1996

# Modular Approaches for Designing Feedforward Neural Networks to Solve Classification Problems.

Parvin Hashemian
*Louisiana State University and Agricultural & Mechanical College*

# INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**UMI**

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

# MODULAR APPROACHES FOR DESIGNING FEEDFORWARD NEURAL NETWORKS TO SOLVE CLASSIFICATION PROBLEMS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Parvin Hashemian
B.A., New York University, 1980
M.S., Columbia University, 1981
December 1996

## ACKNOWLEDGMENTS

In the name of God, the Beneficent, the Merciful. I have been blessed to have the opportunity to pursue my higher education. My appreciation is expressed to all my teachers at Louisiana State University. In particular, I would like to thank my major Professor, Dr. Subhash C. Kak, for his continuous guidance and encouragement during my studies at Louisiana State University.

My special appreciation is expressed to those individuals who have helped me directly or indirectly during these years, but their names are not mentioned here.

I wish to gratefully acknowledge my parents for their support and encouragement toward my education, especially my father who always strived to provide me with the best opportunity to continue my education. Also, I am grateful to my husband, Mohammad R. Madani, for his patience and continuous support and encouragement during these years. In the end, I would like to thank our children, Fatemeh, Hossein, and Ali who have been the sources of our joy and happiness during these years.

# TABLE OF CONTENTS


ACKNOWLEDGMENTS ..... ii

ABSTRACT ..... v

CHAPTER 1. INTRODUCTION ..... 1
1.1 Biological Neurons ..... 1
1.2 Artificial Neural Networks ..... 4
1.3 General Structure of an Artificial Neuron ..... 6
1.4 Neural Nets as Classifiers ..... 9

CHAPTER 2. FEEDFORWARD NEURAL NETWORKS ..... 16
2.1 Single-Node Neural Network ..... 16
2.2 Linear Separability ..... 18
2.3 Single-Node Network Learning Algorithms ..... 20
2.3.1 The Gradient Descent Method ..... 20
2.3.2 Learning Based on Gradient Descent ..... 21
2.3.3 Widrow-Hoff Delta Rule ..... 22
2.3.4 Perceptron Learning Rule ..... 25
2.3.5 Generalized Delta Rule ..... 26
2.4 Capacity of Single-Node Network ..... 28
2.5 Multi-node Neural Network ..... 29
2.6 Multi-Layer Feedforward Neural Nets ..... 29
2.7 The Backpropagation Algorithm ..... 31
2.7.1 Analysis of Rate of Convergence of Backpropagation ..... 34
2.7.2 Deficiencies of the Backpropagation Algorithm ..... 39

CHAPTER 3. A MODULAR APPROACH FOR SOLVING CLASSIFICATION PROBLEMS ..... 42
3.1 Introduction ..... 42
3.2 Goals Behind Proposed Algorithm ..... 42
3.3 Proposed Algorithm ..... 43
3.3.1 Alternative to the Proposed Algorithm ..... 45
3.3.2 Example ..... 46
3.3.3 Analysis of the Algorithm ..... 50
3.4 Handwritten Digit Recognition ..... 53
3.4.1 Data Bases ..... 54
3.4.2 Learning Rule ..... 54
3.4.3 Performance Measure and Assignment Criteria ..... 56
3.4.4 Results and Discussion ..... 59
3.5 Conclusion ..... 64

CHAPTER 4 TRAINING USING CORNER CLASSIFICATION ..... 77
4.1 Introduction ..... 77
4.2 The Network Structure ..... 78
4.3 Corner Classification Algorithms ..... 79
4.3.1 Algorithm CC2 ..... 81
4.3.2 Algorithm CC3 ..... 82
4.3.3 A Remark on Corner Classification ..... 83
4.3.4 Modified CC2 ..... 86
4.3.5 Modified CC3 ..... 87
4.3.6 Simulation Results ..... 89
4.4 Pruning of Hidden Nodes in Corner Classification ..... 93
4.4.1 A Procedure ..... 94
4.4.2 Simulation Results ..... 97

CHAPTER 5 CONCLUSION ..... 98

REFERENCES ..... 101

VITA ..... 106

## ABSTRACT

Perhaps the most popular approach for solving classification problems is the backpropagation method. In this approach, a feedforward network is initially built by an intelligent guess regarding the architecture and the number of hidden nodes and then trained using an iterative algorithm. The major problem with this approach is the slow rate of convergence of the training. To overcome this difficulty, two different modular approaches have been investigated. In the first approach, the classification task is reduced to sub-tasks, where each sub-task is solved by a sub-network. An algorithm is presented for building and training each sub-network using any of the single-node learning rules such as the perceptron rule. Simulation results of a digit recognition task show that this approach reduces the training time compared to that of backpropagation while achieving comparable generalization. This approach has the added benefit that it develops the structure of the network while learning proceeds as opposed to the approach of backpropagation where the structure of the network is guessed and is fixed prior to learning. The second approach investigates a recently developed technique for training feedforward networks called corner classification. Training using corner classification is a single step method unlike the computationally intensive iterative method of backpropagation. In this dissertation,

modifications are made to the corner classification algorithm in order to improve its generalization capability. Simulation results of the digit recognition task show that the generalization capabilities of the networks of the corner classification are comparable to those of the backpropagation networks. But the learning speed of the corner classification is far ahead of the iterative methods. Designing the network by corner classification involves a large number of hidden nodes. In this dissertation, a pruning procedure is developed that eliminates some of the redundant hidden nodes. The pruned network has generalization comparable to that of the unpruned method.

# CHAPTER 1

# INTRODUCTION

## 1.1 Biological Neurons

The human brain contains about $10^{11}$ neurons [Tho85]. A neuron is a cell. The major role of a neuron is to transmit information to other neurons or to other (muscle or gland) cells. A typical neuron has three major parts: the cell body containing the nucleus, the dendrites, and the axon. A neuron usually receives information at its dendrites, and sends information out to other neurons and cells along its one long fiber, the axon.

As the axon approaches its target cells, it branches into a number of smaller fibers that end in synaptic terminals or knobs. These terminals form synapses with other cells. The synapse is the place where one neuron transmits information to another. A given neuron in the brain may have several thousand synaptic connections with other neurons. If the human brain has $10^{11}$ neurons, then it must have at least $10^{14}$ synapses.

A neuron's ability to generate and conduct electrical impulses depends on the different kinds of protein molecules in its cell membrane. These protein molecules are known as different kinds of ion channels, such as sodium and potassium channels. These molecules are said to

be "voltage gated" because the voltage differences across the cell membrane determine whether the channel is open or closed. Most of the potassium channels in the membrane are without a gate and are open, and most potassium ions are inside the cell. On the other hand, most sodium channels are outside the cell. The differences in the concentrations of sodium and potassium ions inside and outside the cell results in a resting potential of about -70 millivolts.

The action potential is the quick voltage change that sweeps along the neuron membrane. It begins with a slight reduction in the negative potential across the membrane where the axon leaves the cell body. This voltage change opens some of the sodium channels for a short time (about half a millisecond). Sodium ions ($Na^+$) rush in and add to the inside of the local region of the membrane where the sodium channel has opened positive relative to outside (+50 millivolts). This voltage change causes the sodium channels to close and the potassium channels to open. The potassium ions ($K^+$) move out until the resting membrane potential is restored there. Meanwhile, the membrane potential at the next closed sodium channel is a little less negative than at its rest since some sodium ions have been accumulated there. When the potential at the next closed sodium channel becomes sufficiently less negative

(about -60 millivolts) its gate opens and the process goes on down the axon.

A neuron can affect another cell with which it synapses by either increasing or decreasing its activity. Synaptic excitation increases the activity of the target neuron whereas synaptic inhibition decreases its activity. Almost all synapses in the mammalian nervous system are chemical in nature, and only some are electrical. In chemical synapses, information is passed from one cell to another by means of neurotransmitter molecules. In electrical synapses, electrical impulses are transmitted directly from one cell to another.

In chemical synapses, a space about 20 nanometers wide, called a synaptic cleft, separates the terminal knob of the axon referred to as the presynaptic terminal and the target cell membrane, or postsynaptic membrane. There are many small vesicles inside the presynaptic terminal near the membrane that are filled with the chemical neurotransmitter molecules.

The process of chemical synaptic transmission starts when an action potential arrives at the terminal knob of an axon. Upon its arrival, large numbers of neurotransmitter molecules are released into the synaptic cleft. Every neuron releases only one kind of neurotransmitter molecules which either excites or inhibits the postsynaptic cell with which it synapses. The released neurotransmitter molecules

diffuse to the postsynaptic membrane and attach to receptor protein molecules in it. This attachment causes a shape change in the receptor molecules which further causes some ion channels to open.

As a result of the flow of ions, the target postsynaptic membrane potential changes. Finally, what determines whether or not a neuron generates an action potential is the combination and magnitude of all excitations and inhibitions it receives. If the magnitude of this combination is above the action potential threshold level an action potential is initiated at the start of the axon.

The electrical characteristics of the neuron are captured by the McCulloch-Pitts neuron model, where each neuron is a threshold circuit. Networks of such model neurons have been investigated for their computational abilities.

## 1.2 Artificial Neural Networks

Present computer systems are able to successfully solve a variety of tasks. But in problems that are hard to define, such as speech or image recognition, present computer systems fail, performing far behind the brain. Although the functioning of the brain is imperfectly understood, it has provided inspiration for new ideas for developing more intelligent systems. Artificial neural

networks, or simply neural networks or neural models, are the results of the first steps in this direction.

Artificial neural networks estimate a function without a mathematical definition of how outputs depend on inputs. Training data forms the input-output function of neural models. In other words, neural networks "learn from experience". "Learning from experience" without mathematical formulas enables systems to generalize. Generalization is a property of a system to generate an appropriate output in response to an unseen input.

Problems that are hard to define generally require an enormous amount of processing. The brain accomplishes these problems using massive parallelism. Instead of performing a set of instructions sequentially, as in a von Neumann computer, neural models process information simultaneously using massively parallel nets. Information is processed in a neural net over the entire network in a distributive manner, not at specific places as is done in conventional systems. This style of information storage makes the network fault tolerant since no single site is of critical importance.

Development on artificial neural networks began more than fifty years ago. The McCulloch and Pitts model of 1943 led to the study of simple neural networks represented as electrical circuits. Another important contribution was Donald Hebb's book, *The Organization of Behavior* (1949),

which pointed out that a neural pathway is reinforced each time it is used; this is now known as Hebb's learning rule. In 1960, two similar neural models were developed independently, perceptron by Rosenblatt [Ros73], and Adaline by Widrow and his colleagues [Wid90]. These networks were able to perform input to output mapping through simple learning algorithms. It was believed that these networks were the right start in developing more sophisticated neural networks. However, it was soon shown that these models were incapable of mapping functions such as XOR. This slowed down research in neural networks for several years. Later, in the early eighties, networks such as Hopfield's feedback model [Hop82] and the backpropagation algorithm [Rum86] triggered renewed interest in neural networks.

## 1.3 General Structure of an Artificial Neuron

The artificial or model neuron is based on the salient signal characteristic of biological neurons. An artificial neural network consists of a large number of model neurons usually called nodes that are connected together. Information (signal) is passed from one node to another through a connection line. A connection line simulates an axon or a dendrite in the nervous system. Associated with each connection line there is a weight which is analogous to the connection strength at a synapse in the nervous system. Each node performs a function. In general, the

function that a node performs involves three parts. In the first part, each input signal coming through a connection line to the node is multiplied by the weight on that connection line. All such weighted inputs are summed together and combined with the threshold of the node at the summing element of the node. Finally, the output of the summing element is usually passed through a non-linearity (activation function) to produce the output of the node. The non-linearity is sometimes a simple threshold function which produces ±1 as output. In general, other forms of non-linearity, mostly of sigmoid type, have been used. Figure 1.1 shows the structure of a typical node along with its two typical non-linearities (activation functions).

In real networks of neurons learning occurs at synapses; when the connection strengths at synapses change the neurons behavior change [CAM92]. Learning in artificial neural networks occurs when the weights on the connection lines between the nodes change.

Several types of artificial neural models have been described in the literature; these vary according to the network architecture, node characteristic (i.e. activation function used for nodes), and the training or learning algorithm used to adjust the connection weights so that the network responds correctly to an input [Lip87], [Vem90].

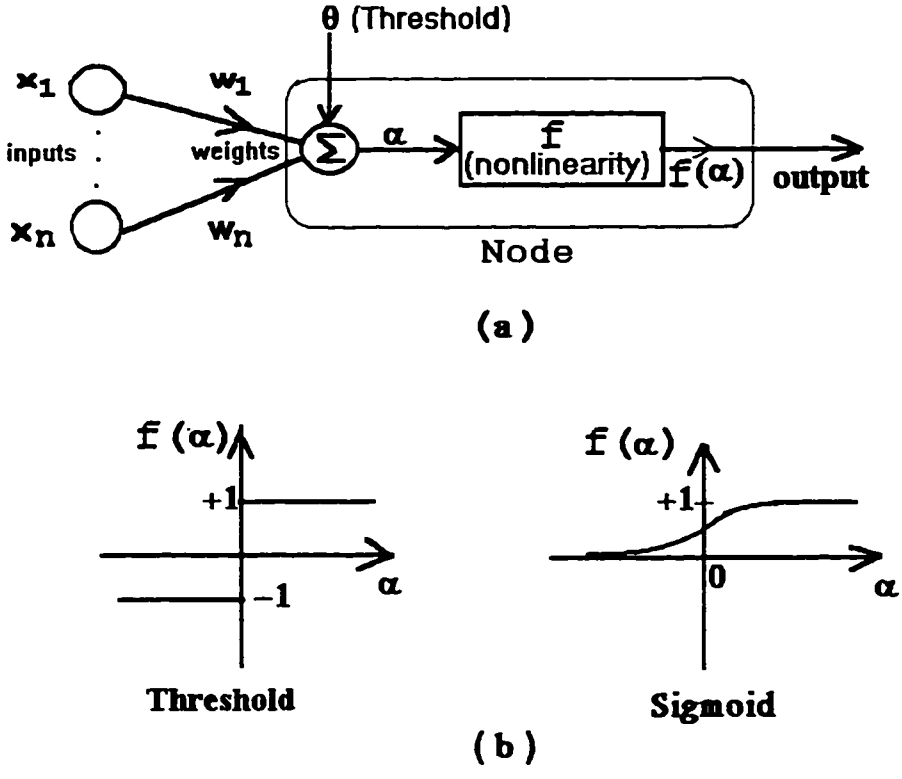


Figure 1.1 (a) The structure of a typical node
(b) Two typical activation functions.

The arrangement of nodes and their interconnections in an artificial neural network defines the architecture of the network. There are different types of architectures [McS92]. In one kind of architecture, known as the feedback networks [Hop82], [Kak89], [Kak92], and [Kak93a], there is one layer of nodes; the outputs of the nodes are fed back to their inputs. The feedback networks are used as associative memories [Sti88], [Has90], [Has91], or for optimization problems [Hop85].

In another architecture, the network consists of distinct layers of nodes; the nodes in each layer receive their input from the nodes of the preceding layer and feed their output to the nodes in the succeeding layer. The networks of this kind of architecture are known as feedforward networks [Wid90]. Biological structures, such as the vision system, have aspects of feedforward as well as feedback networks. Feedforward networks are used for pattern classification problems. This dissertation focuses on feedforward networks.

## 1.4 Neural Nets as Classifiers

In classification problems, given a set of disjoint known classes, the task is to assign each input pattern (to be classified) to one of the known classes. A pattern is represented in terms of a set of n features; as a n-dimensional binary vector.

A popular approach in solving multi-class classification tasks is a monolithic approach. In this approach, the separation of the m classes is considered as one problem. A network is designed. Given a set of sample patterns of the m classes - called a training set - the network is trained to assign the samples in the training set to their respective classes. The training algorithm used is called the backpropagation algorithm.

This approach, though very powerful on relatively small problems, breaks down as soon as sufficiently complex problems are considered [Bal93]. The followings are the major problems with this approach:

(1) The backpropagation algorithm is not guaranteed to converge to a solution. For large problems, even if this algorithm converges, the speed of convergence is very slow. The computation time is proportional to the number of weights in the network. Bigger networks have more weights to train, and also have more training samples to present to the network for training. As a result, as the size of the network (problem) increases the time required to train the network increases by even more.

(2) The number of hidden nodes needed in a network trained by the backpropagation algorithm must be picked "right". Also, the architecture of the

network must be fixed prior to learning. If the number of hidden nodes is picked too small, the training samples (the problem) would not be learned and a larger network would have to be retrained. If this number is picked too large, a pruning procedure would be needed as the network's generalization capability would degrade. On the other hand, there is no exact guideline for choosing this number.

Biology seems to have chosen another approach in solving complex problems. Studies of human and animal brains suggest that different parts of the brain are specialized for different tasks [Sha92], [Gol92], [Dam92]. An example of functional specialization is the visual system. Studies show that different areas in the visual cortex are individually specialized to perform different tasks; i.e. color, form, and motion are processed separately in parallel [Zek92].

Following the approach of nature, some researchers have suggested modular approaches in solving multi-class classification problems.

Some approaches have chosen a two-stage classification scheme in which the first stage scans the input and recognizes the overall features of it. Depending on the results of the first stage, either one [Cho92] or several [Wan91] of the networks of the second stage are selected to

resolve the confusion among similar classes and perform the final mapping. Networks based on the backpropagation algorithm are used in their schemes. This way of mapping, where several smaller neural networks are designed and used, requires less training time than the approach of designing a single network trained by backpropagation.

In another approach for recognition of hand-printed Chinese characters [Yon88], a neural network based on the Hopfield model is used. This network is a content addressable associative memory. It consists of one fast sub-network and several slow sub-networks. The fast sub-network is used to recognize the overall structure of the input whereas the slow sub-networks tell us about the detail. Being based on a content addressable memory, this network uses a large number of nodes.

Not knowing the optimum network for a task, in [Alp93] several single networks are independently developed and trained for the same task. Then a vote over their responses is taken. Such an approach uses far more resources than a modular approach.

Some researchers have developed more modular schemes than the two-stage scheme mentioned earlier. In [Aud94], a modular network is designed for a character recognition task. At first, they divide the classes into a number of groups of classes. This is done by an unsupervised network; a network trained using an unsupervised learning algorithm.

This way they have divided the task into a number of simpler sub-tasks. Then, each sub-task, which is to classify the classes within its group, is solved using one module trained by backpropagation. Upon the arrival of an input character, each module independently tries to recognize it. In this sense this approach is more modular than the two-stage approaches mentioned earlier where the first stage scans the input and then one of the several networks in the second stage is activated. Then, another layer above the outputs of the modules is introduced whose task is to integrate the work of the different modules in order to vote for the final decision. They have shown that their scheme gives a faster learning time and a somewhat higher accuracy in response than a single backpropagation network.

In some other approaches, the modularity is designed by the network-builder in order to use resources more efficiently [Ana95]. Instead of using a network for deciding on the different modules needed to carry out the task, [Ana95] simply uses a module to separate each class from all other classes. In other words, their approach reduces a m-class classification task into a set of m 2-class sub-tasks. Each module of a sub-task is trained by backpropagation. They have also shown that their approach is much faster than a non-modular backpropagation network while achieving comparable performance results.

In another approach [Kne90], [Kne92], in which the modularity is still planned by the network builder, simpler networks are used. Here, networks having one layer of trainable connections are used instead of multi-layer trainable networks of backpropagation. They have shown that these networks are simpler in structure and take much less time to train while they achieve comparable performance to the single networks of backpropagation.

In this dissertation, we have used different modular approaches to a multi-class classification problem. After a review of the different feedforward learning algorithms in chapter 2, chapter 3 presents one such modular approach. In this approach also the modularity is designed by the network-builder; a m-class classification task is broken into m 2-class sub-tasks. Each sub-task is independently solved by a sub-network having one layer of trainable connections. An algorithm is presented for building and training each sub-network. Our algorithm starts with one node per class and creates more nodes as needed to separate each class from all others. This approach differs from the approach of [Kne92] which is more of a pairwise separation between classes and as such it tends to create more nodes. Also, in the case where two classes are not pairwise linearly separable we suggest simple automatic rules which are carried out without user intervention, instead of using backpropagation networks or other more complex approaches.

Furthermore, in their simulations, [Kne92] have used nodes with sigmoidal function, whereas in our simulations we have used nodes with simple threshold function which is much easier for hardware implementations.

The corner classification approach [Kak94], which is a more modular approach, is presented in chapter 4. In this approach, the classification task is broken into sub-tasks where each sub-task is to separate each sample training pattern along with its neighboring patterns from all other training patterns. This contrasts with the approach of chapter 3 in which each sub-task was to separate all the training samples of one class from all other patterns. In the corner classification, each sample training pattern is considered a corner in the n-dimensional cube to be separated from all other training patterns using a single hidden node.

In chapter 4, improvements are also made to the corner classification algorithms in order to improve their generalization capability. The modular structure of this network along with its fast learning algorithms make this approach much faster than all the previously mentioned approaches. Finally, a pruning procedure for the networks of corner classification is introduced that reduces the number of the hidden nodes in the network. Chapter 5 concludes the work of this dissertation.

# CHAPTER 2

# FEEDFORWARD NEURAL NETWORKS

## 2.1 Single-Node Neural Network

The single-node neural network [Wid90] consists of an input layer of nodes and a single output node. The input layer does not perform any operation on input data. It only passes the data through the connection weights to the single output node. The output node is a processing node of the kind discussed in 1.1. The name single-node comes because there is only one processing node in the network. The threshold of the output node is simulated in the following way. There is a special connection between a constant input $x_{n+1} = 1$ and the output node. The weight on this connection, $w_{n+1}$, simulates the threshold level of the output node. By changing this weight, $w_{n+1}$, the threshold level of the output node is changed.

Upon presentation of an input vector at the input layer the corresponding output of the output node is computed. This output is compared to the desired output. If there is a discrepancy, an error measure is computed which is then used to adjust the weights so as to produce the desired output. This process is shown in Figure 2.1. Depending on the different error measures taken, there are different learning algorithms for adjusting the weights so that it responds correctly to the given input patterns and

input patterns not presented during training. These learning algorithms, the perceptron rule, the delta rule, and the generalized delta rule are explained in the following sections.

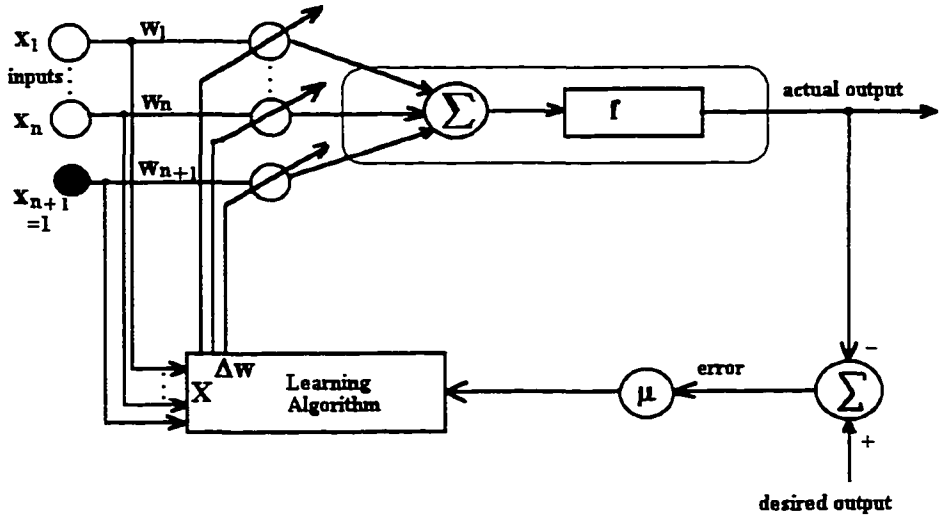


Figure 2.1. A single-node network

## 2.2 Linear Separability

Many real world problems involve the categorization of regions of points in a n-dimensional space into different classes. One way to define this categorization is to choose hyperplanes to separate the space into the proper regions [Fre91]. In a n-dimensional space a hyperplane is a n-1 dimensional object. The equation of a hyperplane in a n-dimensional space is;

$$\sum_{i=1}^{n} w_i x_i = \theta \qquad (2.1)$$

where $w_i$s and $\theta$ are constants such that at least one $w_i \neq 0$, and $x_i$s are the coordinates of the space.

The single-node neural network used the above technique to classify a set of patterns. Consider a node that classifies a 2-dimensional input vector into two classes A and B. Let its nonlinearity element be a simple threshold function.

The equation for the output node becomes:

$$o = \begin{cases} 1 \rightarrow \textit{class A if } w_1 x_1 + w_2 x_2 > \theta \\ -1 \rightarrow \textit{class B otherwise} \end{cases} \qquad (2.2)$$

where $\theta$ is the threshold of the output node. This node separates the space spanned by the two inputs into two regions. This division is done by a hyperplane, in this case a line whose equation is :

$$\theta = w_1 x_1 + w_2 x_2. \qquad (2.3)$$

Now, it remains to find the values of the weights $w_1$ and $w_2$ and the threshold $\theta$ that define this line. A single output node can compute these values only if the classes A and B are linearly separable. Two classes are linearly separable if there exists a hyperplane to separate them [Ull73]. But, many classification problems are not involved with linearly separable classes.

For example the XOR problem, shown in Table 2.1 is a classification problem that involves non linearly separable classes:

Table 2.1: XOR Truth Table

| $x_1$ | $x_2$ | output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

There is no line that can separate points (0,0) and (1,1) from the points (0,1) and (1,0).

The single-node neural network is limited to only linearly separable pattern classification. In order to be able to classify nonlinearly separable patterns either multi-node networks or multi-layer feedforward neural networks are used.

## 2.3 Single-Node Network Learning Algorithms

The iterative supervised learning algorithms described in the following sections attempt to reduce the output error for the current training pattern with minimal disturbance to patterns already learned. There is one layer of connections from the input nodes to the output node to be trained.

These algorithms could be grouped into one of the following two groups [Wid90]. Error-correction rules change the weights of a network with the objective of reducing error in the present input training pattern. Gradient rules update the weights during pattern presentations by gradient descent with the objective of reducing mean square error (MSE) in the output averaged over all training patterns.

### 2.3.1 The Gradient Descent Method

One of the basic mathematical tools to find the minimum of a function is the method of gradient descent. Let $X = (x_1, x_2, \ldots x_n)$, and $f(X)$.

The gradient of $f(X)$ is :

$$\nabla f(X) = [\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \ldots . \frac{\partial f}{\partial x_n}]. \qquad (2.4)$$

The basis of the gradient descent method could be described in the following way.

**Proposition:** If f(X) is a scalar function and $\nabla f \neq 0$, then $\nabla f(X)$ at $X = X_0$ gives the direction of maximum increase of f(X) at $X_0$ .

Basically, the gradient shows the direction of maximum ascent of the function. Therefore, the negative of the gradient defines the direction of minimum descent. Knowing only the gradient of the function, a reliable approach for finding the minimum of a function would be to follow the direction of the negative gradient, the steepest descent.

2.3.2 Learning Based on Gradient Descent

Usually, the objective of adaptation is to reduce error averaged over all the patterns in the training set. A common error function is mean square error (MSE). A popular approach to MSE reduction is by the method of steepest descent. Training a network by this method starts with an initial value for the weight vector, $W_0$. The gradient of the MSE function is measured. Then, the weight vector is changed in the direction opposite of the measured gradient. This procedure is repeated until the weight vector approaches a locally optimal value.

Updating the weight vector by steepest descent can be described by:

$$W_{k+1} = W_k + \mu(-\nabla_k) \qquad (2.5)$$

where k is the learning cycle, $\mu$ is a parameter that controls stability and rate of convergence, and $\nabla_k$ is the value of the gradient at a point on the MSE surface corresponding to $W_k$ [Wid90].

### 2.3.3 Widrow-Hoff Delta Rule

This algorithm attempts to minimize the MSE surface in the weight space. The connection weights connecting the n input nodes to the output node are represented as a vector $W = [w_1\ w_2\ \ldots . w_{n+1}]$. There is a special connection between a constant input $x_{n+1} = 1$ and the output node. The weight on this connection, $w_{n+1}$, simulates the threshold level of the output node. By changing this weight, $w_{n+1}$, the threshold level of the output node is changed.

The training set consist of pattern pairs $\{X_k, t_k\}$, k=1, 2,...p (p = size of training set), of sample pattern vectors $X_k = [x_1 x_2 \ldots . x_n]_k$ and their corresponding desired outputs $t_k$. At each iteration of the learning (training) cycle one of the input-output training pairs, $(X_k, t_k)$, is presented to the network. If the actual output computed for this pattern $(X_k)$ is different from the target output, $t_k$, the weights are updated using a small learning rate $\mu$. One reason $\mu$ is kept small is to insure that the weight updates are not geared to any particular pattern. The

whole training set is presented repeatedly (k = 1,2,....,p) in the same order.

At the $k^{th}$ iteration, let the weight vector be $W_k$ and the input vector be $X_k$. Then, the linear error is the difference between the desired output and the actual linear output:

$$e_k = t_k - W_k^T X_k. \qquad (2.6)$$

This algorithm tries to minimize the mean of the square of the difference between the desired output and the actual linear output summed over all input/output training pattern

$$E = \sum_k (t_k - W_k^T X_k)^2 = \sum_k e_k^2. \qquad (2.7)$$

pairs [Wid90]:

The gradient of the MSE at $W_k$ is:

$$\nabla_k = \begin{bmatrix} \frac{\partial E(e_k^2)}{\partial w_{1k}} \\ \vdots \\ \frac{\partial E(e_k^2)}{\partial w_{(n+1)k}} \end{bmatrix}. \qquad (2.8)$$

The above gradient is approximated by:

$$\hat{\nabla}_k = \frac{\partial e_k^2}{\partial W_k} = 2e_k \frac{\partial e_k}{\partial W_k} = -2e_k X_k. \qquad (2.9)$$

Updating the weight vector by steepest descent on the MSE surface gives:

$$\begin{aligned} W_{k+1} &= W_k + \mu(-\hat{\nabla}_k) \\ &= W_k + 2\mu e_k X_k. \end{aligned} \qquad (2.10)$$

The weight update equation for the original form of the delta rule is :

$$W_{k+1} = W_k + \mu e_k \frac{X_k}{|X_k|^2} \qquad \text{where } 0<\mu<1. \quad (2.11)$$

In other words, weights are corrected at each iteration by an amount which is proportional to the difference between the target output and the actual output. The choice of $\mu$ controls stability and speed of convergence.

This learning rule makes error corrections proportional to the error itself:

$$\begin{aligned} e_k &= t_k - W_k{}^T X_k \\ \Delta e_k &= \Delta(t_k - W_k{}^T X_k) = -X_k{}^T \Delta W_k \\ \Delta W_k &= W_{k+1} - W_k = (W_k + \mu \frac{e_k X_k}{|X_k|^2}) - W_k \\ &= \mu \frac{e_k X_k}{|X_k|^2} \\ \therefore \Delta e_k &= -\mu \frac{e_k X_k{}^T X_k}{|X_k|^2} = -\mu e_k. \end{aligned} \qquad (2.12)$$

This learning algorithm can generalize from training patterns (patterns used during learning). Patterns similar to a training sample are classified into the same class as that of the training sample. This rule converges to the least squares solution. But, in some cases it may fail to separate training patterns that are linearly separable.

2.3.4 Perceptron Learning Rule

This rule uses a binary state node, o = {+1,-1}. It alters the weights to correct error in the output in response to the present input pattern. In contrasts to delta rule which is a linear rule that makes error corrections proportional to the linear error, perceptron learning rule is a nonlinear rule.

This rule uses the nonlinear error which is the difference between the desired response and the actual nonlinear output at the presentation of training pattern pair { $X_k$, $t_k$ }:

$$e_k = t_k - o_k$$

$$where\ o_k = \begin{cases} +1 & if \sum_{i=1}^{n+1} w_i x_i > 0 \\ -1 & otherwise. \end{cases} \qquad (2.13)$$

The possible values for each $e_k$ are {0,2,-2}. The weight update equation for the perceptron rule is, then:

$$W_{k+1} = W_k + \mu e_k X_k. \qquad (2.14)$$

In contrast to the delta rule which sometimes may fail to separate linearly separable training sets, the perceptron rule is capable of separating linearly separable sets. The following example is a case where the perceptron rule succeeds in separating the patterens and the delta rule fails. Suppose we have a one-dimensional input vector. The following three input pattern pairs are to be learned {x = 2, t = 1}, {x = 1, t = -1}, and {x = -1, t = -1}. We would like to separate the first input (x = 2) from the other two inputs (x = 1, x = -1 ) by assigning an output of t = 1 for the first input and t = -1 for the others. The delta rule with its linear output function, $t = w_1x + w_2$ , can not find any $w_1$ and $w_2$ such that the desired t values are obtained for the above 3 input values. However, the perceptron rule where the output function is:

$$t = \begin{cases} 1 & \text{if } w_1x + w_2 > 0 \\ -1 & \text{otherwise} \end{cases} \qquad (2.15)$$

can find values for $w_1$ and $w_2$ (e.g.$w_1 = 2$, $w_2 = -2$) such that the desired t values are obtained for the given three inputs. The perceptron rule can separate lineraly separable sets, but if the patterns are not linearly separable this algorithm goes on forever unable to find a solution that separates the patterns.

2.3.5 Generalized Delta Rule

Nodes with sigmoidal nonlinearity elements are used for this learning rule. These nonlinearities have

saturation points helpful for decision making, and they have differentiable characteristics that help learning. As an example of such a function is the sigmoid function (sketched in Figure 1.1b):

$$o = f(v) = 1 / (1 + \exp^{-v})$$
$$\text{where } v = \sum_{i=1}^{n+1} w_i x_i. \qquad (2.16)$$

The error measure to be minimized is:

$$E = \sum_{k=1}^{p} (t_k - o_k)^2. \qquad (2.17)$$

As of with the delta rule, the method of gradient descent is used in minimizing the above error function. At the $k^{th}$ iteration the error is :

$$e_k = t_k - o_k = t_k - f(v_k). \qquad (2.18)$$

The approximate gradient $\nabla_k$ of the MSE at $W_k$ is:

$$\hat{\nabla}_k = \frac{\partial e_k^2}{\partial W_k} = 2e_k \frac{\partial e_k}{\partial W_k}$$

*where*

$$\frac{\partial e_k}{\partial W_k} = - \frac{\partial f(v_k)}{\partial W_k} = - f'(v_k) \frac{\partial v_k}{\partial W_k} \qquad (2.19)$$

*and*

$$\frac{\partial v_k}{\partial W_k} = X_k.$$

Then, the weight update equation for this learning rule becomes [Wid90]:

$$\begin{aligned} W_{k+1} &= W_k + \mu(-\hat{\nabla}_k) \\ &= W_k + 2\mu(t_k - o_k) f'(v_k) X_k. \end{aligned} \tag{2.20}$$

The generalized delta rule reduces to the delta rule by reducing the sigmoid function of the output node to the linear function ( $o = W^T X$).

## 2.4 Capacity of Single-Node Network

The average number of random patterns that a single-node network (Figure 2.1) can learn to correctly classify is approximately twice the number of connection weights in the network (p= 2w) [Wid90]. It is found that the probability of a set of random input patterns being linearly separable by a single-node network is a function of the number of patterns in the set, p, and the number of weights in the network, w:

$$P = \begin{cases} 2^{-(p-1)} \sum_{i=0}^{w-1} \binom{p-1}{i} & \text{for } p > w \\ 1 & \text{for } p \leq w. \end{cases} \tag{2.21}$$

The above result applies to randomly selected patterns. In real problems the patterns are not random. They have some regularities which support generalization. In a practical problem the number of patterns that a

single-node network can learn is far more than the capacity mentioned above.

## 2.5 Multi-node Neural Network

Multi-node neural networks are networks consisting of an input layer, multiple output type nodes at the second layer, and various fixed logic devices such as AND and OR in the third layer. The learning rule used to train the weights connecting the input layer to the second layer are those of the single-node network learning algorithms, namely; the perceptron rule; the delta rule; and the generalized delta rule. Later in this dissertation (chapter 4) a recent algorithm [Kak94], is discussed that can be used for training the weights in these networks.

Multi-node neural nets are capable of separating nonlinear separable sets, while a single-node network is limited to only linearly separable sets. Also, the average number of random patterns that a multi-node network can learn to correctly classify is approximately twice the number of connection weights in the network.

## 2.6 Multi-Layer Feedforward Neural Nets

A multi-layer feedforward neural network is a structured hierarchical layered network. It consists of an input layer, an output layer, and one or more layers of (hidden) nodes separating the input and the output layers [Lip87], [Hay94]. Unlike the multi-node networks where only the connections between the input layer and second layer

are adaptive, all the connections between all the layers in a multi-layer network are usually adaptive. The nodes in the hidden layers create intermediate representation of the input useful for solving recognition tasks. Figure 2.2 shows a typical feedforward neural network with one layer of hidden nodes. Usually, the number of nodes in the output layer (m) corresponds to the (m) different number of classes to be separated.

Usually, each node in one layer is connected to all the nodes in adjacent layers. Each connection between two nodes at different levels is associated with a weight which measures the degree of interaction between them.

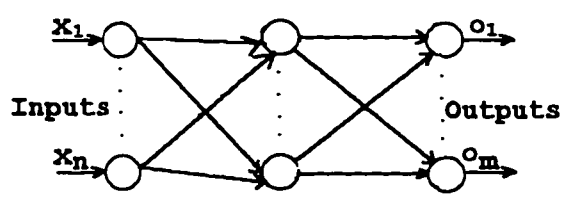


Figure 2.2 A typical 3-layer feedforward network with one layer of hidden nodes.

Generally, each layer receives input only from its previous layer and produces output to be fed as input to its next

higher layer. Information flows in one direction from the input layer to the output layer.

The next section describes the most popular algorithm for training multi-layer feedforward networks, the backpropagation algorithm.

## 2.7 The Backpropagation Algorithm

The backpropagation algorithm is a supervised learning algorithm [Rum86] [Vee95] for feedforward neural networks. Input / output pattern pairs which are to be learned (training patterns) are supplied to the network during training. The goal is to adjust the weights of connections between the nodes so as to obtain the desired outputs given the input training patterns.

The total input to node j at a hidden layer or output layer upon the presentation of input training pattern k is:

$$a_{kj} = \sum_i w_{ji} o_{ki} \qquad (2.22)$$

where $o_{ki}$ is the output of node i from previous layer which is fed as input to node j upon presentation of pattern k, and $w_{ji}$ is the weight of the connection between nodes i and j at adjacent layers. The total input to each node j at a hidden layer or output layer passes through a sigmoid function $f_j$ to produce the output of node j at the $k^{th}$ presentation :

$$o_{kj} = f_j(a_{kj}) . \qquad (2.23)$$

Let,

$$E_k = \frac{1}{2}\sum_j (t_{kj} - o_{kj})^2 \qquad (2.24)$$

be our error measure upon the presentation of input pattern k with $t_{kj}$, $o_{kj}$ being the target output and the actual output for output node j at the $k^{th}$ presentation, respectively. The goal of this algorithm is to minimize the overall measure of error which is the sum of the errors of all output units upon the presentation of all input training patterns:

$$E = \sum_{k=1}^{p} E_k. \qquad (2.25)$$

To achieve this goal, upon the presentation of pattern k the gradient vector of the error measure $\nabla E_k$ which is a function of all free parameters (weights) is computed. Initially the weights on the connections between the nodes are randomly chosen to small values. The reason for choosing the initial weights small is to keep the output of the nodes away from its extreme values in which case the convergence could be extremely slow. Then, the weights are updated in the direction of $-\nabla E_k$, as :

$$W_{k+1} = W_k - \mu \nabla E_k \qquad (2.26)$$

where $W_k$ is the weight vector at the $k^{th}$ iteration.

To compute the elements of the gradient vector $\nabla E_k$ at the $k^{th}$ iteration (presentation of an input pattern), we notice that:

$$\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial a_{kj}} \frac{\partial a_{kj}}{\partial w_{ji}} \qquad (2.27)$$

where $\partial a_{kj}$ is the change in the input of node j at the $k^{th}$ iteration, $\partial w_{ji}$ is the change in the weight between node i and node j. By differentiating Eq.(2.22) with respect to $w_{ji}$, We have that :

$$\frac{\partial a_{kj}}{\partial w_{ji}} = o_{ki}. \qquad (2.28)$$

Rumelhart has defined:

$$e_{kj} = -\frac{\partial E_k}{\partial a_{kj}} \qquad (2.29)$$

to be the error signal at the $k^{th}$ iteration for each node in the hidden or output layer. To compute $e_{kj}$, one can write:

$$e_{kj} = -\frac{\partial E_k}{\partial a_{kj}} = -\frac{\partial E_k}{\partial o_{kj}} \frac{\partial o_{kj}}{\partial a_{kj}} \qquad (2.30)$$

where from Eq.(2.23):

$$\frac{\partial o_{kj}}{\partial a_{kj}} = f'_j(a_{kj}). \qquad (2.31)$$

By differentiating Eq.(2.24) with respect to $o_{kj}$, we have:

$$\frac{\partial E_k}{\partial o_{kj}} = -(t_{kj} - o_{kj}). \qquad (2.32)$$

Hence, at the $k^{th}$ iteration the error signal for a node in the output layer becomes :

$$e_{kj} = (t_{kj} - o_{kj}) f'_j(a_{kj}). \qquad (2.33)$$

At the $k^{th}$ iteration the error signal for any node in the hidden layer is shown [Rum86] to be :

$$e_{kj} = f'_j(a_{kj}) \sum_i e_{ki} w_{ij} \qquad (2.34)$$

where $e_{ki}$ s are the error signals of nodes at the output layer to which hidden node j gives input.

Finally, during the $k^{th}$ iteration the weights are adjusted as :

$$\Delta w_{ji} = \mu e_{kj} o_{ki} \qquad (2.35)$$

where $o_{ki}$ is the output of the $i^{th}$ node which is the input to the $j^{th}$ node at the next higher layer ( if $i^{th}$ node is in the input layer its output is equal to its input). Also, $e_{kj}$ is the error signal of the $j^{th}$ node at the next higher layer, and $\mu$ is the learning rate which controls stability and speed of convergence ( $0 < \mu < 1$ ).

### 2.7.1 Analysis of Rate of Convergence of Backpropagation

In an study [Ana95], it has been shown that even though each iteration of backpropagation reduces the

difference between the actual and the desired output vectors, in the first few iterations the difference between some of the elements of these vectors actually increases. This results in small changes in weight values which neccessitates a large number of iterations. This study concludes that as the number of classes increases the speed of convergence decreases.

Consider a three layer network designed to classify input patterns into one of m classes. There are m output nodes in the output layer, each node corresponds to one class. Let us denote a weight vector connecting the input layer to a node in the hidden layer by W, and denote a weight vector connecting the hidden layer to a node in the output layer by V. The error associated with the $s^{th}$ output node upon presentation of the $k^{th}$ pattern of class i is:

$$e_s^{(k,i)} = t_s^{(k,i)} - o_s^{(k,i)} \qquad (2.36)$$

where $t_s^{(k,i)}$ and $o_s^{(k,i)}$ are the target output and the actual output of the $s^{th}$ output node, respectively. The square error associated with the $s^{th}$ output node for all patterns of class i is:

$$E_{(i,s)} = \sum_k (e_s^{(k,i)})^2. \qquad (2.37)$$

The total square error associated with the $s^{th}$ output node

upon presentation of patterns of all classes would be :

$$E_{(s)} = E_{(s,s)} + \sum_{i \neq s} E_{(i,s)} \qquad (2.38)$$

$$\Delta v_{s,r}^{(k,i)} = \mu((t_s^{(k,i)} - o_s^{(k,i)})\, o_s^{(k,i)} (1 - o_s^{(k,i)}))\,(h_r^{(k,i)}) . \qquad 2.39$$

The weight change on the connection between the $r^{th}$ hidden node and the $s^{th}$ output node upon the presentation of the $k^{th}$ pattern of class i is a product of the error signal for the $s^{th}$ output node and the output of the $r^{th}$ hidden node:
where $h_r^{(k,i)}$ is the output of the $r^{th}$ hidden node upon the presentation of the $k^{th}$ pattern of class i. Also, the weight change on the connection line between the $l^{th}$ input node and $r^{th}$ hidden node due to the $(k,i)^{th}$ sample is :

$$\Delta w_{r,l}^{(k,i)} = \mu(h_r^{(k,i)})\,(1 - h_r^{(k,i)})\,(x_l^{(k,i)}) \sum_{j=1}^{m} ((t_j^{(k,i)} - o_j^{(k,i)})\, o_j^{(k,i)} (1 - o_j^{(k,i)})\, v_{j,r} \qquad (2.40)$$

where $x_l^{(k,i)}$ is the input from the $l^{th}$ node at the input layer at the presentation of the $k^{th}$ pattern of class i.

*Proposition1:* In any iteration of backpropagation, the weight change between any hidden node and the $s^{th}$ output node is positive upon presentation of any sample from class s, $(\Delta v_{s,r}^{(k,i)} > 0)$, and is negative upon presentation of any sample from class i where $i \neq s$, $(\Delta v_{s,r}^{(k,i)} < 0)$ [Ana95].

*Proof:* In the equation for the weight change on the connection between a hidden node and an output node every term is positive except for ( $t_s^{(k,i)} - o_s^{(k,i)}$ ). The sign of this term is positive for all patterns of class s and is negative otherwise.

*Proposition 2:* In the first few iterations of backpropagation, the change of weight in a link between a hidden node and an output node is expected to be negative for multiclass problems upon presentation of all patterns [Ana95].

*Proof:* Each output node is intended to recognize the patterns of one class, only. The change in weight between the $r^{th}$ hidden node and the $s^{th}$ output node due to the presentation of all patterns is :

$$\Delta v_{s,r} = \Delta v_{s,r}^{(s)} + \sum_{i \neq s} \Delta v_{s,r}^{(i)}. \qquad (2.41)$$

The first term on the right hand side of the above equation is the change in weight due to the presentation of patterns of class s only, and the second term is the change in weight due to the presentation of patterns of classes other than s ( $\forall i \neq s$ ). The change in weight due to patterns of class s is positive:

$$\Delta v_{s,r}^{(s)} = \sum_{k} \Delta v_{s,r}^{(k,s)} > 0. \qquad (2.42)$$

The change in weight due to all other patterns is negative :

$$\Delta v_{s,r}^{(i)} = \sum_{k} \Delta v_{s,r}^{(k,i)} < 0 \qquad (2.43)$$

where $i \neq s$. For output node s, when the number of classes m>2, the number of samples in classes other than s are about m-1 times more than the number of samples in class s. Hence the number of negative weight changes above is more than the number of positive weight changes. Since the weights in the network are initialized to small random values, the magnitude of most of the weight changes are small for the first few iterations. The small values of weight changes along with the more negative summands in the equation for $\Delta v_{s,r}$ will make the expected value for $\Delta v_{s,r}$ negative in the first few iterations. The mathematical derivations are shown in [Ana95].

*Proposition 3* : In the first few iterations of backpropagation on a network with one hidden layer, the change in each weight on a link between the input layer and the hidden layer is expected to be negative upon presentation of all the training samples.

*Proof:* It is shown [Ana95] that the expected change of each weight connecting an input node and a hidden node is:

$$\Delta w_{r,l}^{(k,i)} \approx -\mu \frac{S}{48} ((h_r^{(k,i)})^2 (1-h_r^{(k,i)}) (x_l^{(k,i)}) . \qquad (2.44)$$

since every term in the above equation is positive, the minus sign makes the expression negative. The sum of these expressions over all training samples :

$$\Delta w_{r,l} = \sum_{(k,i)} \Delta w_{r,l}^{(k,i)} \qquad (2.45)$$

will also be negative.

**Conclusion:** In brief, the above results show that all weights decrease after the first few iterations. This causes the outputs of the hidden nodes (h) to decrease since they are monotonic functions of the first layer weights (W). Outputs of the output nodes decrease as they are monotonic functions of the second layer weights (V) and the hidden node outputs (h). For a training sample of class s the error $E_{(s,s)}$ increases but $E_{(i,s)}$ , ( $\forall i \neq s$ ) decreases. The increase in $E_{(s,s)}$ causes the magnitude of $\Delta v_{s,r}^{(s)}$ to increase in later iterations. Conversely, the decrease in $E_{(i,s)}$ , ( $\forall i \neq s$ ), causes the magnitude of each $\Delta v_{s,r}^{(i)}$, ( $\forall i \neq s$ ) to decrease. These small positive and negative contributions almost cancel each other resulting in very small changes in V weights which in turn necessitates a very large number of iterations in order to converge. Finally, one can conclude that as the number of classes increases the speed of convergence decreases even more.

### 2.7.2 Deficiencies of the Backpropagation Algorithm

The deficiencies of the backpropagation algorithm are summarized in the followings:

- The algorithm is not guaranteed to converge to a solution even if one exists.

- The training procedure is very slow. The training time increases in a faster rate than the size of the network. On the one hand the amount of the computation needed to update the weights upon the presentation of a training sample is proportional to the number of weights in the network, and on the other hand the bigger the size of the network is the more training patterns are required for it.

- Given a problem, there is no procedure to guide one to a specific architecture (number of layers and number of nodes per layer). The number of hidden nodes for a three-layer network must be guessed. There are studies [Mir89], [Geo91] that help us in choosing this number. They show that in a n-dimensional space, the maximum number of regions r that are linearly separable using h hidden nodes is:

$$r(h,n) = \sum_{i=0}^{n} \binom{h}{i} \qquad (2.46)$$

*where*

$$\binom{h}{i} = 0, \quad h < i.$$

One would like to place at least one training pattern in each of the r regions in order to separate the r regions. Hence, the minimum number of training patterns needed to train a three-layer network by

backpropagation is equal to the number of linearly separable regions in the input space (p >= r). Having p number of training patterns and replacing it for r in equation 2.46, one can estimate the necessary number of hidden nodes. Although this study sheds some light on the size of the network, it fails to give accurate results. For classification problems the shapes of the regions are not known. If the number of hidden nodes,h, is picked too small the training patterns would not be separated, and if it is picked too large the generalization would degrade [Ree93].

- The architecture of the network is fixed prior to the learning. Any addition/deletion of nodes requires the computations to be redone.

# CHAPTER 3

# A MODULAR APPROACH FOR SOLVING CLASSIFICATION PROBLEMS

## 3.1 Introduction

A popular approach for multi-class classification tasks is a monolithic approach. In this approach a network is designed and trained to separate the training samples of the different classes. The training algorithm used is the backpropagation algorithm.

In this chapter, an attempt is made to design a modular network for a classification task. In this approach a m-class classification task is reduced to m 2-class sub-tasks where each sub-task is independently solved by a sub-network. An algorithm is presented for building and training each sub-network using any of the single-node network learning rules such as the perceptron rule [Wid90]. Our aim is to overcome some of the deficiencies of the backpropagation algorithm [Rum86]; in particular the problems of slow rate of convergence and of fixed architecture.

## 3.2 Goals Behind Proposed Algorithm

The goals behind the proposed algorithm are:

- To improve the rate of learning compared to that of the non-modular network trained by the backpropagation algorithm. By breaking a classification task into sub-tasks such that each sub-task is independently run

on a sub-network, it is expected that each sub-network is trained faster than the comparable non-modular network trained by backpropagation.

- To have a constructive method to guide us in determining the number of hidden nodes needed in the network as opposed to the backpropagation algorithm in which one has to guess.
- To have a structure that develops as it learns as opposed to that of backpropagation where the architecture of the network is fixed prior to learning and any addition or deletion of nodes requires retraining.

## 3.3 Proposed Algorithm

Given a set of m disjoint classes S = { $C_1$, $C_2$, ....., $C_m$ } the problem is to separate each class $C_i$ ( where $C_i \in S$ ) from all other classes in the set S, S - {$C_i$}. The following algorithm should be run for each class $C_i$ independently.

Algorithm

- For a class $C_i$, order the other classes and assign them to set A, A = S - {$C_i$} = { $C_{a1}$, $C_{a2}$, ....., $C_{al}$ } where $C_{ai} \in S$ & $C_{ai} \neq C_i$.
- Linear - Sep($C_i$, A) ; Linearly separate class $C_i$ from the classes in set A. Set a maximum number of iterations allowed for this attempt.
- If $C_i$ is not linearly separable from A, then

- If set A has two or more classes, then recursively
  - Break recursively the set A into two subsets $A_1$ and $A_2$ where

    $A_1 = \{ C_{a1}, C_{a2}, \ldots\ldots, C_{a\lfloor 1/2 \rfloor} \}$ and

    $A_2 = \{ C_{a\lfloor 1/2 \rfloor+1}, C_{a\lfloor 1/2 \rfloor+2}, \ldots\ldots, C_{a1} \}$
    - Linear - Sep($C_i$, $A_1$)

      [node ( $i/a_1, \ldots., a_{\lfloor 1/2 \rfloor}$ ) is created ]
    - Linear - Sep($C_i$, $A_2$)

      [node ( $i/a_{\lfloor 1/2 \rfloor+1}, \ldots., a_1$ ) is created]
    - $(i/a_1, \ldots., a_1) = ( i/a_1, \ldots., a_{\lfloor 1/2 \rfloor} ) \wedge ( i/a_{\lfloor 1/2 \rfloor+1}, \ldots., a_1 )$
  - Else, set A has one class only, $A = \{C_a\}$.

    consider the training samples of this class, and order them; $C_a = \{t_1, \ldots. t_1\}$.

    Break the training samples in $C_a$ into two subsets $C_{a1}$, and $C_{a2}$ where;

    $C_{a1} = \{t_1, \ldots, t_{\lfloor 1/2 \rfloor}\}$, $C_{a2} = \{t_{\lfloor 1/2 \rfloor+1}, \ldots, t_1\}$
    - Linear-Sep($C_i$, $C_{a1}$)

      [node ($i/a_1$) is created]
    - Linear-Sep($C_i$, $C_{a2}$)

      [node ($i/a_2$) is created]
    - $(i/a) = (i/a_1) \vee (i/a_2)$
- Else, $C_i$ is linearly separable from the classes in set A. Create a node called, ($i/a_1$, ..... $a_1$).

Training samples of one class is always linearly separable from one training sample of another class. In doing so, one can use either the perceptron rule or one of the corner classification algorithms discussed in chapter-4.

### 3.3.1 Alternative to the Proposed Algorithm

Given a set of m disjoint classes $S=\{C_1, C_2, ...., C_m\}$ such that the classes are ordered in the set, separate each class $C_i$ from the set of classes numbered higher than itself $\{C_{i+1}, C_{i+2}, ...., C_m\}$. The algorithm introduced in the previous section is used to create the necessary nodes for this separation. The classes numbered lower than this class are already separated from $C_i$. Then, an AND gate is used to separate $C_i$ from $S - \{C_i\}$ as shown in Figure 3.1.

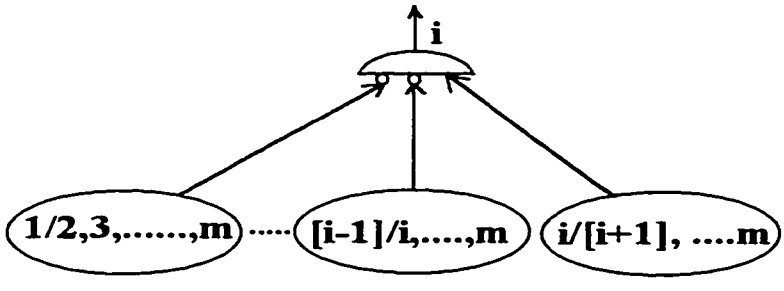


Figure 3.1 An alternative way to separate $C_i$ from $S-\{C_i\}$.

This alternative approach creates less number of nodes in the network, but it has the drawback that it either correctly assigns an input pattern to a class or it misclassifies it. It never rejects the input due to ambiguity.

### 3.3.2 Example

The following example shows the steps of our algorithm to classify the 4-class classification task of Figure 3.2. The problem is to classify the points in a two-dimensional space into one of the 4 classes.

Our proposed approach breaks this 4-class classification task into four 2-class sub-tasks each with its own network 1/2,3,4 , 2/1,3,4 , 3/1,2,4 , and 4/1,2,3.

Class 1 is linearly separable from all other classes. Hyperplane 1/2,3,4 in Figure 3.2 separates this class from all other classes. Hence, the sub-network of 1/2,3,4 has only one node as shown in Figure 3.3.

Class 2 is not linearly separable from all other classes. An attempt is made to linearly separate this class from class 1 by one node 2/1, and also linearly separate it from classes 3 and 4 by another node 2/3,4. To train node 2/1 pattern samples corresponding to classes 2 and 1 are used, only. The outputs of the nodes 2/1 and 2/3,4 are fed as inputs to an AND gate to create the sub-network of 2/1,3,4.

Classes 3 and 4 are not linearly separable from each other. A number of nodes are created to separate training samples of class 3 from different sub-sets of training patterns of class 4. The outputs of these nodes are fed to an OR gate to separate all training samples of class 3 from those of class 4. A layer of AND gates is used at the last layer so that an input pattern is assigned to at most one class. It should be mentioned that only the connections between the input nodes and the created nodes are trainable.

In the alternative network structure each class is linearly separated from another class by only one node. For example for the 4 class problem, as Figure 3.4 shows, a node is created to separate class 2 from classes 3 and 4, only. Class 2 has already been separated from class 1 by the node 1/2,3,4. Class 4 has been separated from all the other classes by the nodes created; hence, no nodes needs to be created for it. This way the number of nodes created are less than the number of nodes created by the first approach in which each class is separated from all other classes. Less nodes to be created means less time is spent during learning. But as the simulation results show, the generalization capability of the first network structure,

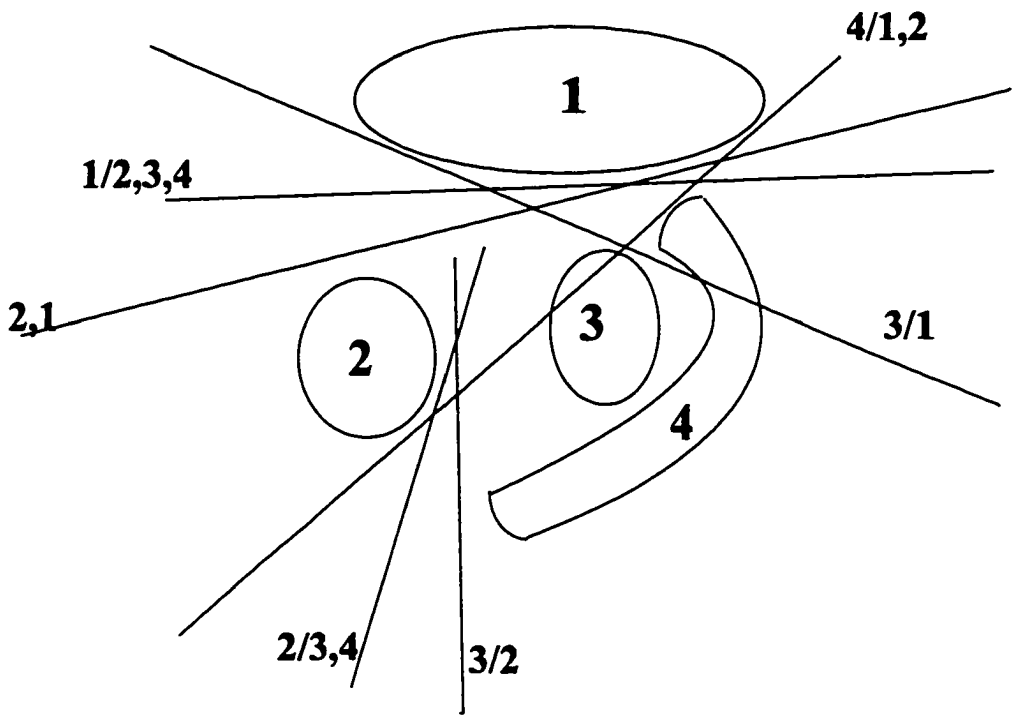


Figure 3.2 Two-dimensional input space divided into four classes.

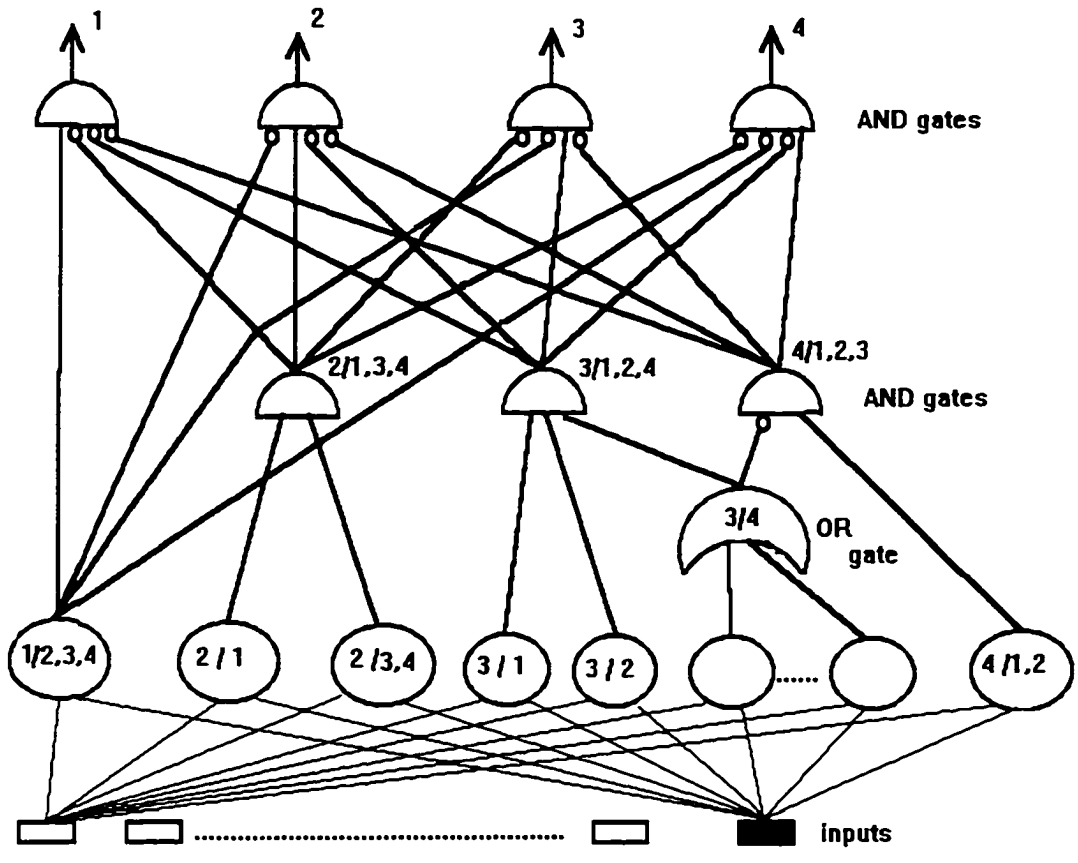


Figure 3.3 The proposed network structure for the 4 class example.

Figure 3.3, is somewhat better than the generalization capability of the second network structure Figure 3.4.

### 3.3.3 Analysis of the Algorithm

The complexity of a neural network can be measured by the number of mapping that are possible in the network. For a network with n binary input nodes, the number of possible binary patterns is $2^n$. A number of these possible pattern samples $p <= 2^n$ is usually chosen to train the network. If we design a single non-modular network to assign these p training patterns into m different classes, then the number of possible mapping to be considered by our network would be:

$$ {}_m\Pi_p = m^p \qquad \text{where } p \leq 2^n. \qquad (3.1) $$

On the other hand, if the task is decomposed into m 2-class sub-tasks each handled by a sub-network then the number of possible mapping to be considered by each sub-network would be:

$$ {}_2\Pi_p = 2^p \qquad \text{where } p \leq 2^n. \qquad (3.2) $$

These sub-networks run in parallel. Hence, for multi-class classification tasks where $m \geq 2$ the modular scheme would be faster. Our algorithm guarantees the separation of training samples of one class from the training samples of all the other classes. In doing so, our algorithm keeps

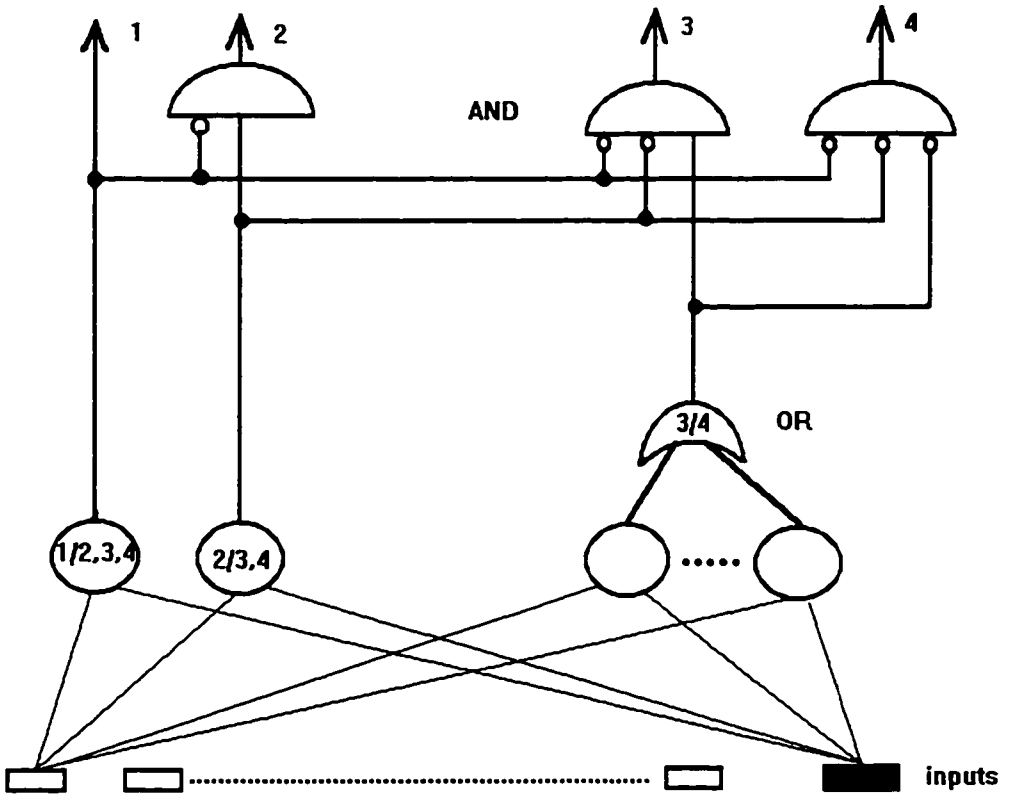


Figure 3.4 An alternative network structure for the 4 class problem.

breaking the problem into smaller ones. This breaking if necessary goes on until samples of the class are separated from only one sample of another class, and we know from the convergence of the perceptron [Ros73] that this can be done.

A network designed according to our modular approach is anticipated to learn faster than a comparable non-modular network trained by the backpropagation algorithm in the following ways:

1) The sub-networks of our modular network can be trained independently and in parallel.

2) Each sub-network of our modular network is smaller than the comparable non-modular network trained by backpropagation. As a result, the number of weights updated in training each sub-network is anticipated to be less than the number of weights updated to train the non-modular network of backpropagation:

$$\sum_{j=1}^{N} p_j i_j w_j < piw. \qquad (3.3)$$

The right hand side of the above inequality shows the number of weights updated for the non-modular network trained by backpropagation. This number is the product of the following three numbers; the number of samples in the training set (p);

the number of presentations (iterations, i) of this training set; and the number of weights in the network (w).

The number of weights updated for each sub-network of our modular network is represented in the left hand side of the above inequality. For each sub-network a number of nodes are attempted to be created but some may have failed to be created. Assume the sum of all these nodes be N for a sub-network. Let $p_j$ be the number of samples in the training set presented to the $j^{th}$ node of the sub-network, $j \in \{1,2,...N\}$. And, $i_j$ is the number of presentations (iterations) of this training set to the network. Finally, $w_j$ is the number of weights from the inputs to the $j^{th}$ node.

Our modular approach is adaptable; adding a new class adds new nodes to the existing network. For example, if class 5 were to be added to the 4-class problem of Figure 3.2, the sub-networks created would remain. In addition, more nodes would be created for each sub-network to separate class 5 from each of the existing 4 classes.

## 3.4 Handwritten Digit Recognition

The performance of the proposed algorithm is investigated by its application to a handwritten digit

recognition task. In this experiment two different size data bases are used.

### 3.4.1 Data Bases

The first data base (DB1) consists of 500 digits written by 3 different people. Each digit has 50 samples. Each digit was created using Bitmap software supported by Unix on a Sun workstation. Each digit was coded as a 8 * 8 binary pattern (+1,-1). The data base was randomly partitioned into a training set of 400 samples and a test set of 100 samples.

The second database (DB2) consists of 900 digits written by 4 different people, 500 digits of which are the same as the first database. Each digit has 90 samples, and was created using the same software as the first database. The second data base was partitioned into a randomly picked training set of 700 samples, and a test set of 200 samples.

Some digit samples used are shown in Figure 3.5(a) along with an example for two sample digits 3.5(b). The digit samples shown in the Figure 3.5(a) and the Figure 3.5(b) are some of the hand written digits that are generated for this purpose.

### 3.4.2 Learning Rule

Each hidden node of our network which separates training samples of a class from the training samples of other classes is trained using the perceptron learning

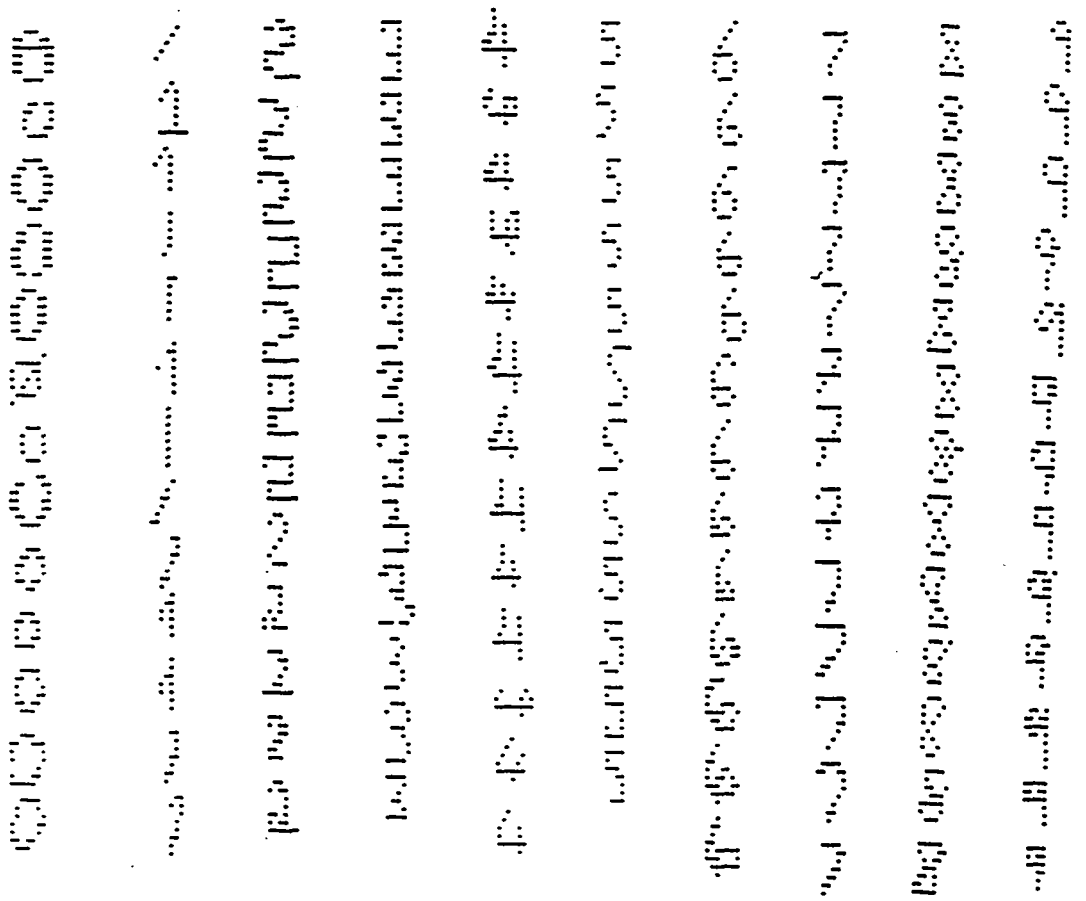

Figure 3.5(a) Hand written Numeral Samples.

```
..oooo..        ..o.....
..o..o..        ..o.....
.....o..        ..o.o...
.....o..        .oo.o...
...ooo..        oo..o...
.....o..        ooooooo.
.....o..        ....o...
...ooo..        ....o...
```

Figure 3.5(b) Two 8x8 grids (64-input vectors) for two digits.

rule. In other words the weights connecting input lines to each hidden node are updated as:

$$W_{k+1} = W_k + \Delta W_k$$

$$\textit{Where}$$

$$\Delta W_k = \mu (t_k - o_k) X_k \qquad (3.4)$$

$$\textit{and}$$

$$o_k = \begin{cases} +1 & \textit{if } W_k^T X_k > 0 \\ -1 & \textit{otherwise} \end{cases}.$$

$W_k$ is the weight vector and $X_k$ is the input pattern vector at the $k^{th}$ iteration, and $t_k$, $o_k$ are the target and actual outputs, respectively. This learning rule has the advantage that it uses nodes with simple threshold function. From the hardware implementation point of view, the structure of the analog sigmoidal node is more complex than the simple threshold one. For this reason many researchers have designed training approaches using simple threshold nodes [Yu94], [Zha94], and [Goo94]. Studies [Roy95] show that if the perceptron algorithm is applied to linearly nonseparable input patterns, it can learn a large linearly separable subset of the given nonseparable training set, and identify nonseparable patterns in the training set.

### 3.4.3 Performance Measure and Assignment Criteria

To measure the performance of the classifier a set of test samples, samples not seen during training the network,

is supplied. Each such input sample is given to the trained network. Depending on its output result, this input is either classified as a well classified pattern, a misclassified pattern, or a rejected item. It is well classified if the input is assigned to the correct class only; it is misclassified if it is assigned to a wrong class only; and rejected otherwise. The rate of well classified $R_{wc}$, rejected $R_r$, and misclassified patterns $R_{mc}$ measure the generalization performance of a classifier. Let us assume that $G = G(R_{wc}, R_r, R_{mc})$. In order for G to measure the generalization capability of the classifier, it should satisfy the following constraints:

$$\begin{aligned} \frac{\partial G}{\partial R_{wc}} &> 0, \\ \frac{\partial G}{\partial R_r} &< 0, \\ \frac{\partial G}{\partial R_{mc}} &< 0. \end{aligned} \qquad (3.5)$$

For most applications $\partial G/\partial R_{wc}$, $\partial G/\partial R_r$, and $\partial G/\partial R_{mc}$ are considered constant [Cor95]. Therefore, G is expressed as a linear function of its three parameters:

$$G = C_{wc}R_{wc} - C_rR_r - C_{mc}R_{mc} \qquad (3.6)$$

where $C_{wc}$, $C_r$, $C_{mc}$ are the cost of well classification, rejection, and misclassification, respectively. For most applications misclassification degrades the generalization function G much more than rejection:

$$C_{mc} \gg C_r. \qquad (3.7)$$

We have assumed the following normalized function G for our simulation results:

$$G = R_{wc} - \frac{1}{10}R_r - R_{mc}. \qquad (3.8)$$

To assign a pattern to a class the following procedure is taken. A node in a network designed according to our algorithm separating class i from a set $J_i$ of one or more classes in the set S of all possible classes (S = 1, 2, ....., m) is represented as $(i/J_i)$ where $J_i = \{j;\ j \in S\}$. The binary output (+1,-1) of this node is represented as $v(i/J_i)$.The output node for each class is determined as:

- $v(i/S - i) = v(i/J_1) \wedge v(i/J_2) \wedge .... \wedge v(i/J_s)$ (3.9)

  where $J_1 \sqcup J_2 \sqcup .... \sqcup J_s = S - i$

  and $J_1, J_2, ....., J_s$ are disjoint.

The final decision of the network is then :

- if $v(i/S-i) = 1$

  and $v(j/S-j) = -1$

  for all $j \neq i$ & $j \in S$

  then, the input pattern is assigned to class i.
- Otherwise, the input pattern is rejected.

For the network designed according to the alternative to the proposed algorithm, the assignment of a pattern to a

class is done in the following way:

$$v(i/(i+1), \ldots\ldots, m) = v(i/J_1) \wedge \ldots\ldots \wedge v(i/J_s)$$

and

$$v(i/S-i) = \sim v(1/2, \ldots\ldots, m) \wedge \ldots\ldots \wedge \sim v((i-1)/i, \ldots\ldots, m) \wedge v(i/(i+1), \ldots\ldots, m). \qquad (3.10)$$

3.4.4 Results and Discussion

Each row of Table 3.1 corresponds to a sub-network of our proposed network which has learned the 400 digit samples of our first database (DB1). The column '# of nodes created' shows the number of nodes in each sub-network. For example, to separate samples of digit 1 from samples of all other digits three nodes are created such that each node separates samples of digit 1 from samples of one of the following three sets of digits {0,2,3,4}, {5,6}, {7,8,9}.

In an attempt to create a node to linearly separate samples of one digit from samples of other digits, we have to set the maximum number of iterations (presentations of the training set) which are allowed before we decide that the perceptron learning rule has not converged. The maximum number of iterations allowed for table 3.1 is 100.

The column '# of weights created' shows the total number of connection weights in each sub-network; i.e. to separate samples of digit 1 from all the other digits 3x65 = 195 connection weights are created. Upon presentation of each training sample, the connection weights to one node

(65 weights) are updated at a time. In other words, nodes are created one at a time.

The column '# of weights updated' shows the total number of weights that are updated to train each sub-network. The following example shows how to compute the number of weights that are updated in training a sub-network of our proposed network. Consider the sub-network of the second row of table 3.1 where samples of digit 1 are separated from samples of all the other digits. First, an attempt is made to do this separation by one node. A maximum number of iterations (100) through the training set (400 samples) is run before we decide that this separation can not be done by one node; 400x100 patterns are presented. Upon presentation of each pattern 65 connection weights to the node are updated. Hence, 400x100x65 weights are updated in an attempt to separate digit 1 from other digits by one node. Since this attempt is not successful, another attempt is made to separate digit 1 from digits 0,2,3,4 by one node, and also separate it from digits 5,6,7,8,9 by another. The weights to the node 1/0,2,3,4 are found in 40 iterations through the training set of 5x40 = 200 samples. Hence, 40x200x65 weights are updated to create node 1/0,2,3,4. After an unsuccessful attempt to create node 1/5,6,7,8,9 (240x100x65 weight updates), attempts are made to create nodes 1/5,6 and 1/7,8,9. These last two nodes are successfully created by 20 and 53

iterations through their respective training sets. Hence, the total number of weights updated to create the sub-network for digit 1 is computed as:

$400x100x65 + 200x40x65 + 240x100x65 + 120x20x65 + 160x53x65 = 5387.2x10^3$.

The first row of Table 3.2 corresponds to the network of Table 3.1 where the maximum number of iterations for each node creation is set to 100. The second row of it corresponds to another network where the maximum number of iterations is set to 300. Table 3.3 shows the generalization capability of our two networks of Table 3.2 when a test set of 100 samples is presented on them.

Table 3.6 and 3.7 show the sub-networks of two of our proposed networks which have learned the 700 training samples of our database 2(DB2). The maximum number of iterations allowed for Table 3.6 and Table 3.7 are 300 and 100, respectively. As Figure 3.6 shows the higher we set the maximum number of iterations the less nodes are created in the construction of our network. If this number is set at a too large number, there is a waste of C.P.U. time. Table 3.8 corresponds to a network designed according to the alternative to the proposed algorithm.

The rows of Table 3.9 show the generalization capability of the networks of Tables 3.8, 3.6, and 3.7 on a test set of 200 samples digits, respectively. As this table shows the network designed by the alternative to the

proposed algorithm (row 1) has higher misclassification than the other two networks.

Tables 3.4 and 3.10 show the results of constructing different networks to train samples of database 1 (DB1) and database 2 (DB2) by the backpropagation algorithm, respectively. Each network consists of a single-layer of hidden nodes. Hence, one of the problems is to determine the number of hidden nodes needed. The number of hidden nodes must be large enough for the network to be able to learn the problem, and restricted enough for the network to generalize properly [Eig93]. Having p number of training patterns, the number of hidden nodes, h, is approximated by [Mir89], [Geo91]:

$$h = \log_2 p \qquad (3.11)$$

where p = 400 for database 1 (DB1) and p = 700 for database 2 (DB2) in a 65-dimensional input space. For each network the learning rate and the momentum term were 0.35 and 0.9, respectively. Networks of Table 3.4 learned 100% of the 400 training samples, whereas one of the networks of Table 3.10 learned 99.86% of the 700 training samples by 300 iterations (through the training set) of the backpropagation algorithm.

According to our modular approach, the process of separating samples of one numeral from samples of all the other numerals is an independent sub-task. In other words, all these 10 sub-tasks can simultaneously run on different

processors. The maximum number of weights updated to train a sub-network of our proposed modular network of Table 3.1 is $5387.2 \times 10^3$ whereas the minimum number of weights updated training any of the comparable non-modular networks of Table 3.4 by the backpropagation is $2128 \times 10^4$. Comparison of the number of weights updated training each sub-network of our modular networks of Table 3.6 or of Table 3.7 with any of the comparable non-modular networks of Table 3.10 show that training our modular networks is faster than the comparable non-modular networks by the backpropagation algorithm. Working on a time shared system, it was hard to get accurate C.P.U. times spent during execution of the programs. On the basis of different runs, it was noticed that to train each sub-network of our modular network of Table 3.7 took about 0.1 second C.P.U. time whereas to train the second network of Table 3.10 by the backpropagation took about 1.5 second C.P.U. time.

The generalization capability of our modular networks could be compared with those of the non-modular networks trained using the backpropagation algorithm; Table 3.3 vs. Table 3.5; and Table 3.9 vs. Table 3.11. For Table 3.11 the networks of second and fourth row of Table 3.10 are used. Each row of Table 3.5 and Table 3.11 shows a different limit of error tolerated in the output nodes of the networks. $E_k$ is the measure of the error upon presentation

of pattern k:

$$E_k = \sum_j ( t_{kj} - o_{kj} )^2. \qquad (3.12)$$

Comparisons of Table 3.3 and Table 3.5 on one hand, and Tables 3.9 and 3.11 on the other show that the generalization capability of our networks is comparable to those of the networks learned using the backpropagation algorithm.

## 3.5 Conclusion

In this chapter a modular approach to a multi-class classification task is taken. In this approach a m-class task is broken into m 2-class sub-tasks where each sub-task is independently solved by a sub-network. An algorithm is presented for building and training each sub-network using any of the single-node network learning rules such as the perceptron rule. Our modular approach is compared to the non-modular approach in which the whole task is considered as one problem for which a network is designed and trained by the backpropagation algorithm. Our modular approach to a classification task has advantages over the non-modular approach in the following ways:

A. A network designed according to our proposed approach is trained faster than a non-modular network trained by the backpropagation.

a1. Each sub-network (module) of our modular

Table 3.1 This table shows the sub-networks created using the proposed algorithm for training 400 samples. Maximum number of iterations (presentation of all the training samples) allowed for each node creation is 100.

# of nodes created = # of nodes in a sub-network to separate one digit class from all other digit classes.
# of weigths created = # of nodes created x(64+1)
# of weight updates is explained earlier in this section (3.4.4).

| digit to be separated from all other digits | #of nodes created | # of weights created | # of iterations through the training set | # of weight updates |
|---|---|---|---|---|
| 0 | 1 | 65 | 17 | $442x10^3$ |
| 1 | 3 {0,2,3,4}, {5,6}, {7,8,9} | 195 | 313 =sum of 40,20,53,100, 100 | 5387.2 $x10^3$ |
| 2 | 1 | 65 | 25 | $650x10^3$ |
| 3 | 1 | 65 | 36 | $936x10^3$ |
| 4 | 1 | 65 | 15 | $390x10^3$ |
| 5 | 1 | 65 | 32 | $832x10^3$ |
| 6 | 2 {0,1,2,3}, {4,5,7,8,9} | 130 | 195 =sum of 13,82,100 | 4048.2 $x10^3$ |
| 7 | 1 | 65 | 41 | $1066x10^3$ |
| 8 | 1 | 65 | 91 | $2366x10^3$ |
| 9 | 1 | 65 | 30 | $780x10^3$ |
| Total | 13 | | | |

Table 3.2 Training 400 digit samples using proposed algorithm with different maximum number of iterations allowed for each node created.

| maximum number of iterations for each successful node created | # of nodes created for the network |
|---|---|
| 100 | 13 |
| 300 | 11 |

Table 3.3 Results showing generalization capability of the two networks of Table 3.2 on 100 test samples.

$R_{wc}$ = the rate of well classified patterns
$R_r$ = the rat of rejected patterns
$R_{mc}$ = the rate of misclasified patterns

$G = R_{wc} - (1/10)\ R_r - R_{mc}$.

| # of nodes in the network | $R_{wc}$ | $R_r$ | $R_{mc}$ | G |
|---|---|---|---|---|
| 11 | 42% | 41% | 17% | 20.9 |
| 13 | 46% | 41% | 13% | 28.9 |

Table 3.4 Different size networks for training 400 digit samples by Backpropagation algorithm.

| # of hidden nodes | # of output nodes | # of weights | # of iterations through the training set | # of weight updates | learned |
|---|---|---|---|---|---|
| 9 | 10 | 685 | 123 | $3370.2 \times 10^4$ | 100% |
| 10 | 10 | 760 | 70 | $2128 \times 10^4$ | 100% |
| 15 | 10 | 1135 | 69 | $3132.6 \times 10^4$ | 100% |

Table 3.5 Results showing generalization capability of the networks of Table 3.4 for 100 test samples.

$R_{wc}$ = the rate of well classified patterns
$R_r$ = the rate of rejected patterns
$R_{mc}$ = the rate of misclasified patterns

$G = R_{wc} - (1/10)\ R_r - R_{mc}$.

| number of hidden nodes | mean square error | $R_{wc}$ | $R_r$ | $R_{mc}$ | G |
|---|---|---|---|---|---|
| 9 | $E_k < 0.1$ | 40% | 53% | 7% | 27.7 |
| | $E_k < 0.2$ | 47% | 36% | 17% | 26.4 |
| | $E_k < 0.3$ | 54% | 20% | 26% | 26.0 |
| 10 | $E_k < 0.1$ | 30% | 62% | 8% | 15.8 |
| | $E_k < 0.2$ | 41% | 44% | 15% | 21.6 |
| | $E_k < 0.3$ | 52% | 26% | 22% | 27.4 |
| 15 | $E_k < 0.1$ | 35% | 57% | 8% | 21.3 |
| | $E_k < 0.2$ | 43% | 41% | 16% | 22.9 |
| | $E_k < 0.3$ | 48% | 33% | 19% | 25.7 |

Table 3.6 This table shows the sub-networks created using the proposed algorithm for training 700 sample digits. Maximum number of iterations (presentation of all training samples) allowed for each node created is 300.

| digit to be separated from all other digits | # of nodes created | # of connection weights created | # of iterations through the training set | # of weight updates |
|---|---|---|---|---|
| 0 | 2 {1,2,3,4} {5,6,7,8,9} | 130 | 435 = sum of 70,65,300 | $170.17 \times 10^5$ |
| 1 | 3 {0,2,3,4} {5,6} {7,8,9} | 195 | 947 = sum of 206,35,106, 300,300 | $289.3345 \times 10^5$ |
| 2 | 1 | 65 | 156 | $70.98 \times 10^5$ |
| 3 | 2 {0,1,2,4} {5,6,7,8,9} | 130 | 508 = sum of 81,127,300 | $189.5985 \times 10^5$ |
| 4 | 2 {0,1,2,3} {5,6,7,8,9} | 130 | 465 = sum of 20,145,300 | $180.635 \times 10^5$ |
| 5 | 2 {0,1,2,3} {4,6,7,8,9} | 130 | 547 = sum of 114,133,300 | $198.744 \times 10^5$ |
| 6 | 3 {0,1,2,3} {4,5} {7,8,9} | 195 | 823 = sum of 20,192,11, 300,300 | $251.16 \times 10^5$ |
| 7 | 1 | 65 | 220 | $100.1 \times 10^5$ |
| 8 | 2 {0,1,2,3} {4,5,6,7,9} | 130 | 429 = sum of 58,71,300 | $169.078 \times 10^5$ |
| 9 | 1 | 65 | 86 | $39.13 \times 10^5$ |
| Total | 19 | | | |

Table 3.7 This table shows the sub-networks created using the proposed algorithm for training 700 sample digits. Maximum number of iterations (presentation of all training samples) allowed for each node created is 100.

| Digit to be separated from all other digits | # of nodes created | # of weights created | # of iterations through the set | # of weight updates |
|---|---|---|---|---|
| 0 | 2{1,2,3,4}, {5,6,7,8,9} | 130 | 235 | $79.17 \times 10^5$ |
| 1 | 5{0,2},{3,4},{5,6},{7},{8,9} | 325 | 563 | $133.861 \times 10^5$ |
| 2 | 2{0,1,3,4},{5,6,7,8,9} | 130 | 206 | $71.1165 \times 10^5$ |
| 3 | 3{0,1,2,4},{5,6},{7,8,9} | 195 | 337 | $100.4185 \times 10^5$ |
| 4 | 3{0,1,2,3},{5,6},{7,8,9} | 195 | 277 | $85.722 \times 10^5$ |
| 5 | 4{0,1},{2,3},{4,6},{7,8,9} | 260 | 400 | $110.3375 \times 10^5$ |
| 6 | 4{0,1,2,3},{4},{5},{7,8,9} | 260 | 374 | $96.915 \times 10^5$ |
| 7 | 3{0,1},{2,3},{4,5,6,8,9} | 195 | 287 | $83.265 \times 10^5$ |
| 8 | 2{0,1,2,3},{4,5,6,7,9} | 130 | 229 | $78.078 \times 10^5$ |
| 9 | 1{0,1,2,3,4,5,6,7,8} | 65 | 86 | $39.13 \times 10^5$ |
| Total | 29 | | | |

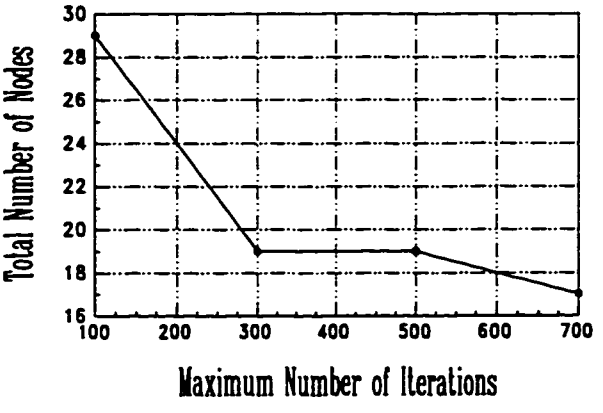


Figure 3.6 The number of nodes in the networks vs. different number of maximum iterations allowed for creation of each node by the proposed algorithm.

Table 3.8 This table shows the number of nodes created using the alternative to the proposed algorithm for training 700 sample digits. Each digit class is separated from the digit classes numbered higher than itself. Maximum number of iterations allowed to create each node is 300.

| digit to be separated from digits numbered higher than itself | # of nodes created | # of weights trained |
|---|---|---|
| 0 | 2{1,2,3,4},{5,6,7,8, | 130 |
| 1 | 9} | 195 |
| 2 | 3{2,3,4,5},{6,7},{8, | 65 |
| 3 | 9} | 65 |
| 4 | 1 | 65 |
| 5 | 1 | 65 |
| 6 | 1 | 65 |
| 7 | 1 | 65 |
| 8 | 1 | 65 |
| | 1 | |
| | 1 | |
| Total | 12 | |

Table 3.9 Results showing generalization capability of the networks of Tables 3.6 - 3.8 for 200 test samples.

$R_{wc}$ = the rate of well classified patterns
$R_r$ = the rate of rejected patterns
$R_{mc}$ = the rate of misclasified patterns

$G = R_{wc} - (1/10)R_r - R_{mc}$.

| number of nodes in the network | $R_{wc}$ | $R_r$ | $R_{mc}$ | G |
|---|---|---|---|---|
| 12 | 82% | 0 | 18% | 64.00 |
| 19 | 66.5% | 27.5% | 6% | 57.75 |
| 29 | 72% | 24.5% | 3.5% | 66.05 |

Table 3.10 Different size networks for training 700 digit samples by Backpropagation algorithm.

| number of hidden nodes | number of output nodes | weights trained | number of iterations through the training set | # of weight updates | learned |
|---|---|---|---|---|---|
| 10 | 10 | 760 | 100 | $53.2 \times 10^6$ | 75.12% |
| 10 | 10 | 760 | 300 | $159.6 \times 10^6$ | 99.86% |
| 10 | 10 | 760 | 500 | $266 \times 10^6$ | 99.86% |
| 15 | 10 | 1135 | 100 | $79.45 \times 10^6$ | 99.4% |
| 15 | 10 | 1135 | 300 | $238.35 \times 10^6$ | 99.1% |

Table 3.11 Results showing generalization capability of the networks of Table 3.10 for 200 test samples.

$R_{wc}$ = the rate of well classified patterns
$R_r$ = the rate of rejected patterns
$R_{mc}$ = the rate of misclasified patterns

$G = R_{wc} - (1/10)\ R_r - R_{mc}$.

| number of hidden nodes | mean square error | $R_{wc}$ | $R_r$ | $R_{mc}$ | G |
|---|---|---|---|---|---|
| 10 | $E_k$<0.1 | 67.5% | 25% | 7.5% | 57.50 |
| | $E_k$<0.2 | 71% | 20% | 9% | 60.00 |
| | $E_k$<0.3 | 76% | 13.5% | 10.5% | 64.15 |
| 15 | $E_k$<0.1 | 66.5% | 31.5% | 2% | 61.35 |
| | $E_k$<0.2 | 74% | 21.5% | 4.5% | 67.35 |

network can be trained independently and in parallel.

a2. The number of weights updated to train each sub-network is less than the number of weights updated to train the non-modular network by backpropagation. The simulation results for the digit recognition task presented in this chapter reveal this finding.

a3. Each weight updated by the perceptron learning rule takes less computation than a weight updated by backpropagation.

B. As the simulation results of this chapter show, the generalization capabilities of our modular networks are comparable to the non-modular networks trained by backpropagation.

C. Our modular approach is adaptable; adding a new class to an existing classification problem introduces only new nodes to separate this class from the existing classes.

D. Finally, the number of nodes in our modular network is not guessed and fixed in advance. By setting a maximum number of attempts allowed to linearly separate patterns of one class from those of other classes, the nodes are created as needed.

# CHAPTER 4

# TRAINING USING CORNER CLASSIFICATION

## 4.1 Introduction

In this chapter another modular approach [Kak94] is presented. This approach is more modular than the approach of the previous chapter. Instead of breaking a m-class classification task into m 2-class sub-tasks as done in the previous chapter, the task is broken into many more 2-class sub-tasks. The number of these sub-tasks can grow to equal the number of training samples used. The main motivation behind this approach is to have a network trained much faster than the networks of other approaches, i.e. networks of backpropagation. One hidden node is allocated for each training pattern. This hidden node recognizes the designated training pattern and patterns at a close hamming distance from it. The weights to this node are adjusted by a direct approach rather than the iterative approach of backpropagation which makes this approach very fast.

The following section describes the procedure for building the structure of the network. Later sections review the different learning rules called corner classification algorithms [Kak93b],[Kak94] for training the connection weights in the network. These learning algorithms are studied and then modified to give better results. Finally, a pruning procedure is given that

eliminates the unnecessary hidden nodes to make the network smaller. The reduced network has the same performance as that of the unpruned network.

## 4.2 The Network Structure

This network structure along with its learning algorithms can be used to map any given binary associations [Kak93b], [Kak94]. In this network the output of the hidden node j is given by the function:

$$o_j = \begin{cases} 1 & if \sum_{i=1}^{n+1} w_{ji} x_i > 0 \\ 0 & otherwise. \end{cases} \qquad (4.1)$$

An extra input of value one is fed as input to each hidden node whose weight with that node simulates the node's threshold.

Consider the function Y = f(X), where X and Y are p binary vectors of dimensions n and m respectively.

Table 4.1 Input output pattern pairs.

| Sample | Inputs | Outputs |
|---|---|---|
| 1 | $x_{11}x_{12}\cdots x_{1n}$ | $y_{11}y_{12}\cdots y_{1m}$ |
| 2 | $x_{21}x_{22}\cdots x_{2n}$ | $y_{21}y_{22}\cdots y_{2m}$ |
| . | . | . |
| . | . | . |
| . | . | . |
| p | $x_{p1}x_{p2}\cdots x_{pn}$ | $y_{p1}y_{p2}\cdots y_{pm}$ |

The p output patterns could be viewed as m p-dimensional vectors $Z_i$s. Consider the output vector $Z_1$, whose elements are the first output elements ($y_1$ values) for the p inputs. There are $\beta_1$ elements of value 1 in $Z_1$. The key step in this algorithm is to create $\beta_1$ nodes in the second layer in order to be able to separate the individual n-dimensional input patterns that produce a value of 1 for $y_1$. These $\beta_1$ nodes are connected to the n input lines through weights that define the correct hyperplanes to isolate each of the input patterns. The outputs of these $\beta_1$ nodes go through a logical-OR gate at the output node to produce the correct response at the output layer. This procedure is done for the other $Z_i$ vectors, too.

Figure 4.1 shows the structure of this network. In classification tasks the number of output elements, m, is equal to the number of classes. Each output element represents a class for which only the input patterns belonging to it have a value of 1 and all other patterns have a value of 0. This structure only needs to train a single layer of weights, the weights of connections from the n-input lines to the hidden nodes.

## 4.3 Corner Classification Algorithms

To train the connections from the input nodes to the hidden nodes of the described network, one can use the Perceptron rule. Another approach presented by Kak [Kak94] is called corner classification. In this approach training

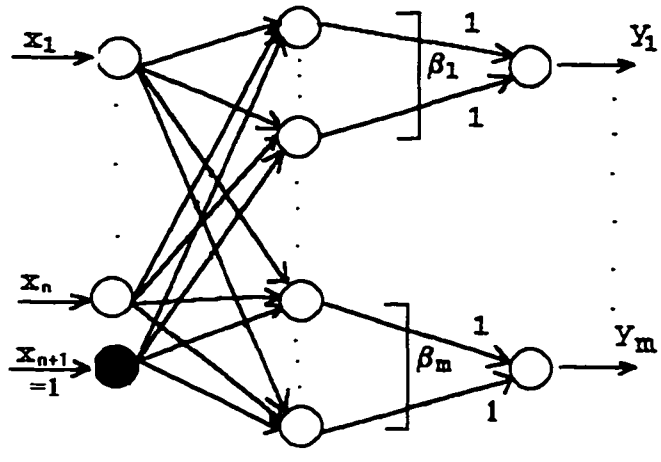


Fig. 4.1: The general structure of the network.

samples are considered as the corners of a n-dimensional cube. The algorithms that find a set of weights to isolate a corner of the n-dimensional cube do so by a direct method of inspection of the training samples.

This direct way of adjustment of weights has the benefit that information does not have to travel in reverse direction. Some researchers [Hin92] believe that the backpropagation algorithm seems biologically implausible. In backpropagation information travels through the same connections in reverse direction. It seems that this does not happen in real neurons.

The direct style of weight adjustment of the corner classification algorithms result in very fast training. For these algorithms the amount of computation grows only linearly with the number of patterns to learn. Also, the time to learn each pattern is independent of the number of training patterns and as a result it is very fast. This contrasts with the slow training procedure of the backpropagation algorithm for which the time to learn each training pattern depends on the number of training patterns.

### 4.3.1 Algorithm CC2

This algorithm, CC2 [Kak94], is based on the corner classification learning approach. Each hidden node isolates a particular training pattern vector, $X^i$. It produces a 1 when $X^i$ is presented and a 0 when other patterns are applied. To accomplish this, the weights must be assigned so that they are positively correlated with $X^i$ and negatively correlated with the other input training vectors. Suppose the input training vector $X^i$ is represented by vector $(x_1, x_2, \ldots\ldots x_n\ x_{n+1})$. The weights of connections to this hidden node are computed as:

$$w_j = \begin{cases} -1 & \text{for } x_j = 0 \\ 1 & \text{for } x_j = 1 \\ -(s-1) & \text{for } j = n+1 \end{cases} \qquad (4.2)$$

where s is the number of 1's in the input vector $X^i$, and

$w_{n+1}$ simulates the threshold of the hidden node. However, this learning algorithm does not generalize from the training patterns. Each hidden node of the network picks only one training pattern, one corner of the n-dimensional cube. In order for CC2 to generalize, it should pick the neighboring corners as well. It has been suggested in [Kak94] that by randomly modifying the weight values after using CC2 some of the learnt training patterns become unlearned, but the network achieves generalization capability. One interesting randomization [Mad94] adds small positive values to weights with positive value and small negative values to weights with negative value. This makes the network to classify patterns in the vicinity of the corner patterns as the corner patterns.

### 4.3.2 Algorithm CC3

The randomization process of algorithm CC2 could result in misclassifying patterns and thus rejecting them. This specially could happen when large random numbers are added and subtracted from the weights. In order to somewhat correct this problem, a procedure is suggested in [Kak94] that for each weight vector $W^i$ it increases its correlation with its corresponding input training vector $X^i$, and it decreases its correlation with all other input vectors in the training set. This procedure works in the following manner [Rai94]. Consider the weight vector $W^i$ corresponding to pattern $X^i$ in the training set. The inner

product of this weight vector, $W^i$, with every input vector $X^j$ in the training set such that $j \neq i$, $s = X^j.W^i$, is computed. If $s>0$, one of the followings is followed:

1. Find a 1 such that $x^i_1 = 0$ and $x^j_1 = 1$, update $w^i_1$ to $w^i_1-s$. Note that $x_1^i$ is an element of the $X^i$ training vector corresponding to $W^i$ weight vector.
2. Find a 1 such that $x^i_1 =1$ and $x^j_1 = 0$, update $w^i_1$ to $w^i_1+s$ and change $w^i_{n+1}$ to $w^i_{n+1}-s$.

4.3.3 A Remark on Corner Classification

Consider a set of input patterns represented as vectors in a n-dimensional space. A hidden node in a feedforward network consisting of one hidden layer, such as a network discussed in this chapter, acts as a (n-1)-dimensional hyperplane that divides the space into two regions [Mir89]. If the portions of these hyperplanes divide the input space into r disjoint regions, we say that the input space is linearly separable into r regions. The association of the regions with classes is done by the output nodes at the output layer. Figure 4.2 shows an input space divided into 5 disjoint regions. Each region is associated with one of the 2 classes. Having a n-dimensional space, the total number of binary

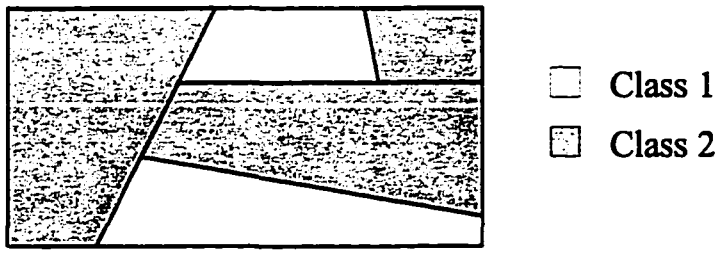


Figure 4.2 Two-dimensional input space with five separable regions divided into two classes.

patterns in this space is $2^n$. Assuming that the space of the $2^n$ patterns is divided into r disjoint regions where each region i has $p_i$ number of patterns, then we have that:

$$\sum_{i=1}^{r} p_i = 2^n. \qquad (4.3)$$

Let each region i be identified by the following two characteristics; (1) a pattern; (2) a radius of generalization $d_i$. In other words let us assume that associated with each region i there is a pattern such that the other patterns in this region are at most at a hamming distance of $d_i$ from it. Then the number of patterns in

each such region i with $d_i$ radius of generalization is [Kak95]:

$$p_i = \sum_{k=0}^{d_i} \binom{n}{k}. \qquad (4.4)$$

Also that the total number of patterns in all the r regions is:

$$\sum_{i=1}^{r} \left( \sum_{k=0}^{d_i} \binom{n}{k} \right) = 2^n. \qquad (4.5)$$

Example

Consider n=3, the total number of all possible binary patterns is $2^3=8$. Assume that this space is divided into the following two regions:

| region 1 | region 2 |
|---|---|
| 001 | 000 |
| 010 | 100 |
| 011 | 101 |
| 111 | 110 |

If we pick the patterns 011 and 100 from the above two regions as the patterns for each region respectively, and take the radius of generalization $d_i=1$ for each region then we have all the patterns:

$$2 \times \sum_{k=0}^{d_i=1} \binom{3}{k} = 2^3. \qquad (4.6)$$

If one places a training pattern in each of the regions and sets the radius of generalization for each region such that it covers the patterns in that region, then the classification problem is solved. This is the idea behind the corner classification algorithms; i.e. by randomly adding positive values to weights with positive value and adding negative values to weights with negative value, one is trying to create a radius of generalization.

Not knowing which patterns to place as centers of the regions, one picks all the training patterns and forms a region around each one. In doing so, one has to pick the radius for each region large enough to be able to pick patterns close by and small enough not to include outside patterns. Also that having placed a pattern in region i, some patterns at $d_i$ hamming distance from it may belong to this region while some other patterns at the same hamming distance may not belong to this region. In the following sections, we modify the corner classification algorithms so as to have a better radius of generalization for each region.

### 4.3.4 Modified CC2

Consider training pattern $X^i=(x^i_1, x_2{}^i, \ldots, x^i_{n+1})$ belonging to class c. The weights $W^i=(w^i_1, w_2{}^i, \ldots, w^i_{n+1})$ to the hidden node for this pattern are assigned as:

$$w_j^i = \begin{cases} 1 + \alpha * \dfrac{p_j}{p_c} & \text{if } x_j^i = 1 \\ -1 - \alpha * \dfrac{p_c - p_j}{p_c} & \text{if } x_j^i = 0 \\ -(s - 1) & \text{for } j = n + 1 \end{cases} \qquad (4.7)$$

where $p_j$ is the number of training patterns belonging to class c that have a value of 1 in their $j^{th}$ element. Also, $p_c$ is the total number of training patterns in class c and $\alpha$ is an integer value that gives the step size. In the same way as before, s is the number of 1s in the $X^i$ training vector.

This way of adjusting the weights differs from the algorithm of CC2 in that to create a radius of generalization knowledge of the training patterns is used. For example, for a connection to a hidden node corresponding to a pattern of a particular class, if most of the patterns in that class have one sign then a greater weight is assigned toward that sign.

### 4.3.5 Modified CC3

When adjusting the weights according to the procedure of the modified CC2, as larger values are chosen for $\alpha$ more of the training patterns become unlearned. This is because the radiuses of the generalization become larger which

result in having overlapping regions. This causes the generalization to degrade.

In order to ensure that most of the training patterns are learned, after using the procedure of the modified CC2 a modification of the algorithm of CC3 is used. For each weight vector $W^i$, where $W^i$ corresponds to $X^i$, compute its inner product

with every other input vector $X^j$ in the training set($j \neq i$);

$$s = X^j \cdot W^i. \qquad (4.8)$$

If s exceeds zero, do the followings;

1. Compute the number of input places where patterns $X^i$ and $X^j$ differ; call this number c.
2. For each place where $x^i_1 = 0$ and $x^j_1 = 1$, update $w_1$ to $w_1 - s/c$.
3. For each place where $x^i_1 = 1$ and $x^j_1 = 0$, update $w_1$ to $w_1 + s/c$ and also update $w_{n+1}$ to $w_{n+1} - s/c$.

As of CC3 this procedure increases the correlation between $W^i$ and $X^i$, and reduces the correlation between $W^i$ and the other input vectors in the training set. This procedure differs from the CC3 in that the weights are updated in a smoother way. Instead of updating one element of a weight vector, the change is broken into small pieces with more elements updating each piece. Experiments have shown that networks with smoother solution weights generalize better than networks with less smooth weights[Jea94].

### 4.3.6 Simulation Results

The digit recognition task of chapter 3 is used to compare the generalization capability of the corner classification algorithms with that of backpropagation. Database 1 of chapter 3 which consists of 400 training samples and 100 test samples is used. Tables 4.2 and 4.3 show the results using the different corner classification algorithms. Each of these algorithms adds positive values to positive weights and negative values to negative weights in certain way. For each of them the ranges of positive and negative values added are $(0,\alpha)$ and $(-\alpha,0)$, respectively. As the results of tables 4.2 and 4.3 show, the performance of the modified CCs are better than the original CCs. Table 4.4 shows the generalization ability of two networks of chapter 3 trained by backpropagation. One of the networks has 10 hidden nodes in its second layer and the other network has 15 hidden nodes. Both networks learned the 400 input training patterns. Generalization ability of the corner classification networks ( as shown in Tables 4.2 - 4.3) are comparable to the generalization ability of the networks trained by backpropagation (as shown in Table 4.4). However, in the learning time, the corner classification algorithms outperform backpropagation.

Table 4.2 Different corner classification Algorithm used for the digit recognition task. In each algorithm $\alpha=2$.
$R_{wc}$ = *the rate of well classified patterns*
$R_r$ = *the rate of rejected patterns*
$R_{mc}$ = *the rate of misclassified patterns*
$G = R_{wc} - (1/10)R_r - R_{mc}$

| Learning 400 samples | | | | Generalization 100 samples | | | |
|---|---|---|---|---|---|---|---|
| Algorithm | $R_{wc}$ | $R_r$ | $R_{mc}$ | $R_{wc}$ | $R_r$ | $R_{mc}$ | G |
| CC2 | 74% | 26% | 0% | 40% | 51% | 9% | 25.9 |
| Modified CC2 | 77.3% | 22.7% | 0% | 40% | 55% | 5% | 29.5 |
| CC3 | 90% | 10% | 0% | 37% | 54% | 9% | 22.6 |
| Modified CC3 | 95.8% | 4.2% | 0% | 38% | 57% | 5% | 27.3 |

Table 4.3 Different corner classification algorithm used for the digit recognition task ($\alpha$=4).
*$R_{wc}$ = the rate of well classified patterns*
*$R_r$ = the rate of rejected patterns*
*$R_{mc}$ = the rate of misclassified patterns*
$G = R_{wc} - (1/10)R_r - R_{mc}$

| | Learning 400 samples | | | Generalization 100 samples | | | |
|---|---|---|---|---|---|---|---|
| Algorithm | $R_{wc}$ | $R_r$ | $R_{mc}$ | $R_{wc}$ | $R_r$ | $R_{mc}$ | G |
| CC2 | 19% | 81% | 0% | 17% | 78% | 5% | 4.2 |
| Modified CC2 | 40.3% | 59.7% | 0% | 33% | 59% | 8% | 19.1 |
| CC3 | 70.8% | 24.2% | 0% | 32% | 56% | 12% | 14.4 |
| Modified CC3 | 89% | 11% | 0% | 37% | 57% | 6% | 25.3 |

Table 4.4 Results showing generation capability of the backpropagation algorithm on 100 test samples.
$R_{wc}$ = *the rate of well classified patterns*
$R_r$ = *the rate of rejected patterns*
$R_{mc}$ = *the rate of misclassified patterns*
$G = R_{wc} - (1/10)R_r - R_{mc}$

| number of hidden nodes | mean square error | $R_{wc}$ | $R_r$ | $R_{mc}$ | G |
|---|---|---|---|---|---|
| 10 | $E_p < 0.1$ | 30% | 62% | 8% | 15.8 |
| | $E_p < 0.2$ | 41% | 44% | 15% | 21.6 |
| | $E_p < 0.3$ | 52% | 26% | 22% | 27.4 |
| 15 | $E_p < 0.1$ | 35% | 57% | 8% | 21.3 |
| | $E_p < 0.2$ | 43% | 41% | 16% | 22.9 |
| | $E_p < 0.3$ | 48% | 33% | 19% | 25.7 |

## 4.4 Pruning of Hidden Nodes in Corner Classification

An important question in neural network research is the size of network; the number of nodes and connections needed for a particular task. The size of network must be large enough to learn the training patterns, and small enough to extract the relevant features from them. It is commonly believed that smaller networks that fit the data generalize better than larger networks [Ree93]. This is because smaller networks have less space to store information. Hence, they extract only the relevant features when learning the training set. Besides better generalization, smaller networks have other advantages to larger networks; they cost less to build and are faster.

For backpropagation networks, not knowing the optimum size network, one starts with a network larger than necessary and then tries to prune it. Studies for the backpropagation networks have shown that even if the optimum size is known, the smallest size network to fit the data is usually sensitive to initial conditions and rarely moves to a solution. Many algorithms start with a network larger than necessary and then remove the nodes or connections that are not required [Kar90], [Sie91]. In the following section a procedure for pruning corner classification networks is described. This is then followed by some simulation results.

4.4.1 A Procedure

To ensure the separation of the separable regions in the input space one would like to place a training pattern in each of the regions and then create a hidden node for each pattern according to one of the corner classification algorithms. But, for most problems one does not know the shapes of the regions so as to choose the necessary training patterns. Usually one takes all the training patterns and creates a hidden node for each one. Since the number of training patterns is often high, a pruning procedure is desired to reduced the number of hidden nodes. In this procedure, the outputs of the hidden nodes are examined to determine which nodes can be eliminated.

Consider the hidden nodes corresponding to the training patterns of class c. For each such hidden node i, compute two numbers; (1) the number of training patterns in class c for which hidden node i gives an output of 1; (2) the number of training patterns in all other classes for which node i gives output of 1. We say that node i covers the training patterns for which it gives output of 1. The hidden node with a maximum number of covering of training patterns in class c and a low number of covering of training patterns in other classes is kept. All the hidden nodes corresponding to the covered training patterns of class c are eliminated. Since their removal does not change the final output. Each hidden node separates a

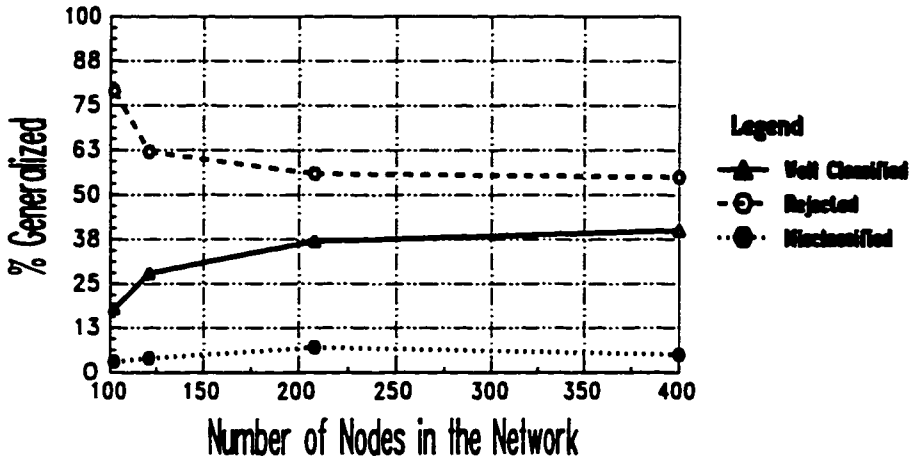


Figure 4.5 The effect of pruning a network of corner classification on generalizing 100 digit test samples.

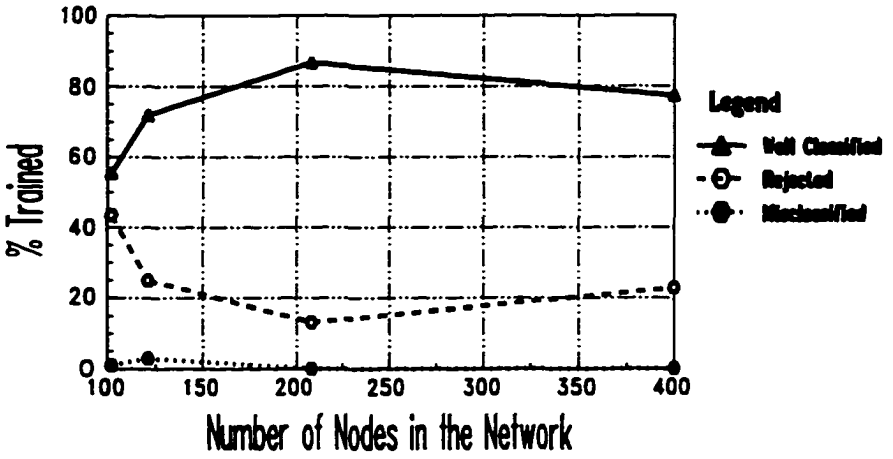


Figure 4.6 The effect of pruning a network of corner classification on learning 400 digit samples.

region around one training pattern. If a few training patterns of a class are in one region and can be separated by one node, then the nodes that separate each of the patterns in that region separately can be removed.

This procedure is repeated until all or most of the training patterns of class c are covered by some hidden nodes. Following this procedure for the hidden nodes of the other classes, one can prune the network.

### 4.4.2 Simulation Results

The second network of table 4.2 with 400 hidden nodes is used to be pruned by the above procedure. When every training pattern is covered by some hidden node according to the pruning procedure, a total of 208 hidden nodes is used. Some training patterns are rejected because two or more hidden nodes of different classes cover them. As figures 4.5 and 4.6 show, when half the number of hidden nodes are eliminated according to our procedure the performance of the network remains comparable to that of the unpruned network.

# CHAPTER 5

# CONCLUSION

In this dissertation two different modular approaches for designing feedforward neural networks for classification problems are presented. In the first approach, discussed in chapter 3, a classification task is broken into sub-tasks each one of which is solved by a sub-network. An algorithm is presented by building and training each sub-network using any of the single-node learning rules such as the perceptron rule. The performance of this approach is compared with that of the popular non-modular approach of the backpropagation algorithm by their application on a digit recognition task. The simulation results show that this approach is faster than backpropagation while it has similar generalization. This approach has the benefit of incrementally building the hidden nodes as opposed to that of backpropagation where there is no procedure to guide one to a specific architecture.

The second modular approach, presented in chapter 4, is used in conjunction with a recent method called corner classification. Training using corner classification is a one step method and as a result it is a much faster method than the computationally intensive iterative method of backpropagation. In this dissertation some modifications

are made to corner classification to improve its performance. The digit recognition task of chapter 3 is used to compare the generalization capability of the corner classification algorithms with that of backpropagation, and it is shown that they are comparable. Designing a network by corner classification involves a large number of hidden nodes. As the number of hidden nodes increases the network becomes more fault tolerant. However, the increase of hidden nodes will require more components for the hardware implementation. So, the unnecessary hidden nodes must be eliminated. In chapter 4 a pruning procedure is introduced that eliminates a large number of unnecessary hidden nodes.

Training algorithms based on iterative procedures have the benefit of not creating too many hidden nodes and the drawback of being too slow. Training based on corner classification has the benefit of being very fast at the expense of creating too many hidden nodes. As a direction for further research one could combine these two approaches by arranging a number of training patterns belonging to a class into clusters and use a hidden node for each cluster. This could be done by some form of corner classification followed by a pruning procedure. Then an iterative learning procedure such as the perceptron rule could be used to learn the remaining training patterns. This way the number of hidden nodes could be reduced compared to the

approach of using only corner classification while having a faster method than an iterative one.

# REFERENCES


[Alp93] E. Alpaydin," Multiple Networks for Function Learning," *IEEE International Conference on Neural Networks*, pp.9-14, 1993.

[Ana 95] R. Anand, K. Mehrotra, C.K. Mohan, and S. Ranka, " Efficient Classification for Multiclass Problems Using Modular Neural Networks," *IEEE Trans. on Neural Networks*, vol. 6, no. 1, January 1995.

[Aud94] G. Auda, M. Kamel, and H. Raafat, " A New Neural Network Structure with Cooperative Modules," *IEEE International Conference on Neural Networks*, pp.1301-1306, 1994.

[Bal93] P. Baldi, and N.B. Toomarian, " Learning Trajectories with a Hierarchy of Oscillatory Modules," *IEEE International Conference on Neural Networks*, pp.1172-1176, 1993.

[Cam92] D. Van Camp, " Neurons for Computers," *Scientific American*, September 1992.

[Cho92] Sung-Bas Chon and J.H. Kim, " A Two-Stage Classification Scheme with Backpropagation Neural Network Classifiers," *Pattern Recognition Letters 13*, pp. 309-313, May 1992.

[Cor95] L. P. Cordella, C. De Stefano, F. Tortorella, and M. vento, " A Method for Improving Classification Reliability of Multilayer Perceptrons," *IEEE Trans. on Neural Networks*, vol. 6, Sept. 1995, pp.1140-1147.

[Dam92] A. R. Damasio, and H. Damasio, " Brain and Language," *Scientific American*, Sept. 1992.

[Eig93] V. Eigel-Danielson, and M.F. Augusteijm, "Neural Network Pruning and its Effect on Generlization, Some Experimental Results," *Neural Parallel & Scientific Computations 1*, 59-70, 1993.

[Fre91] J. A. Freeman and D.M. Skapura, *Neural Networks Algorithms, Applications, and Programming Techniques*, Addison-Wesley Publishing Company, 1991.

[Geo91] G. M. Georgiou, "Comments on 'On Hidden Nodes in Neural Nets', " *IEEE Trans. on Circuits and Systems*, vol. 38, no. 11, 1991.

[Gol92] P. S. Goldman-Rakic, "Working Memory and the Mind," *Scientific American*, Sept. 1992.

[Goo94] R. M. Goodman, and Z. Zeng, " A Learning Algorithm for Multilayer Perceptrons with Hard-Limiting Threshold Units," *IEEE International Conference on Neural Networks*, pp. 193-197, 1994.

[Has90] P. Hashemian, " A Controller to Improve the Convergence of a Neural Network," *IEEE Proceedings of the Southeastcon'90*, vol. 1, pp. 69-71, 1990.

[Has91] P. Hashemian, "A Controller Based on the Energy Function to Improve Convergence of a Neural Network," *IEEE Proceedings of the Southeastcon'91*, vol. 2, pp. 741-745, 1991.

[Hay94] S. Haykin, *Neural Networks A Comprehensive Foundation*, Macmillan College Publishing Company, Inc. 1994.

[Hin92] G. E. Hinton, " How Neural Networks Learn from Experience," *Scientific American*, Sept 1992.

[Hop82] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Natl. Acad. Sci. USA*, vo. 79. pp. 2554-2558, 1982.

[Jea94] J.S.N. Jean and J. Wang, "Weight Smoothing to Improve Network Generalization," *IEEE Trans. on Neural Networks*, vol. 5, no. 5, September 1994.

[Kak89] S.C. Kak and M.C. Stinson, "A Bicameral Neural Network where Information Can Be Indexed," *Electronics Letters*, vo. 25, pp. 203-205, 1989.

[Kak92] S.C. Kak, "State Generators and Complex Neural Memories," *Pramana-Journal of Physics*, vol. 38, pp. 271-278, 1992.

[Kak93a] S.C. Kak, "Feedback Neural Networks: New Characteristics and a Generalization," *Circuits Systems Signal Process*, vol. 12, no. 2, pp. 263-278, 1993.

[Kak93b] S.C. Kak, "On Training Feedforward Neural Networks," *Pramana-Journal of Physics*, vol. 40, no. 1, January 1993.

[Kak94] S.C. Kak, "New Algorithms for Training Feedforward Neural Networks", *Pattern Recognition Letters 15*, 295-298, March 1994.

[Kak95] S.C. Kak, "The Three Languages of the Brain:Quantum, Reorganizational, and Associative", *4th Appalachian Conf. on Behavioral Neurodynamics*, Redford, VA, Sept. 1995.

[Kar90] E.D. Karnin, "A Simple Procedure for Pruning Back-Propagation Trained Neural Networks", *IEEE Trans. on Neural Networks*, vol.1, no.2, 1990.

[Kne90] S. Knerr, L. Personnaz, and G. Drefus, "Single-layer Learning Revisited: A Stepwise Procedure for Building and Training a Neural Network," in *Neuro-computing*, F. Fogelman, and J.Herault,Eds, NewYork: Springer, 1990.

[Kne92] S.Knerr, L. Personnaz, and G.Dreyfus, "Handwritten Digit Recognition by Neurla Networks with Single-layer Training," *IEEE Trans. Neural Networks*, vol.3, no., 6, November 1992.

[Lip87] R.P. Lippmann, " An Introduction to Computing with Neural Nets," *IEEE ASSP Mag.*, pp. 4-22, 1987.

[Mad94] K.B. Madineni, "Two Corner Classification Algorithms for Training the Kak Feedforward Neural Network," *Information Sciences* 81, 229-234, 1994.

[McS92] John McShane, "An Introduction to Neural Nets," *Hewlett - Packard Journal*, February 1992.

[Mir89] G. Mirchandani, and W. Cao, "On Hidden Nodes for Neural Nets," *IEEE Trans. on Circuits and Systems*, vol. 36, no.5, May 1989.

[Rai94] P. Raina, "Comparison of Learning and Generalization Capabilities of the Kak and the Backpropagation Algorithms ", *Information Sciences* 81, 261-274, 1994.

[Ree93] R. Reed, "Pruning Algorithms - A Survey," *IEEE Trans. on Neural Networks*, vol. 4, no. 5, 1993.

[Ros73] F. Rosenblatt, *Principles of Neurodynamics*, Spartan Books, 1973.

[Roy95] V.P. Roychowdhury, K.-Y. Siu, and T. Kailath, "Classification of Linearly Non-Separable Patterns by Linear Threshold Elements," *IEEE Trans. on Neural Networks*, vol.6, no.2, March 1995.

[Rum86] D.E. Rumelhart, J.L. McClelland, et al, *Parallel Distributed Processing*, vol. 1., Cambridge, The MIT Press, 1986.

[Sha92] C. J. Shatz, " The Developing Brain," *Scientific American*, Sept. 1992.

[Sti88] M.C. Stinson, *Neural Networks with Asynchronous Control*, Ph.D. Dissertation, LSU, 1988.

[Sie91] J. Sietsma, and R.J.F. Dow, "Creating Artificial Neural Networks That Generalize," *Neural Networks*, vol. 4, 1991.

[Tho85] R.F. Thompson, *The Brain An Introduction to Neuroscience*, W.H. Freeman & Company, 1985.

[Ull73] J.R. Ullmann, *Pattern Recognition Technigues*, London Butterworths, 1973.

[Vee95] L. P. J. Veelenturf, *Analysis and Applications of Artificial Neural Networks*, Prentice Hall International(UK) Ltd., 1995.

[Vem90] V. Vemuri, "Artificial Neural Networks: An Introduction," *IEEE Computer Society Press*, 1990.

[Wan91] J. Wang, and J. Jean, " Automatic Rule Generation for Machine Printed Character Recognition Using Multiple Neural Networks," *IEEE International Conference on Systems Engineering*, pp. 343-346, Aug. 1991.

[Wid90] Bernard Widrow and Michael A. Lehr, "30 Years of Adaptive Neural Networks : Perceptron, Madaline, and Backpropagation", *Proc. IEEE*, vol. 78, no. 9, pp. 1415-1442, Sept. 1990.

[Yon88] Yao Yong, " Handprinted Chinese Character Recognition via Neural Networks," *Pattern Recognition Letters,* 7, pp.19-25, January 1988.

[Yu94] X. Yu, N. K. Loh, and W. C. Miller, " Training Hard-limiting Neurons Using Back-Propagation Algorithm by Updating Steepness Factors," *IEEE International Conference on Neural Networks,* pp. 526-530, 1994.

[Zha94] D. Zhang, M. Kamel, and M. I. Elmasry, " A Training Approach Based on Linear Separability Analysis for Layered Perceptrons," *IEEE International Conference on Neural Networks,* pp. 517-520, 1994.

[Zek92] S. Zeki, " The Visual Image in Mind and Brain," *Scientific American,* Sept. 1992.

# VITA

Parvin Hashemian received her bachelor of arts degree from New York University in February 1980. She obtained her master of science degree in Computer Science from Columbia University in May 1981. While raising three children, she pursued her doctoral studies in Computer Science. She is currently a doctoral candidate in the Department of Computer Science at Louisiana State University and will receive her doctor of philosophy degree in December 1996. Her current interest are in Neural Networks, Artificial Intelligence, Pattern Recognition, and Database Management Systems.

# DOCTORAL EXAMINATION AND DISSERTATION REPORT

**Candidate:** Parvin Hashemian

**Major Field:** Computer Science

**Title of Dissertation:** Modular Approaches for Designing Feedforward Neural Networks to Solve Classification Problems

**Approved:**

Subhash Kak
**Major Professor and Chairman**

David Fogel
**Dean of the Graduate School**

**EXAMINING COMMITTEE:**

Donald W

Jianhua Chen

Sukhamay Kundu

M Manry

**Date of Examination:**

June 3, 1996