k computes its activation y_k to form the response of the network for the given input pattern.

During training, each output unit compares its computed activation y_k with its target value t_k to determine the associated error for that pattern with that unit. Based on this error, the factor δ_k (k = 1, . . ., m) is computed. δ_k is used to distribute the error at output unit Y_k back to all units in the hidden units that are connected to Y_k. It is also used later on to update the weights between the output and the hidden layer. In a similar manner, the factor δ_j (j = 1, . . ., p) is computed for each hidden unit Z_j. It is not necessary to propagate the error back to the input layer, but δ_i is used to update the weights between the hidden layer and the input layer [Ritter, 92].

After all of the δ factors have been determined, the weights for all layers are adjusted simultaneously. The adjustment to the weight w_{jk} (from hidden unit Z_i to output unit Y_k) is based on the factor δ_k and the activation z_j of the hidden unit Z_j. The adjustment to the weight v_{ij} (from input unit X_i to hidden unit Z_j) is based on the factor δ_j and the activation x_i of the input unit.

6.3 Activation Function and Nomenclature

The activation function most often used in backpropagation networks is the binary sigmoid function, which has a range of 0 to 1 and is defined as

$$f(x)= 1/(1+ e^{-x})$$

with the derivative

$$f'(x)= f(x)*(1-f(x))$$

The function is continuous, differentiable and monotonically non-decreasing [Veelenturf, 95]. Figure 6.2 illustrates the function

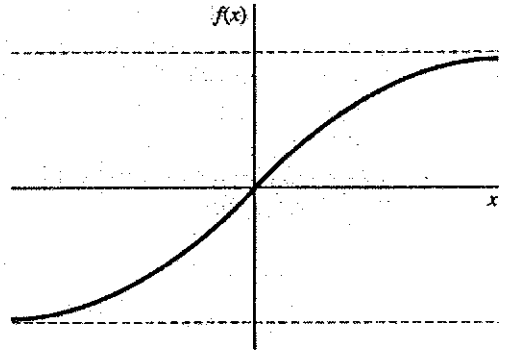


Figure 6.2: Sigmoid Function

Below is a list of symbols used in the algorithm and their meanings.

X Input training vector ($x_1...x_n$)

t Output target vector ($t_1,...,t_m$)

δ_k Portion of error correction weight adjustment for w_{jk} that is due to an error at output unit Y_k; also the information about the error at unit Y_k that is propagated back to the hidden units that feed into Y_k.

δ_j Portion of error correction weight adjustment for v_{ij} that is due to the backpropagation of error information from the output layer to the hidden unit Z_j

α Learning rate

$\mathbf{X_i}$ Input unit i (for an input unit, the input signal and the output signal are the same, namely x_i)

$\mathbf{v_{0j}}$ Bias on hidden unit j

$\mathbf{Z_j}$ Hidden unit j

The network input to Z_j is denoted z_in_j:

$Z\_in_j = v_{0j} + \Sigma_i x_i v_{ij}$

The output signal (activation) of Z_j is denoted z_j

$z_j = f(z\_in_j)$

$\mathbf{w_{ok}}$ Bias on output unit k

$\mathbf{Y_k}$ Output unit k

The network input to Y_k is denoted y_in_k:

$y\_in_k = w_{0j} + \Sigma_j z_j w_{jk}$

The output signal (activation) of Y_k is denoted y_k

$y_k = f(y\_in_k)$

6.4 Algorithm

The algorithm is taken from [Fausett, 94]. Its steps are as follows:

Step 0. Initialize weights.

Step 1. While stopping condition is false, **do** Steps 2-9.

Step 2. For each training pair, **do** Steps 3-8.

Feedforward:

Step 3. Each input unit X_i, (i = 1, ..., n) receives input signal x_i and broadcasts this signal to all units in the layer above (the hidden units).

Step 4. Each hidden unit Z_j, (j = 1, ..., p) sums its weighted input signals, $z\_in_j = v_{0j} + \Sigma_{i=1..n} x_i v_{ij}$, and applies its activation function to compute its output signal, $z_j = f(z\_in_j)$, and sends this signal to all units in the layer above (output units).

Step 5. Each output unit Y_k (k=1, ..., m) sums its weighted input signals

$Y\_in_K = w_{0k} + \Sigma_{j=1...p} z_i w_{jk}$ and applies its activation function to compute its output signal

$Y_k = f(y\_in_k)$

Backpropagation of error:

Step 6. Each output unit Y_k (k = 1, ..., m) receives a target pattern corresponding to the input training pattern, computes its error information term

$\delta_k = (t_k - y_k)f'(y\_in_k)$

calculates its weight correction term

$\Delta w_{jk} = \alpha \delta_k z_j$

Calculates its bias correction term

$\Delta w_{0k} = \alpha \delta_k$

and sends δ_k to units in the layer below

Step 7. Each hidden unit Z_j (j=1, ..., p) sums its delta inputs

$\delta\_in_j = \Sigma_{k=1..m} \delta_k w_{jk}$

Multiplies by the derivative of its activation function to calculate its error information term

$\delta_j = \delta\_in_j \, f'(z\_in_j)$

calculates its weight correction term

$\Delta v_{ij} = \alpha \delta_j x_i$

and calculates its bias correction term

$\Delta v_{0j} = \alpha \delta_j$

Update weights and biases:

Step 8. Each output unit Y_k (k = 1...m) updates its bias and weights (j = 1...p)

$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$

Each hidden unit Z_j (j=1...p) updates its bias and weights (i = 0...n)

$V_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$

Step 9. **Test** Stopping Condition

6.5 Experimental Results and Conclusions

The experimental results in this section are taken from [Brown, 93]. The paper from which the results are taken explores the application of neural networks to the problem of identifying machine-printed characters in an automated manner. In particular, a backpropagation network is trained on an eighty-four-character font and tested on two other fonts. The paper explores the differences between two different character recognition algorithms: a feature extraction method using traditional artificial intelligence techniques for classification, and a neural network approach

with virtually no preprocessing. Only the results obtained from the backpropagation neural network approach will be considered in this report.

The data used consists of three 84-character fonts. The character resolution for all three fonts is 8X8. The sets contain upper and lower case letters as well as numbers and a handful of miscellaneous punctuation marks. The sets are all used as actual display characters on an old CBM C128 computer so they represent real-world data; they were not designed for this experiment [Brown, 93].

```
(D)isplay char, (R)un new set, or (Q)uit:  d

Character number:  68
Character name:  K

  XX  *X
  ** **
  ****
  **X
  ****
  ** **
  XX  *X
```

Figure 6.3: Example of a Character Display

The neural network approach utilized three separate steps. The first step simply translated the binary character data into a friendlier form. The second step took the output of the first and trained a backpropagation network on it, outputting all the resulting weights and general network information. The third step took the output of the second and created a network. It then ran a full character set through the network and output identification information for all the characters the set contained. The network consisted of 64 inputs, 96 hidden nodes, and 7 outputs. It was essentially a flat feedforward network that was fully connected without self-inputs or biases [Brown 93]. Figure 6.4 below shows the network architecture.

Each of the sixty-four inputs corresponds to one of the pixels in the 8X8 character. An input was taken to be zero if the pixel was empty, or a one otherwise. The seven outputs were simply used to make a seven bit numerical label (unique labels for 84 characters require seven bits) that coincided with the ordering of the character set. The labels ran from 0 to 83.

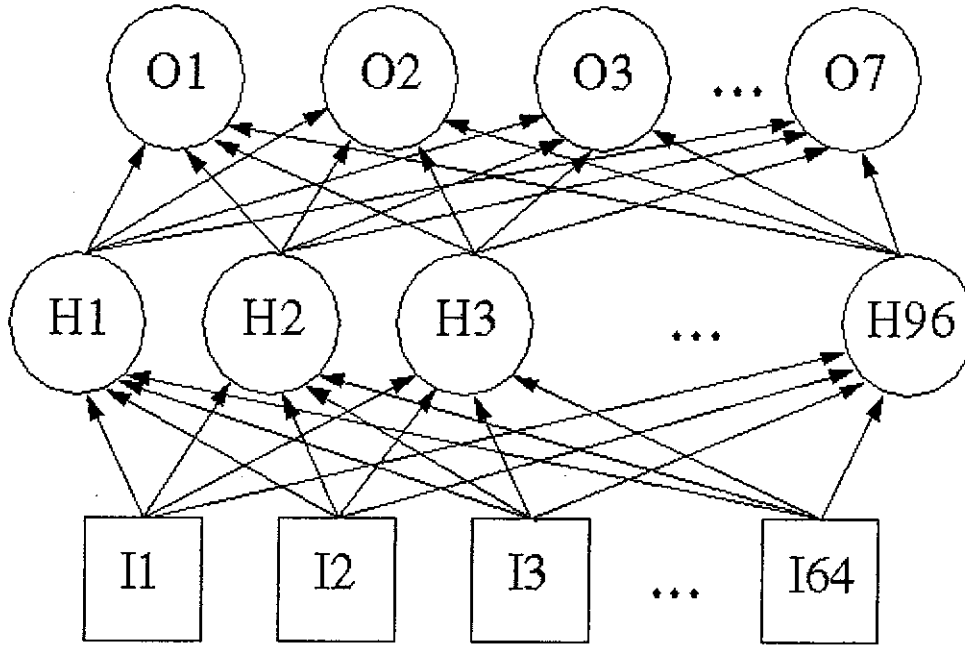


Figure 6.4: Backpropagation Network Topology

Training the network was troublesome and fraught with difficulties. The learning program required several hundred computer hours on a dedicated Sun SPARCstation to run, making it practically impossible to really try and obtain optimal results. It is almost guaranteed that 96 is not the optimal number of hidden nodes; this number was produced by multiplying the number of inputs by 1.5.

With all this difficulty stated, the network was made to learn properly by using a step size of 1.05. Not only did it manage to converge with this particular step size, it converged fairly quickly and only required 136 sweeps through the data set. Once a properly trained network had been obtained, it produced reasonable results for an unoptimized algorithm [Brown 93]. Table 6.1 shows the obtained results.

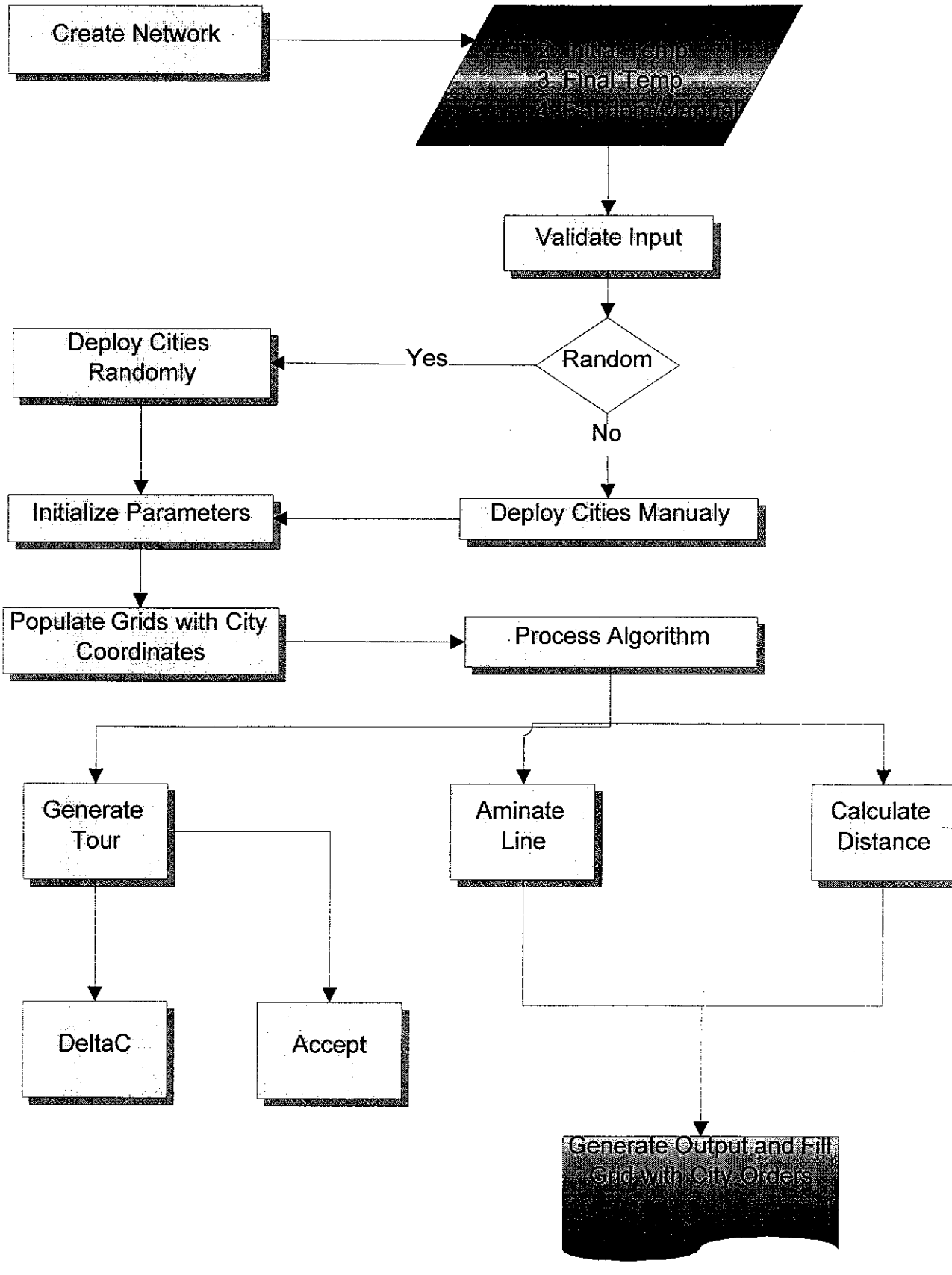
Results for the "Alt" Character Set		
Total Correct	66	79%
Total Correct (without counting identical characters)	15	18%
Total Unknowns	2	2%
Total Wrong Guesses	16	19%
Total Wrong Or Unknown	18	21%
Results for the "Ult" Character Set		
Total Correct	33	39%
Total Correct (without counting identical characters)	12	14%
Total Unknowns	3	4%
Total Wrong Guesses	48	57%
Total Wrong Or Unknown	51	61%
Total Combined Results		
Total Correct	99	54%
Total Correct (without counting identical characters)	27	16%
Total Unknowns	5	3%
Total Wrong Guesses	64	38%
Total Wrong Or Unknown	69	41%

Table 6.1: Results for the "Alt" and "Ult" Character Recognition

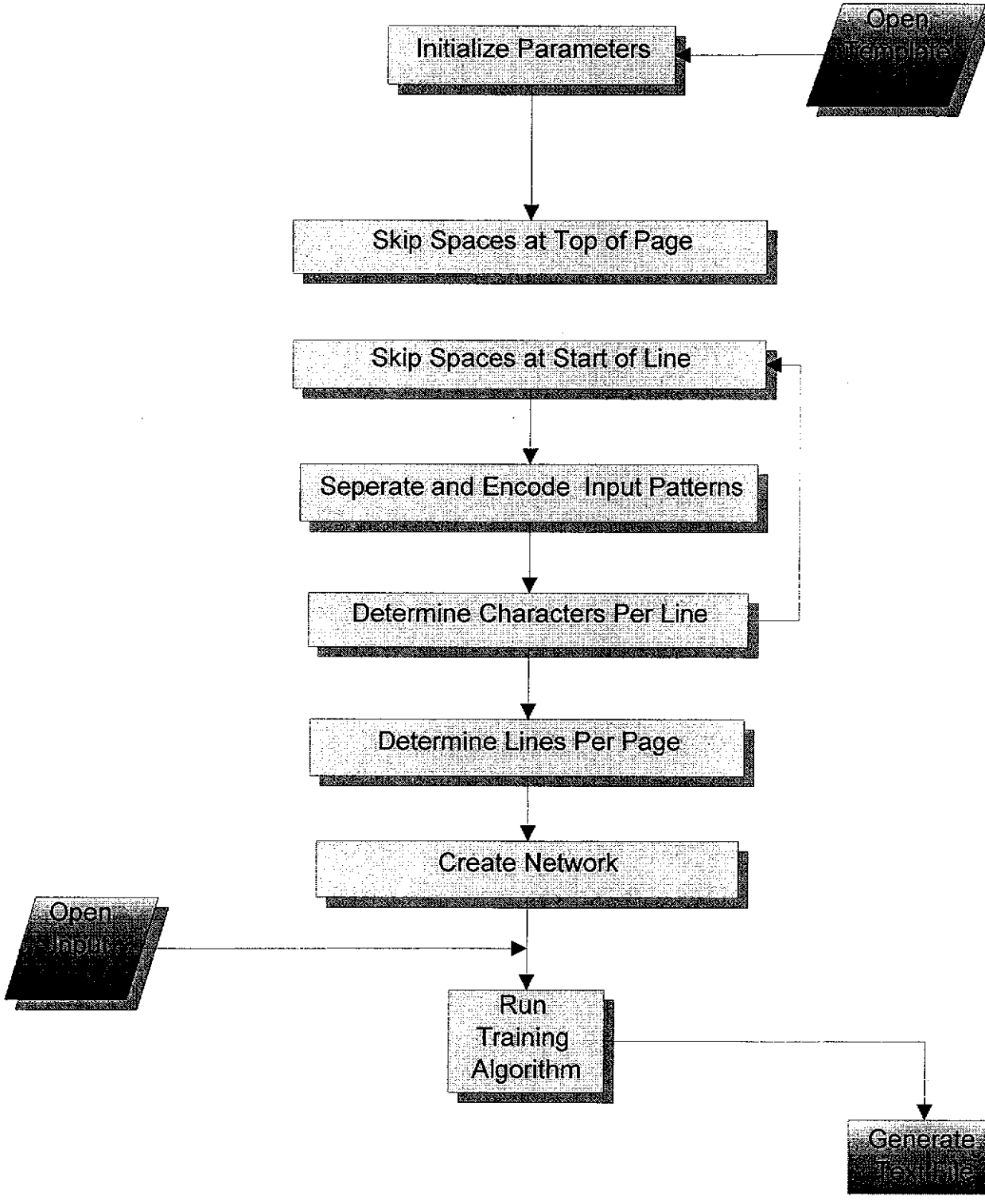
The results in the above table could hardly be compared to those obtained in chapter 4 and five since they don't all work on the same character set. However, having observed the results in the three chapters, one may be able to say that the ART network produced the most favorable results as far as this particular character recognition problem is concerned.

Module Design Appendix

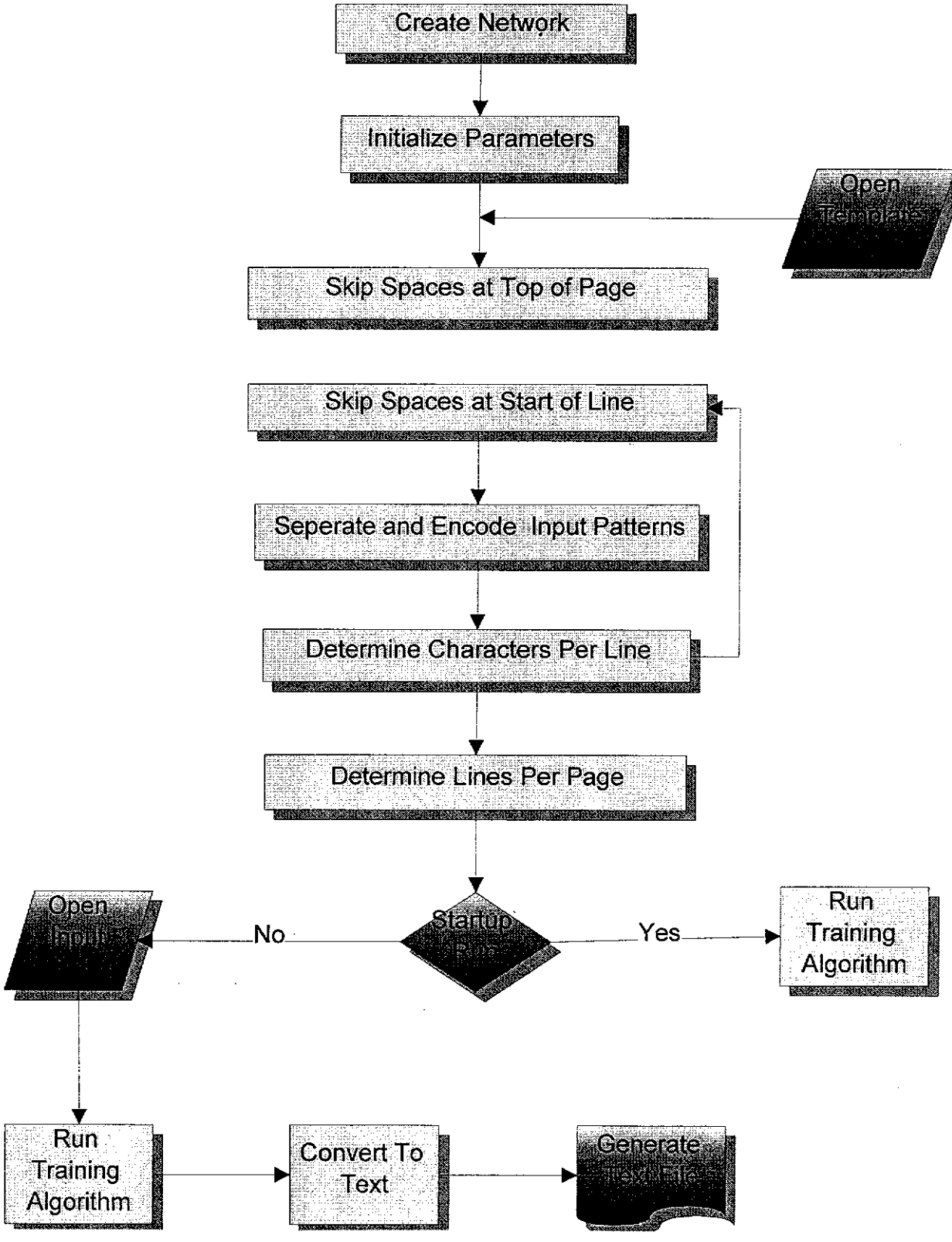
Module Design for TSP Program



Module Design for Kohonen's Self-Organizing Map Program



Module Design for ART Network Program



References

[Aggarwal and Bennett 99] R. K. Aggarwal and A. Bennette, *"A Novel Approach to Fault Diagnosis in Multi Circuit Transmission Lines Using Fuzzy ARTmap Neural Networks"*, IEEE Transactions on Neural Networks, Volume 10, Number 5 (September 1999), pages 1214 – 1221.

[Brown 93] E. W. Brown, "*Applying Neural Networks to Character Recognition*", http://www.ccs.neu.edu/home/feneric/charrecnn.html.

[Burkee and Damany 92] L. I. Burkee, and P. Damany, *" The Guilty Net for the Traveling Salesman Problem"*, Computers Operations Research, Volume 19, Number 3 (September 92), pages 255 – 265.

[Chester 93] M. Chester, *Neural Networks: A Tutorial*, Printice-Hall, NJ (1993).

[Fausett 94] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, And Applications,* Printice-Hall, NJ (1994).

[Freeman and Skapura 91] J. A. Freeman and D. M. Skapura, *Neural Networks: Algorithms, Applications, and Programming Techniques,* Addison-Wesley, NY (1991).

[Gelenbe 91] E. Gelenbe, *Neural Networks Advances and Applications,* Elsevier Science Publishers, NY (1991).

[Harvey 94] R. L. Harvey, *Neural Network Principle,* Printice-Hall, NJ (1994).

[Lavoie and Crespo 99] P. Lavoie and J. F. Crespo, *"Generalization, Discrimination, and Multiple Categorization Using Adaptive Resonance Theory"*, IEEE Transactions on Neural Networks, Volume 10, Number 4 (July 1999), pages 757 – 763.

[Lippman 87] R.Lippman, *"An Introduction to Computing with Neural Networks"*, IEEE ASSP Magazine, April 1987, pages 4-22.

[Ritter and Martinetz 92] H. Ritter and T. Martinetz, *Neural Computation and Self-Organizing Maps*, Addison-Wesley, NY (1992).

[Simpson 96] P. Simpson, *Neural Networks Applications,* IEEE Technology Update Series (1996), pages 102 – 116.

[Takefuji 96] Y. Takefuji, *Neural Network Parallel Computing*, Kluwer Academic Publishers, MA (1996).

[Thathachar and Arvind 99] M. A. L. Thathachar and M. T. Arvind, *"Global Boltzmann Perceptron Network for On-Line Learning of Conditional Distributions"*, IEEE Transactions on Neural Networks, Volume 10, Number 5 (September 1999), pages 1093 – 1098.

[Veelenturf 95] L. P. J. Veelenturf, *Analysis and Applications of Artificial Neural Networks*, Printice-Hall, NJ (1995).

[Zhang and Jabri 99] B. Zhang and M. Jabri, *"Handwritten Digit Recognition by Adaptive Subspace Self-Organizing Map (ASSOM)"*, IEEE Transactions on Neural Networks, Volume 10, Number 4 (July 1999), pages 939 – 946.