

991

# **PRUNING IN NEURAL NETWORKS**

*Dissertation submitted to  
Jawaharlal Nehru University  
in partial fulfilment of the requirements  
for the award of Degree of*

**MASTER OF TECHNOLOGY**

in

**COMPUTER SCIENCE & TECHNOLOGY**

by

**D. SAMBASIVA RAO**

**SCHOOL OF COMPUTER AND SYSTEMS SCIENCES  
JAWAHARLAL NEHRU UNIVERSITY  
NEW DELHI - 110 067**

JANUARY 1993


## CERTIFICATE

This is to certify that the thesis entitled "PRUNING IN NEURAL NETWORKS", being submitted by me to Jawaharlal Nehru University in partial fulfilment of the requirements for the award of the degree of Master of Technology, is a record of original work done by me under the supervision of Prof. K.K. Bharadwaj, School of Computer and Systems Sciences, during the Monsoon semester, 1992.

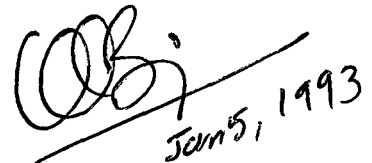
The results reported in this thesis have not been submitted in part or full at any other University or Institution for the award of any degree etc.



SAMBASIVA RAO DASARI



Dr. R. G. Gupta  
Professor & Dean, 5/11/93  
SC&SS, J.N.U.,  
New Delhi - 110 067.



Jan 5, 1993

Dr. K. K. Bharadwaj  
Professor,  
SC & SS, J.N.U.,  
New Delhi - 110 067.

## ACKNOWLEDGEMENTS

*I am very much indebted to my supervisor Prof. K. K. Bharadwaj, School of Computer and Systems Sciences, J N U, for his valuable guidance, constructive criticism, encouragement and unreserved cooperation at each stage of this thesis work.*

*I express my sincere thanks to Prof. R. G. Gupta, Dean, School of Computer and Systems Sciences, J N U, for providing an opportunity to undertake this work.*

*I extend my thanks to Dr Sudhir Kaicker, Director, Computer Centre, School of Computer and Systems Sciences, J N U, for his suggestions in coding.*

*I also express special thanks to my friends Rajesh and Shiva, for their help in getting the material from the libraries of Arizona State University and the University of Kansas. I am grateful to my friends Mathew, Ramana and Krishna.*

*I am thankful to all the faculty and other authorities of SC&SS who are directly or indirectly in completing this work.*



**SAMBASIVA RAO DASARI**

*to my  
parents and  
Rekha teacher*

## ABSTRACT

*A continuing question in artificial neural network research is the size of network needed to solve a particular problem. If training is started with too small a network for the problem no learning can occur. If a network that is larger than required is used then processing is slowed, particularly on a conventional Von Neumann Computer. In this work we start with a larger net, then the redundant connections are removed (pruned) to bring it to the desired state. The approach followed in this work is to estimate the sensitivity of the error function to the exclusion of each connection, then prune low sensitivity connections. Neural networks applications, handwritten character recognition and pattern classification problem, were implemented using Backpropagation algorithm in C and the experimental results were studied with and without pruning.*

# CONTENTS

## Chapter 1 Artificial Neural Networks

1.1	Introduction	1
1.2	Definition	2
1.3	Associative Memory	3
1.4	Historical Perspective	4
1.5	Biological - Artificial Neural Nets Disanalogies	6
1.6	Neural Networks Vs. Expert Systems	7
1.8	Biological Neuron	8
1.8	Applications	9
1.9	Drawbacks	10

## Chapter 2 The Backpropagation

2.1	Learning in Neural nets	11
2.2	Feed Forward nets	12
2.3	The Backpropagation Algorithm	13
2.4	Feed Forward Calculations	14
2.5	Backpropagation Training	16
2.6	Drawbacks	21

## Chapter 3 Pruning in Neural Networks

3.1	Introduction	22
3.2	Background	23
3.3	Different Approaches for Pruning	24
3.4	Implemented Pruning Procedure	25

## Chapter 4 Implementation and Results

4.1	Handwritten Character Recognition	30
4.2	Pattern Classification Problem	31
4.3	Implementation of Handwritten Pattern Recognition	31
4.4	Implementation of Pattern Classification Problem	33
4.5	Results	34

CONCLUSION	41
------------	----

REFERENCES	42
------------	----

CHAPTER 1  
ARTIFICIAL NEURAL NETWORKS

---

*"We already have intelligent machines - humans! But before we agree to add humans to the set of machines we need to ask what is man and what is machine"*

Marvin Minsky

## 1.1 Introduction

In the last decade, neural networks have received a great deal of attention and are being touted as one of the greatest computational tools ever developed. Much of the excitement is due to the apparent ability of neural networks to imitate the brain's ability to make decisions and draw conclusions when presented with complex, noisy, irrelevant and or partial information. Furthermore, at some primitive level, neural networks appear able to imitate the brain's 'creative' processes to new data or patterns.

One of the reasons humans and computers are so useful to each other is that they are so different. Computers do well what we do not, and vice versa. Computers are logical and precise; we are not. Computers can store and retrieve vast amounts of detail without error; we cannot. But humans can handle guess, ambiguity, and integrate information from many sources.

Artificial neural net models or simply neural nets go by many names such as connectionist models, parallel distributed processing models, and neuromorphic systems. All these models attempt to achieve good performance via dense interconnection of simple computational elements. In this respect, artificial neural net structure is based on our present understanding of biological nervous systems. Neural net models have greatest potential in areas such as speech and image recognition where many hypotheses are pursued in parallel, high computation rates are required, and the current best systems are far from equally human performance. Instead of performing a program of instructions sequentially as in a Von Neumann computer, neural net models explore many computing



hypotheses simultaneously using massive parallel nets composed of many computational elements connected by links with variable weights. Simply, neural networks are algorithms for optimization and learning based loosely on concepts inspired by research into the nature of the brain.

## 1.2 Definition

A neural network is a computing system that imitates intelligent behavior; it is made up of simple, highly connected processing elements and processes information by its dynamic state response to external inputs.

They generally consist of five components:

1. A directed graph known as the network topology whose arcs we refer to as links.
2. A state variable associated with each node.
3. A real-valued weight associated with each link.
4. A real-valued bias associated with each node.
5. A transfer function for each node which determines the state of a node as a function of
  - a) its biases
  - b) the weights,  $w_i$ , of its incoming links, and
  - c) the states,  $x_i$ , of the nodes connected to it by these links. This transfer function usually takes the form  $f(\sum w_i x_i - Q)$  where  $f$  is either a sigmoid or a step function.

Computational elements or nodes used in neural net models are nonlinear, are typically analog, and may be slow compared to modern digital circuitry. The simplest node sums  $n$  weighted inputs and passes the result through a nonlinearity as shown in fig (1.1). The node is characterised by an internal threshold or offset  $Q$  and by the type of nonlinearity. Fig (1.2) illustrates three common types of nonlinearities; which limits, threshold logic elements, and sigmoid nonlinearities. More complex nodes may include temporal integration or other types of time dependencies and more complex mathematical operations than summation [7].

## Paradigms

The different paradigms that were developed till recently for the neural networks were:

### 1.3 Associative memory

Associative memories are similar to human memory in that they recall complete situations from partial information. Associate memory plays an important role in pattern recognition and information-processing applications.

The two varieties of associate memory that are of interest in neural networks are auto-and heteroassociate . Autoassociative memories map pieces data to themselves, memorising specific information. The most common autoassociative neural paradigms are the hopfield network, brain-state-in-a-box, and adaptive resonance theory, although several researchers have explored similar

paradigms. Autoassociative memories are often used to reconstruct partial or error-prone patterns into their original forms or to optimize certain operations research problems. Examples of autoassociative applications include character recognition, picture conclusion for airport security, retina recognition, and signal reconstruction.

Heteroassociative memories map one set of patterns to another. Patterns, classifications, and real numbers are often stored as outputs to heteroassociative networks. Neural paradigms that behave as heteroassociative memories include the BAM, Kohonen feature map, and some of the statistical paradigms. Examples include target classification, financial-trend analysis, and process monitoring.

## 1.4 Historical Perspective

Experiments have found the brain and nervous system to be difficult to observe and perplexing in organization. In short, the powerful methods of scientific inquiry that have changed the view of physical reality have been slow in finding application to the understanding of humans themselves.

With the progress in neuroanatomy and neurophysiology, psychologists are developing models of human learning. In 1943 McCulloch and Pitts, in their attempts to simulate the nervous cells by artificial automata-the formal neurons-that a network of such formal neurons was capable of simulating a Turing Machine. Another model, which has proved most fruitful, was that of D.O. Hebb, who in 1949 proposed a learning law that became the starting point for artificial neural networks training algorithms [5]. In the 1950s and 1960s, a group of

researchers combined these biological and psychological insights to produce the first artificial neural networks [10]. Initially implemented as electronic circuits, they were later converted to the more flexible medium of computer simulation. Marvin Minsky, Frank Rosenblatt, Bernard Widrow, and others developed networks consisting of a single layer of artificial neurons. Often called perceptrons, they were applied to such diverse problems as weather prediction, electrocardiogram analysis, and artificial vision. It seemed for a time that the key to intelligence had been found; reproducing the human brain was only a matter of constructing a large enough network. Marvin Minsky, carefully applying mathematical technique developed rigorous theorems regarding network operation. His research led to the publication of the book *Perceptrons* (Minsky and Papert 1969), in which he and Seymour Papert proved that the single-layer networks then in use were theoretically incapable of solving many simple problems. Then a few scientists such as Teuvo Kohonen, Stephen Grossberg, and James Anderson continued their efforts during 1970s and 1980s [7]. Gradually, a theoretical foundation emerged, upon which the more powerful multilayer networks of today are being constructed. Networks are now routinely solving many of the problems that Minsky posed in his work.

Backpropagation, invented independently in three separate efforts Werbers 1974; Parker 1982; and Rumelhart, Hinton and Williams 1986, provided a systematic means for training multilayer networks, overcoming Minsky's limitations.

## 1.5 Biological - Artificial Neural Nets Disanalogies

### Neural nets

1. Though the cycle time of 80286 or 80386 is generally 0.03 - 0.10 micro Sec, the time taken to perform a task is more when compared to BNNs because of the lack of the parallelism.
2. The connections among neurodes can have either positive or negative weights. These weights correspond to excitatory and inhibitory neural connections, so Eccles\* law is violated.
3. The information of activation between connections is passed as a d.c. level.
4. The number of neurons involved in the implementation are very less (few dozen to several hundred dozens).
5. They are programmed.
6. Addressable memory.

### Biological nets

Though the cycle time of BNNs is 10-100 milli Sec, the brain is still able to perform some tasks faster than the fastest digital computer because of the brain's massive parallel architecture.

The Eccles law is obeyed.

Since a train of pulses carry the information between synapses, the activity is passed as a.c. level.

The number of neurons in a BNNs are of the order of billion.

They are not programmed.

Associate memory

## 1.6 Neural Networks Versus Expert Systems

An expert system is a software based system that describes the behavior of an expert in some field by capturing the knowledge of one or more experts in the form of rules and symbols.

One of the major problems with expert systems is with the acquisition and coding of the expert knowledge. A related problem is the evaluation of the accuracy and completeness of the encoded knowledge thus acquired. The different problems associated with acquiring expert knowledge are in language. If all the rules needed to characterise the knowledge to be represented are not present, sooner or later (probably sooner) the expert system will fail. If an incorrect, fuzzy or noisy data is presented to an expert system it may give wrong answer.

But these are precisely the areas in which neural networks shine. The neural networks can be given, within limits, some fuzzy, noisy data and still get the right answers. We can even, within limits, lie to them and come out all right [11].

An expert system might be a better approach if you don't have enough information (patterns) to train a neural network. In some applications, because of simplicity, using a rule-based system is better approach.

## 1.7 Biological Neuron

A neuron is a cell that is a part of the nervous system and that conducts messages to and from the brain. Classically, such a neuron consists of a cell body, a number of finger like process of the cell called dendrites, and one very long thin process the axon (fig 1.3). The axon may or may not be covered with a myelin sheath, the cell body and dendrites are unmyelinated; the dendrites, which form the input processes, have a graded electrical response; the axon, which forms the output process transmits trains of electrical pulses [10]. An example of such a neurone is the anterior horn cell, or motor cell of the spinal cord. Such 'classical' cells are really rather exceptional, and properties that contradict one or more of the above statements.

Neurones are not the sole class of cells making up nervous tissue. The other great class of cells is the neuroglia. Although, neurones may be quite densely packed, there is of course a good deal of space around them. This space is not just filled with extracellular fluid, it is almost entirely occupied by the cell bodies and processes of the neuroglia, which have been thought of as a sort of framework or scaffolding on which the neurons are arranged. Intercellular recordings from glia have not shown them to be electrically excitable.

Human brain has an estimated 10-500 billion neurons. According to Stubbs, neurons are arranged into about 1000 main modules, each with about 500 neural networks. Each network has on the order of 100,000 neurons. The axon of each neuron connects to about 100 other neurons [10].

## 1.8 Applications

The recent studies reveal that due to the immense computational power and the self learning capabilities of the neural networks, they were finding applications in the following:

- \* analysis of medical tests
- \* circuit board problem diagnosis
- \* EEG waveform classification
- \* picking winners of horse races
- \* predicting performance of students
- \* analysis of loan applications
- \* stock market prediction
- \* military target tracking and recognition
- \* process control
- \* oil exploration
- \* psychiatric evaluations
- \* optimizing scheduled maintenance of machines
- \* composing music
- \* spectral analysis
- \* optimizing raw material orders
- \* selection of employees
- \* speech recognition
- \* text-to-speech conversion
- \* selection of criminal investigation targets
- \* analysis of polygraph examination results

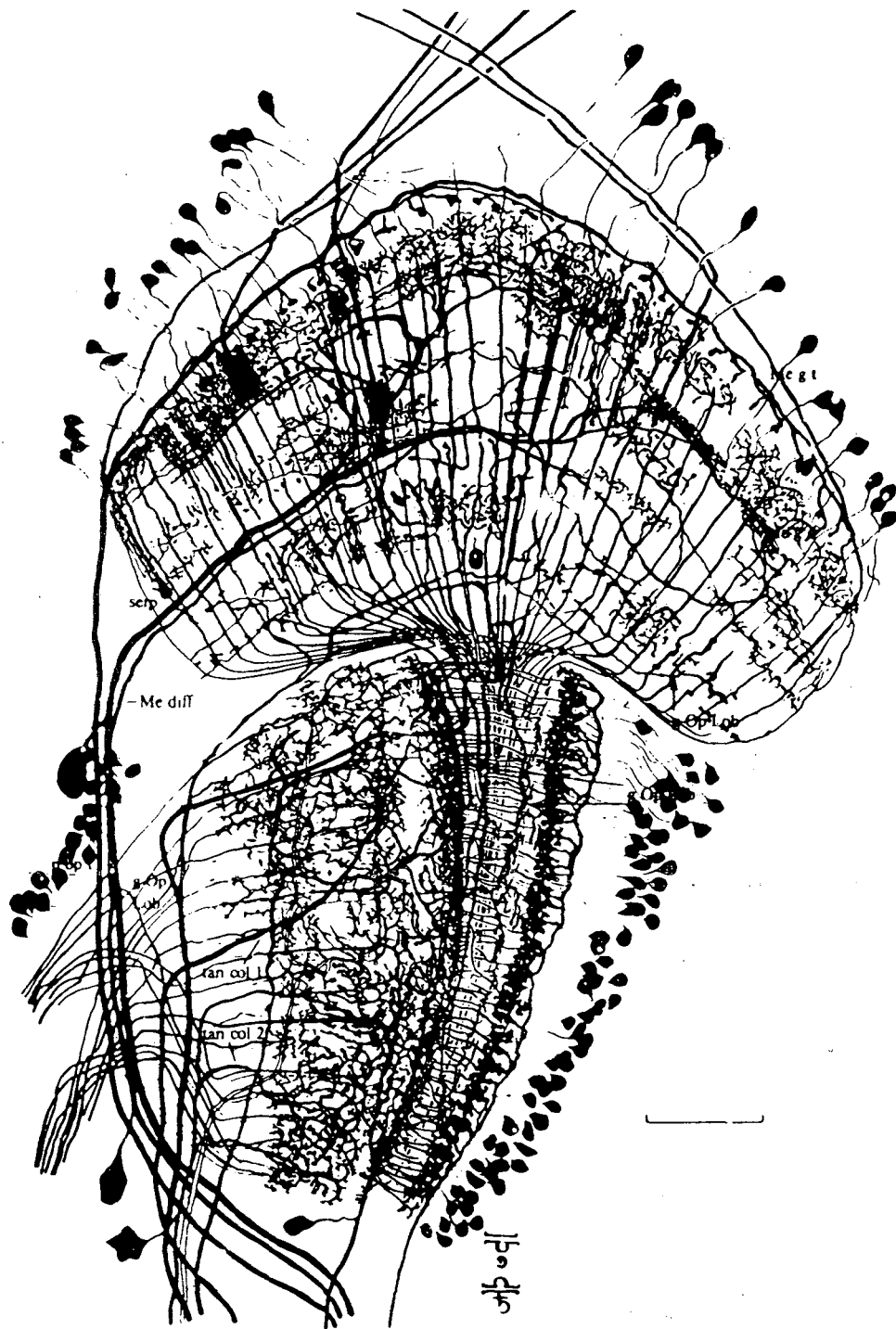


- \* optimization of antenna array patterns
- \* optical character recognition
- \* modeling the operation of the brain

Neural nets research is published in books and journals from such diverse fields that even the most diligent researcher is hard pressed to remain aware of all significant work. The above stated applications are catching one's imagination of yet another area of explore.

## 1.9 Drawbacks

- (1) Neural nets are particularly in appropriate for problems, requiring precise calculations. It is probably, never successfully to balance a checkbook with a neural network.
- (2) It is almost always true that the neural network portion of the solution is only a relatively small part of the overall system
- (3) Neural nets cannot be implemented if the available training data (patterns) is less.



Network from the 'visual brain' of *Musca domestica* (the common housefly). (From N. J. Strausfeld and D. R. Nässel, Neural architectures serving compound eyes, in *Comparative Physiology and Evolution of Vision in Invertebrates*, Ed. H. Autrum, Handbook of Sensory Physiology, Vol. VII/6B, Springer-Verlag, Berlin, 1981. Reproduced by permission of Springer-Verlag.)

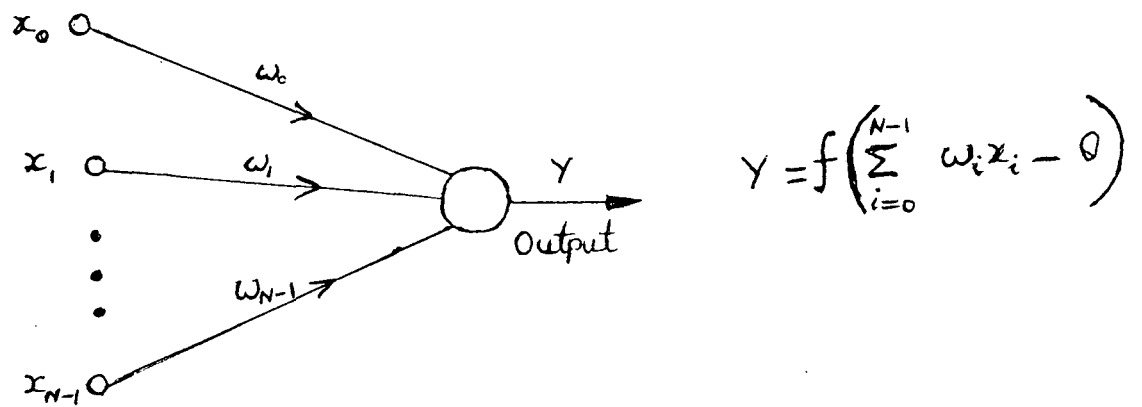


fig. 1.1 Computational element (Node)

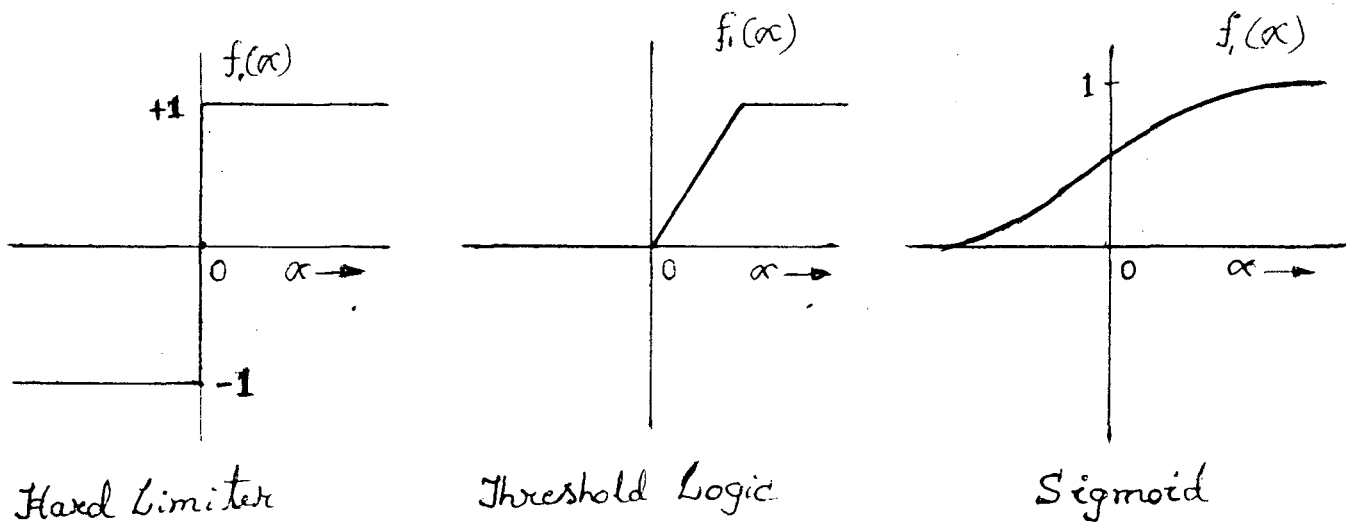
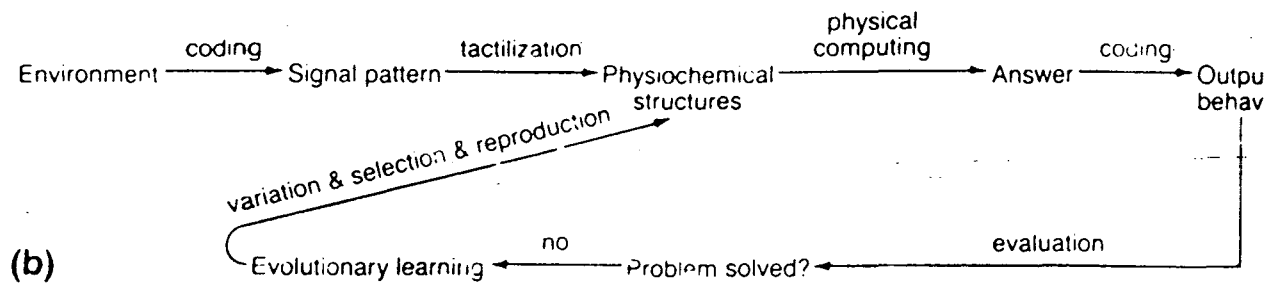
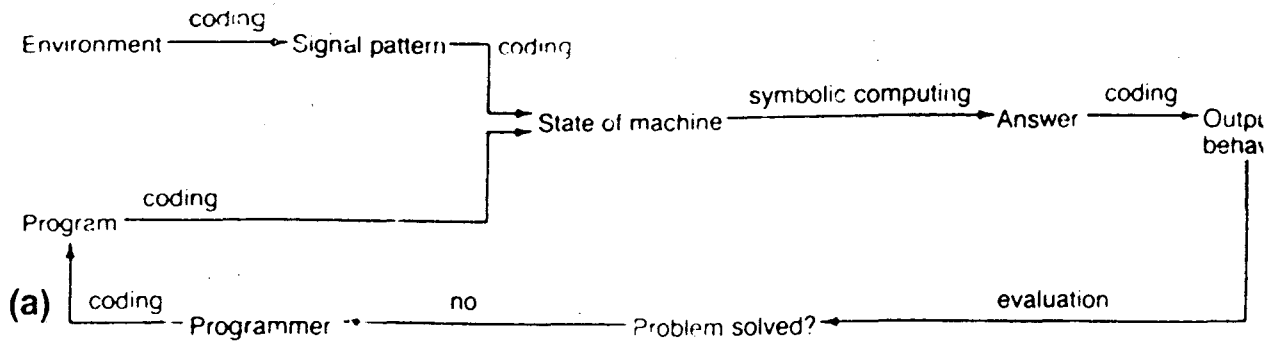


fig. 1.2 The different nonlinearities



(b)  
Computing in (a) Machine and (b) Brain

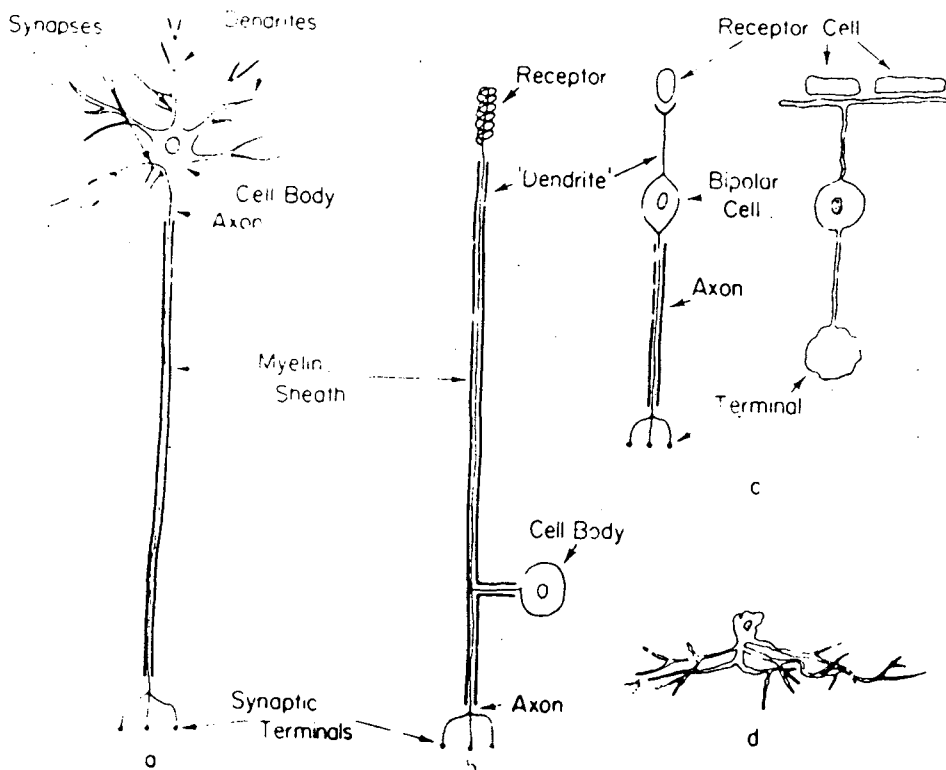


Fig. 1.3 (a) A 'classical' neuron and its constituent parts; the axon may be myelinated or unmyelinated. (b) A monopolar neuron from the somatic system; the 'dendrite' is structurally indistinguishable from an axon and conducts impulses. (c) Bipolar cells. That from the auditory nerve (left) conducts impulses in its axon; that from the retina (right) shows only graded responses. (d) Amacrine cell from the retina. This structurally axonless cell nevertheless fires impulses

**CHAPTER 2**  
**THE BACK-PROPAGATION**

---

*"Neural nets learn from the bottom up.  
Symbolic systems learn from the top down"*

**Elaine Rich**

## 2.1 Learning in Neural Networks

Artificial neural networks can modify their behavior in response to their environment. This factor, more than any other, is responsible for the interest they have received. Shown a set of inputs (perhaps with desired outputs), they self-adjust to produce consistent responses.

A network is trained so that application of a set of inputs produces the desired (or at least consistent) set of outputs. Each such input (or output) set is referred to as a vector. Training is accomplished by sequentially applying input vectors, while adjusting network weights according to a predetermined procedure. During training, the network weights gradually converge to values such that input vector produces the desired output vector.

### Supervised Training

Training algorithms are categorized as supervised and unsupervised. Supervised training requires the pairing of each input vector with a target vector representing the desired output; altogether these are called a training pair. Usually a network is trained over a number of such training pairs. An input vector is applied, the output of the network is calculated and compared to the corresponding target vector, and the difference (error) is fed back through the network and weights are changed according to an algorithm that tends to minimize the error. The vectors of the training set are applied sequentially, and errors are calculated and weights adjusted for each vector, until the error for the entire training set is at an acceptably low level.

## Unsupervised Training

Despite many application successes, supervised training has been criticised as being biologically implausible. Unsupervised training is a far more plausible model of learning in the biological system. Developed by Kohonen (1984) and many others, it requires no target vector for the outputs, and hence, no comparisons to predetermined ideal responses [5]. The training set consists solely of input vectors. The training algorithm modifies network weights to produce output vectors that are consistent; that is, both application of a vector that is sufficiently similar to it will produce the same pattern of outputs. The training process, therefore, extracts the statistical properties of the training set and graphs similar vector into classes. Applying a vector from a given class to the input will produce a specific output vector, but there is no way to determine prior to training which specific output pattern will be produced by a given input vector class. Hence, the outputs of such a network must generally be transformed into a comprehensible form subsequent to training process.

Another type of learning rule that falls between unsupervised learning and supervised learning is reinforcement learning. In this kind of learning, an external observer gives a response as to whether the network response is good or not. The learning rule of Boltzmann machine is based on the Stochastic process, which constructs distributed representations of the reference patterns with the simulated annealing technique.

## 2.2 Feed Forward Networks

A feed forward network is one whose topology has no closed paths. Its input nodes are the ones with no arcs to them, and its output nodes have no arcs

away from them. All other nodes are hidden nodes. The operation of feed forward network consists of calculating outputs given a set of inputs. A layered feed forward network is one such that any path from an input node to an output node traverses the same number of arcs. The  $n$ th layer of such a network consists of all nodes which are  $n$  traversals from an input node. A hidden layer is fully connected if each node in layer  $j$  connected to all nodes in layer  $j+1$  to  $j$ .

Layered feed forward networks have become very popular for a few reasons:

1. They have been found in practice to generalize well, i.e., when trained on a relatively sparse set of data points, they will often provide the right output for an input not in the training set [4].
2. A training algorithm, backpropagation, exists which can often find a good set of weights (and biases) in a reasonable amount of time.

### **2.3 The Backpropagation Algorithm**

Back propagation is a systematic method for training multilayer artificial neural networks. Despite its limitations, backpropagation has dramatically expanded the range of problems to which artificial neural networks can be applied, and it has generated many successful demonstrations of its power.

Backpropagation has an interesting history. Rumelhart, Hinton and Williams (1986) presented a clear and concise description of the backpropagation



algorithm [2]. No sooner was this work published than Parker (1982) was shown to have anticipated Rumelhart's work. Shortly after this, Werbos (1974) was found to have described the method still earlier.

The backpropation algorithm described below is a generalisation of Least Mean Squares algorithm. It uses a gradient search technique to minimize a cost function equal to the mean square difference between the desired and the actual net outputs [7]. It requires contineous differentiable non-linearities. The following algorithm assumes a sigmoid logistic nonlinearity.

The type of network that is referred to here is a layered, feed forward network of units with a deterministic semi- linear output function described by Rumelhart, Hinton and Williams. Each layer feeds only to the layer directly above it. There are a number of internal layers, the first one receiving input from some external source, and a layer of output units at the top, the number depending on the type of output desired.

## 2.4 Feed Forward Calculations

The net input,  $net_j$ , to a unit  $j$  is a linear function of the outputs,  $x_i$ , of all the units,  $i$ , that are connected to  $j$

$$\text{net-input} = net_j = \sum_i w_{ij}x_i + b_j$$

where  $w_{ij}$  is the weight of the connection from unit  $i$  to unit  $j$ , and  $b_j$  is a bias weight at unit  $j$ .

The biases always have an output of 1. They serve as threshold units for the layers to which they are connected, and the weights from the bias neurodes to each of the neurodes in the following layer are adjusted exactly like the other weights.

The squashing function or the output of a unit,  $x_j$ , is a real-valued, non-linear function of its net input

$$x_j = \frac{1}{1 + e^{-net_j}}$$

The output is bounded by (0,1) and  $x_j$  quickly approaches one or zero as  $net_j$  approaches  $\pm \infty$  [3].

The squashing function (fig 2.2) can be viewed as performing a function similar to an analog electronic amplifier. The gain of the amplifier is analogous to the slope of the squashing function, or the ratio of the change in output for a given change in input. As in the fig, the slope of the function (gain of the amplifier) is greatest for total (net) inputs near zero. This serves to mitigate problems caused by the possible dominating effects of large input signals [11]. Other functions can be used as long as they are continuous and possess a derivative at all points. Functions such as  $\text{Sin}(x)$ ,  $\text{Tanh}(x)$  have been used, but the sigmoid has the additional advantage of providing a form of automatic gain control [5].

## 2.5 Backpropagation Training

During the training phase, the feed forward output state calculation is combined with backward error propagation and weight adjustment calculations that represent the network's learning or training.

Central to the concept of training a network is the error. Rumelhart and McClelland [2] defined an error term as the difference between the output value an output neurode is supposed to have (target value  $t_{pj}$ ), and the value it actually has as a result of the feed forward calculations,  $x_{pj}$ . The error term is defined for a given pattern and summed over all output neurodes for that pattern. The error is found on a neurode-by-neurode basis over the entire set (epoch) of patterns, rather than on a pattern-by-pattern basis. We sum the error over all neurodes, giving a grand total for all neurodes and all patterns.

Since the number of patterns in the training set can vary, and we want some standardized error value that allows, us to compare we find the average sun-squared error value by dividing the grand total by the number of patterns.

$$E = 0.5 \sum_p \sum_j (t_{pj} - x_{pj})^2$$

where the inner summation is over all neurons that are considered as output units of the net, and the outer sum is over patterns of the training set. Because 0.5 is a constant, we delete it from over calculations.

**Step 1. Initialize weights and offsets.**

Set all weights and node offsets to small random values.

**Step 2. Present Input and Desired Outputs**

Present a continuous valued input vector  $x_0, x_1, \dots, x_{M-1}$  and specify the desired targets  $t_0, t_1, \dots, t_{N-1}$ . If the net is used as a classifier then all desired outputs, are typically set to zero except for that corresponding to the class the input is from. That desired output is 1. The input could be new on each trail or samples from a training set could be presented cyclically until weights stabilize.

**Step 3. Calculate Actual Outputs**

Use the sigmoid nonlinearity from above and formulas to calculate outputs  $x_0, x_1, \dots, x_{M-1}$ .

**Step 4. Adapt Weights**

Use a recursive algorithm starting at the output nodes and working back to the first hidden layer. Adjust weights by

$$| w_{ij}(t+1) = w_{ij}(t) + \eta \delta_j x_{ij} |$$

In this equation  $w_{ij}(t)$  is the weight from hidden node  $i$  or from an input to node  $j$  at time  $t$ ,  $x_{ij}$  is either the output of node  $i$  or is an input,  $\eta$  is a gain term, and  $\delta_j$  is an error term for node  $j$ . If node  $j$  is an output node, then

$$\delta_j = x_j(1-x_j)(t_j - x_j),$$

where  $t_j$  is the desired output of node  $j$  and  $x_j$  is the actual output.

If node  $j$  is an internal hidden node, then

$$\delta_j = x_j (1-x_j) \sum_k \delta_k w_{jk}$$

where  $k$  is over all nodes in the layers above node  $j$ . Internal node thresholds are adapted in a similar manner by assuming they are connection weights on links from auxiliary constant-valued inputs. Convergence is sometimes faster if a momentum term is added and weight changes are smoothed by

$$\begin{aligned} w_{ij}(t+1) &= w_{ij}(t) + \eta \delta_j x_i + \alpha(w_{ij}(t) - w_{ij}(t-1)), \quad 0 < \alpha < 1 \\ &= w_{ij}(t) + \eta \delta_j x_i + \alpha \Delta(w_{ij}(t)) \end{aligned}$$

Step 5. Repeat by Going to Step 2

The kind of weight updating,

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_j x_i$$

sometimes gets caught in what are called *local energy minima*. It is like a bowl-shaped surface with a lot of little bumps and ridges in it in 3D as shown in fig. 2.3.

The above algorithm is repeated until the error approach the desired value. The error minimization process is analogous to minimizing the energy of our position in the bumpy ridge lined bowl. Ideally, we'd like to move our position (perhaps) marked by a very small ball bearing) to the bottom of the bowl where the energy is minimum; this position is called the 'globally optimal solution". Depending on how much or how little we can move the ball bearing at one time, however, we might get caught in some little depression or ridge that we can't get out of. This situation is most likely with small limits on each individual movement, which corresponds to small value of  $\eta$  [11]. Again  $\eta$  is the training rate coefficient serving to adjust the size of the average weight change.

The situation can be helped by using the momentum of our ball bearing. We take into account its momentum (previous movement) by a *momentum factor* that we label  $\alpha$ [11]. If  $\alpha$  is 0 , then smoothing is minimum; the entire weight adjustment comes from the newly calculated change. If  $\alpha$  is 1.0, the adjustment is ignored and the previous one is repeated. Between 0 and 1 is a region where the weight adjustment is smoothed by an amount proportional to  $\alpha$  [5].

Another parameter to be experimented is the number of iterations of the learning of the training set needed to give an accepted average mean-squared error. First, we pick a reasonable average mean-square error 0.04 or 0.05. If it won't get trained in few thousands of iterations, we should probably adjust the value of learning rate and momentum factor.

It is found that the number of hidden layer neurodes, to start with in many cases, can be obtained by taking the square root of the number of input plus

output neurodes, and adding a few. Hidden layers act as layers of abstraction, pulling features from inputs. Increasing the number of sequential hidden layers augments the processing power of the neural network but significantly complicates training and intensifies black-box effects (errors are more difficult to trace). Adding hidden layers will increase both the time and the number of training examples necessary to train the network properly. As a rule of thumb, start with one hidden layer and add more as required.

Another method of increasing a neural network's processing power is to add multiple slabs within a single hidden layer. This creation of sets of neurons that act as feature detectors for perceptual input. D. Hubel and T. Wiesel discovered feature detectors for lines with different orientations within cats [14]. The multiple parallel slabs may use different types and numbers of nodes. This architecture attempts to force the slabs to extract different features simultaneously. Most applications do not require multiple slabs within a hidden layer, but this architecture provides an alternative to single hidden-layer ones.

After selecting the number of layers for the network, the next step is to determine the size (in number of nodes) of each layer. Because of the similar reasoning involved in setting layer sizes for the different neural paradigms, they are grouped together. The input layer presents data to the network. The number of input nodes is calculated from the number of data sources and nodes required to represent each source. Often the most difficult design decision is to ascertain the correct data sources and nodes required to represent each source. Often the most difficult design decision is to ascertain the correct data sources: volumes of unprocessed or spurious data hinder training, but missing data may preclude it.

## 2.6 Drawbacks

Despite its tremendous success, backpropagation algorithm is found to have some drawbacks. For one, there is the 'scaling problem'. Backpropagation works well on simple training problems. However, as the problem complexity increases (due to increased dimensionality and/or greater complexity of the data), the performance of backpropagation falls off rapidly. The performance degradation appears to stem from the fact complex spaces have nearly global minima which are sparse among the local minima. With a high enough gain (or momentum), backpropagation can escape these local minima. However, it leaves them without knowing whether the next one it finds will be better or worse. When the nearly global minima are well hidden among the local minima, backpropagation can end up bouncing between local minima without much overall improvement, thus making for very slow training.

A second shortcoming of backpropagation is that a gradient requires differentiability. Therefore, backpropagation cannot handle discontinuous optimality criteria or discontinuous node transfer functions. This precludes its use on some common node types and simple optimality criteria [13].



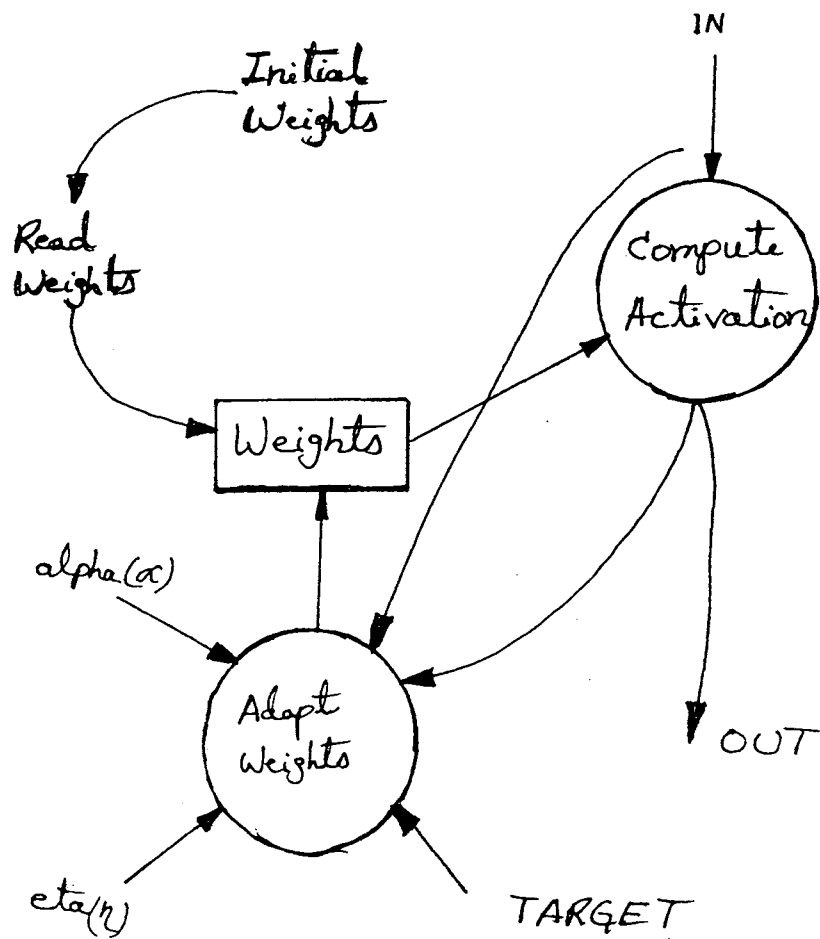


Fig. Processing in Back-Propagation training algorithm

### 3.1 Introduction

When constructing an artificial neural network the designer is often faced with the problem of choosing a network of the right size for the task. In theory atleast, if a problem is solvable with a network of a given size, it can also be solved by a larger net which embeds the smaller one, with (hopefully) all the redundant connections or synapses, having a zero strength. However, the learning algorithm will typically produce a different structure, with nonvanishing synaptic weights spreading all over the net, thus obscuring the existence of a smaller size neural net solution[1].

The advantages of using a smaller neural network:

- \* Since the cost of computation grows (almost) linearly with the number of the synaptic connections, a smaller net will be more efficient in both forward computations and learning.
- \* A network which is too large will tend to memorize the training patterns and thus have poor generalization ability.
- \* There is always the hope that a smaller net will exhibit a behavior that can be described by a simple set of rules.

However, a network which is too small may never solve the problem (a simple example being the XOR function which cannot be implemented with a single neuron), while a larger net may even have the advantage of a faster learning rate. Thus it makes sense to start with a large net and then reduce its

size. Several researchers have proposed methods to accomplish this reduction, and the same approach, i.e., net pruning is pursued in this work.

### **Definition**

Pruning is the name given to the process of examining a solution network, determining which connections or units are not necessary to the solution and removing those connections or units.

Assuming that the network has learned, then an arbitrary setting of weights to zero (which is equivalent to eliminating the synapse that goes from neuron  $j$  to neuron  $i$ ) or the arbitrary removal of units, will typically result in an increase of the error  $E$ . Efficient Pruning means finding the subset of the weights or units that, when set to zero, will lead to the smallest increase in  $E$  [1]. Theoretically, this can be done by training the net under all possible subsets of the set of synapses. However, this exhaustive search is computationally infeasible, unless dealing with a very small net and few training patterns.

## **3.2 Background**

Siestna and Dow [3] have analyzed pruning of the networks by examining all units under the presentation of the entire training data. They removed each unit that did not change state, or replicated another unit. Unfortunately, scaling up of this technique to large nets and many patterns will result in a prohibitively long learning process. Hanson and Pratt [1989] have studied the idea of weight decay, in which the weights that do not have much influence on decreasing the

error while learning will experience an exponential time delay which is equivalent to adding a penalty term to the error function. They have extensively experimented with various forms of weight decay, and concluded that these penalties do not seem to minimize the number of units. Mozer and Somolensky [1989] have introduced the idea of estimating the sensitivity of the error function to the elimination of each unit [1].

### **3.3 Different Approaches for Pruning**

The different approaches suggested by researchers for pruning neural networks were:

(1) **Noncontributing Units**

These are units which either have approximately constant output across the training set, or have outputs across the training set which mimic the outputs of another unit. These units can be pruned and the weights given to their outputs redistributed in such a way as to make almost no change to the network's performance over the training set.

(2) **Unnecessary Information Units:**

In this approach, pruning is removing units which are independent of the other units in the layer but give information which is not required at the next layer. Removing units which contribute unnecessary information to the next layer may lead to the outputs of the pruned layer being linearly inseparable with respect to the classes or outputs of the layer above.

(3) Another possible approach to identifying in-essential units would be one form of sensitivity analysis. In this approach the activation of a unit is not to zero for all training set inputs and the effect on network output expressed. This would identify units which either had activation always near zero, or whose activation was given little weight by units in the next layer. This approach would not remove a unit which had activation parallel to another unit. The units removed in this approach are likely to be the same as, or a subset of, those that are removed by 'Non contributing pruning approach', and will definitely be a subset of those removed by 'Unnecessary information pruning' approach.

(4) Another form of pruning technique using sensitivity analysis is setting weights to zero. This is equivalent to removing connections between units, rather than removing entire units. The connection to the removed are found by sensitivity analysis. This approach is followed in this dissertation work for pruning.

(5) Some form of pruning could be implemented as part of the 'training process'. This could be done as a dynamic form of sensitivity analysis, with a strictness parameter that increased as a network approach solution, or by adding a cost, related to the size or number of connections, to the error cost function. These are completely different approaches to the ones we have described above, which apply solely to trained solution network.

### **3.4 Implemented Pruning Procedure**

In this work the pruning procedure suggested by Ehud d. Karnin who is in IBM Science and Technology, Haifa, is followed[1]. In this we pursued the idea

of evaluating sensitivities for connections. The approach followed does not require any change of the error function. Thus the two tasks of training the network and evaluating the candidates for pruning are completely separated, which means this has the advantage of learning process not interfering any extraneous process.

Our approach to pruning is to estimate sensitivity of the error function  $E$  to the exclusion of each synapse(connection), then prune the low sensitivity connections. The sensitivity  $S_{ij}$  is defined as

$$S = \frac{E(w^f) - E(0)}{w_f - 0} w^f$$

where  $w^f$  is the (final) value of the connection upon the completion of the training phase. And  $w = w_{ij}$ ;  $E$  is expressed as a function of  $w$ , assuming that all other weights are fixed (at their final states, upon completion of learning).

A typical learning process does not start with  $w=0$ , but with some small (often randomly chosen) initial value  $w^i$ . Fig 3.1 depicts an example of the decrease in  $E$ , as a function of the weight  $w$ , during the training phase.

Since we do not know  $E(0)$ , we will approximate the slope of  $E(w)$  (when moving from 0 to  $w^f$ ) by the average slope measured between  $w^i$  and  $w^f$ , namely

$$S = \frac{E(w^f) - E(w^i)}{w^f - w^i} w^f$$

The initial and final weights,  $w^i$  and  $w^f$ , respectively, are quantities that are readily available during the training phase. However, for the numerator, it was

implicitly assumed that only one weight, namely  $w$ , had been changed, while other weights remained fixed. This is not the case during normal learning. To elaborate, consider the example of a network with only two weights, denoted  $u$  and  $w$  (the extension to more weights will become obvious). For this case the numerator of  $S$  is

$$E(u^f, w^f) - E(u^f, w^i)$$

i.e only the contribution due to the change in  $w$  is taken into account. Fig 3.2 clarifies the situation.

The error  $E(u, w)$  is illustrated by constant value contours. The initial point in the weight space is designed by  $I$  in Fig 3.2, and the learning path is the dashed line from  $I$  to  $F$ , the final point. For a precise evaluation of  $S$ , the numerator of  $S$  in the first equation can be evaluated as

$$E(w = w^f) - E(w = 0) = \int_A^F \frac{\partial E(u, w)}{\partial w} dw$$

The integral is along the line from point  $A$ , which corresponds to  $w=0$ , to the final weight state  $F$ . However, the training phase starts at point  $I$  (rather than  $A$ ), so we have to compromise on an approximation to integral above, namely we will use

$$E(w = w^f) - E(w = 0) = \int_A^I \frac{\partial E(u, w)}{\partial w} dw$$

This expression will be further approximated by replacing the integral by summation, taken over the discrete steps that the network passes while learning.

Thus the estimated sensitivity to the removal of connection  $w_{ij}$  will be evaluated as

$$\hat{S}_{ij} = - \sum_0^{N-1} \frac{\partial E}{\partial w_{ij}}(n) \Delta w_{ij}(n) \frac{w_{ij}^f}{w_{ij}^f - w_{ij}^i}$$

where  $N$  is the number of training epochs.

The above estimate for sensitivity uses terms that are readily available during the normal course of training. (Indeed, this was the main motivation for deriving such an approximation). Obviously the weight increments  $w_{ij}$  are the essence of every learning process, so they are always available. Also, virtually every optimization search (e.g., steepest descent, conjugate gradient, Newton's method) uses gradient to find the direction of change, so the partial derivative, which are the components of the gradient, are available. Therefore, the only extra computational demand for implementing our procedure is the summation. This (negligible) overhead merely calls for maintaining a 'shadow array' (of the same size as the number of connections in the network) that keeps track of the accumulated terms that build up  $S_{ij}$ .

For the case of backpropagation algorithm, which implemented in this work, the sensitivities are defined from the above equation as

$$\hat{S}_{ij} = \sum_0^{N-1} [\Delta w_{ij}(n)]^2 \frac{w_{ij}^f}{\eta (w_{ij}^f - w_{ij}^i)}$$

Upon completion of training we are equipped with a list of sensitivity numbers, one per each connection. They were created by a process that runs



concurrently, but without interfering, with the learning process. At this point a decision may be taken of the smallest sensitivity numbers in each layer. Usually the sensitivities of all the connections at each layer are arranged in the descending order in an array and the last values of the array are removed.

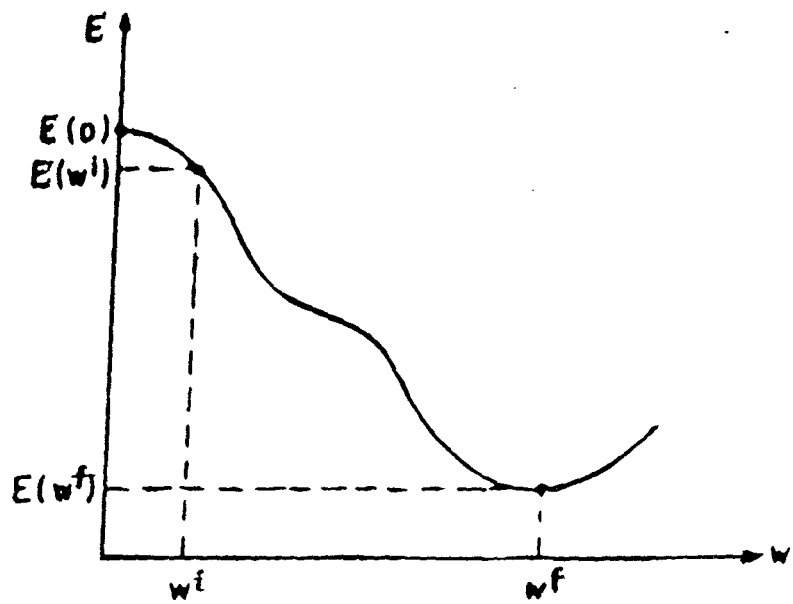


fig. 3.1 The error as a function of one weight

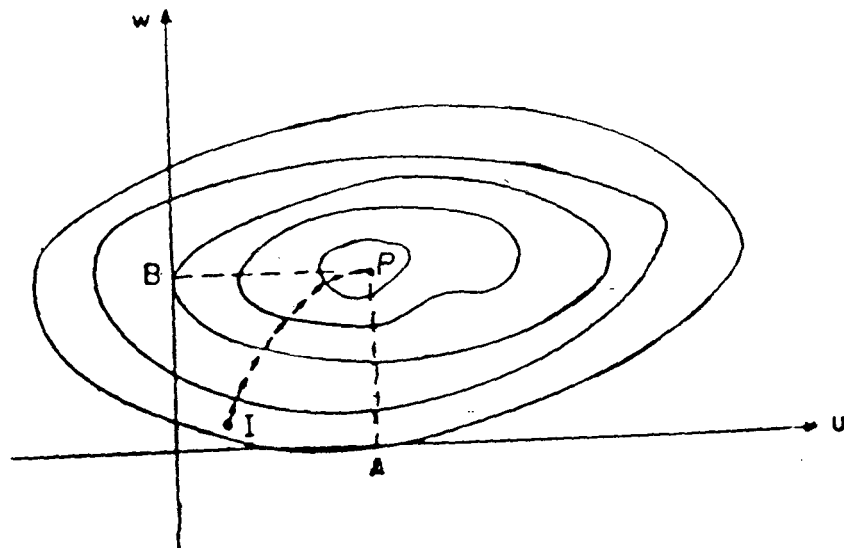


fig. 3.2 Learning as motion on an error function surface

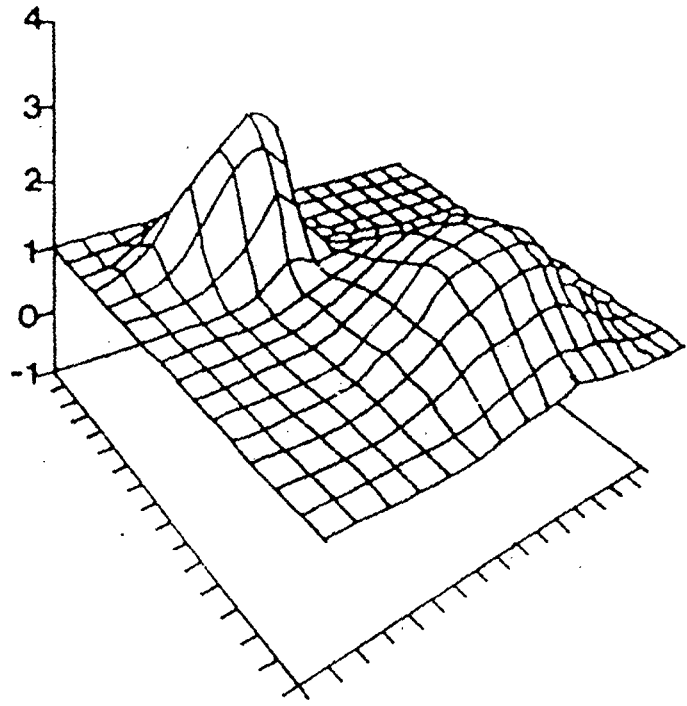


fig. 3.3 Error Surface

**CHAPTER 4**  
**IMPLEMENTATION AND RESULTS**

---

*"Neural nets made possible the transfer of methodology from other unrelated areas (physics, thermodynamics, systems control, signal processing etc.) in the AI arena"*

**Cris Kartrougeras**

The major design decisions for key neural network levels were:

Node Level		Network Level		Training Level	
1.	Type of input accepted	1.	Number of layers or slabs	1.	Learning algorithm
2.	Transfer function	2.	Number and type of nodes	2.	Learning parameters
3.	Means of combination	3.	Size of hidden layers	3.	Halting conditions
		4.	Number and type of output nodes		
		5.	Connectivity		

#### 4.1 Handwritten Character Recognition

Character recognition technique associate symbolic identity with the image of a character. This problem of replication of human functions by machines involves the recognition of both machine printed and hand printed cursive-written characters. Character recognition is better know as optical character recognition (OCR) since it deals with recognition of optimally processed characters rather than magnetically processed ones. The first successful attempt was made by Russian scientist Tyurin in 1900 as an aid to the visually handicapped [8]. In the middle of the 1940s, the modern versions of the OCR for business applications were found with the development of digital computers. The state of the art reports on character recognition research have been presented by Nagy, Harmon, Stallings, Stuen et al., Mori et al., Mantas, Davis and Chatterji. The principle aim of the OCR system is the need to cope with the enormous flood

of paper such as bank cheques, commercial forms, government records, credit card imprints and mail sorting generated by the expanding technology society.

## 4.2 Pattern Classification Problem

The pattern classification problem in the 2D feature space implemented is shown in the fig 4.1. The feature vector  $(X,Y)$  is uniformly distributed in  $[-1,1] \times [-1,1]$ , and the two non-classes are equally probable (i.e.,  $b = 2/(a+1) - 1$ ). The implementation aim is to distinguish whether a given pattern (co-ordinate) is in Class 1 or in Class 2. The output should be zero for patterns in Class 1 and one for Class 2.

## 4.3 Implementation of Handwritten Character Recognition

The work involved in this dissertation is the development of a C program on DEC VAX 11/780 computer to simulate a three layer (one hidden layer) neural network and to implement the backpropagation training algorithm to train the network to recognize handwritten letters.

### Training by Noisy Data

The different letters used in teaching the network were A, B, G, R, S and X. The input data for training the network were collected from 25 individuals. Each of the persons were given graph papers containing squares of 8 X 8 boxes. Then they were asked to write these six letters, in uppercase, large enough to fill the boxes. After the letters were entered, the squares of the boxes which get

intersected by any part of the letters were shaded. Since the pattern of letters written by different persons will have large variations, the collection of the patterns will certainly contain noisy input patterns. As it is described in the results, the training of the network by noisy data will considerably improve the performance of the network.

Out of 21 sets (126 patterns) collected, 20 sets (120 patterns) were used for testing and 4 patterns were used for testing the performance of backpropagation algorithm implemented. The data were entered into a file as a sequence of ones and zeros, representing the filled (shaded) and unfilled squares of the boxes respectively. The three bit identification codes used were:

A : 0 0 1	B : 0 1 0	G : 0 1 1
R : 1 0 0	S : 1 0 1	S : 1 1 0

#### The Network before Pruning

Number of layers = 3	Momentum factor = 0.15
Number of input nodes = 64 (8X8)	Learning rate = 0.075
Number of hidden nodes = 5	
Number of output nodes = 3	
Number of connections = 343 (including biases)	

#### The Network after Pruning

Number of layers = 3	Momentum factor = 0.15
Number of input nodes = 64	Learning rate = 0.075
Number of hidden nodes = 5	
Number of output nodes = 3	

Number of connections left = 280  
Number of connections pruned = 63  
(including biases)

#### 4.4 Implementation of Pattern Classification Problem

The network specified below for the backpropagation is trained by 24 different patterns both class 1 and class 2. The configuration of the network designed is shown in the fig 4.2.

The network before Pruning

Number of layers = 3	Momentum factor = 0.05
Number of input nodes = 2	Training rate = 0.5
Number of hidden nodes = 2	
Number of output nodes = 1	
Number of connections = 9 (including biases)	

The network after Pruning

Number of layers = 3	Momentum factor = 0.05
Number of input nodes = 2	Training rate = 0.5
Number of hidden nodes = 1	
Number of output nodes = 1	
Number of connections left = 6	
Number of connections pruned = 3 (including biases)	



## 4.5 Results

The results of the implementations of the two problems, handwritten character recognition and pattern classification, were given in the four tables. The results include the results of the performance of the pruned networks in both the cases. The failures given in the table are calculated as the deviation of the output for more than 25% (we say failure if the output deviates by more than 25%) and the disasters as when the output deviates by more than 50% (we say disaster if the output deviates by more than 50%).

As is it is evident from the results, the average sum- squared error increases almost linearly with the number of training epochs. The graph 1.1, graph 1.2 and graph 2.1, graph 2.2 reveals this fact. It is also clear that the performance of the networks increased with the number of iterations (see graphs ~~3 and 4~~). The pruning procedure adopted decreased the success rate of the network to a little extent , but, by obtaining the most wanted reduction of the size and the reduction of training time. It was also found that the training of the network with noisy data improves the recognition capability of the network.

**Table 1**  
**Results of Handwritten Character Recognition**  
**(without Pruning)**

Original Network (64-5-3):

Number of connections	= 343	Alphabets trained={A,B,G,R,S,X}	$\eta = 0.15$
Number of i/p nodes	= 64	Alphabets tested	$\alpha = 0.075$
Number of hidden nodes	= 5	A - 001 R - 100	No. of training
Number of o/p nodes	= 3	S - 101 X - 110	patterns = 120

Number of Iterations	Ave sum-squared Error	Output obtained	Desired Output	CPU Time (Min:Sec)	% of Failures	% of Disasters
50	0.105704	0.0381 0.8401 0.9421	0 1 1	01:56	75%	12.5%
		0.4774 0.0867 0.9021	1 0 0			
		0.7028 0.0835 0.9369	1 0 1			
		0.8541 0.6010 0.0614	1 1 0			
100	0.021061	0.0125 0.9566 0.9319	0 1 1	03:46	12.5%	6.25%
		0.7927 0.2211 0.5886	1 0 0			
		0.9532 0.0477 0.9628	1 0 1			
		0.9718 0.8434 0.0205	1 1 0			
300	0.006825	0.0020 0.9748 0.9795	0 1 1	11:37	6.25%	0%
		0.9577 0.4895 0.2283	1 0 0			
		0.9768 0.0217 0.9913	1 0 1			
		0.9922 0.8909 0.0115	1 1 0			
400	0.000011	0.0020 0.9748 0.9796	0 1 1	17:29	6.25%	0%
		0.9577 0.4898 0.2285	1 0 0			
		0.9768 0.0217 0.9913	1 0 1			
		0.9923 0.8909 0.0114	1 1 0			

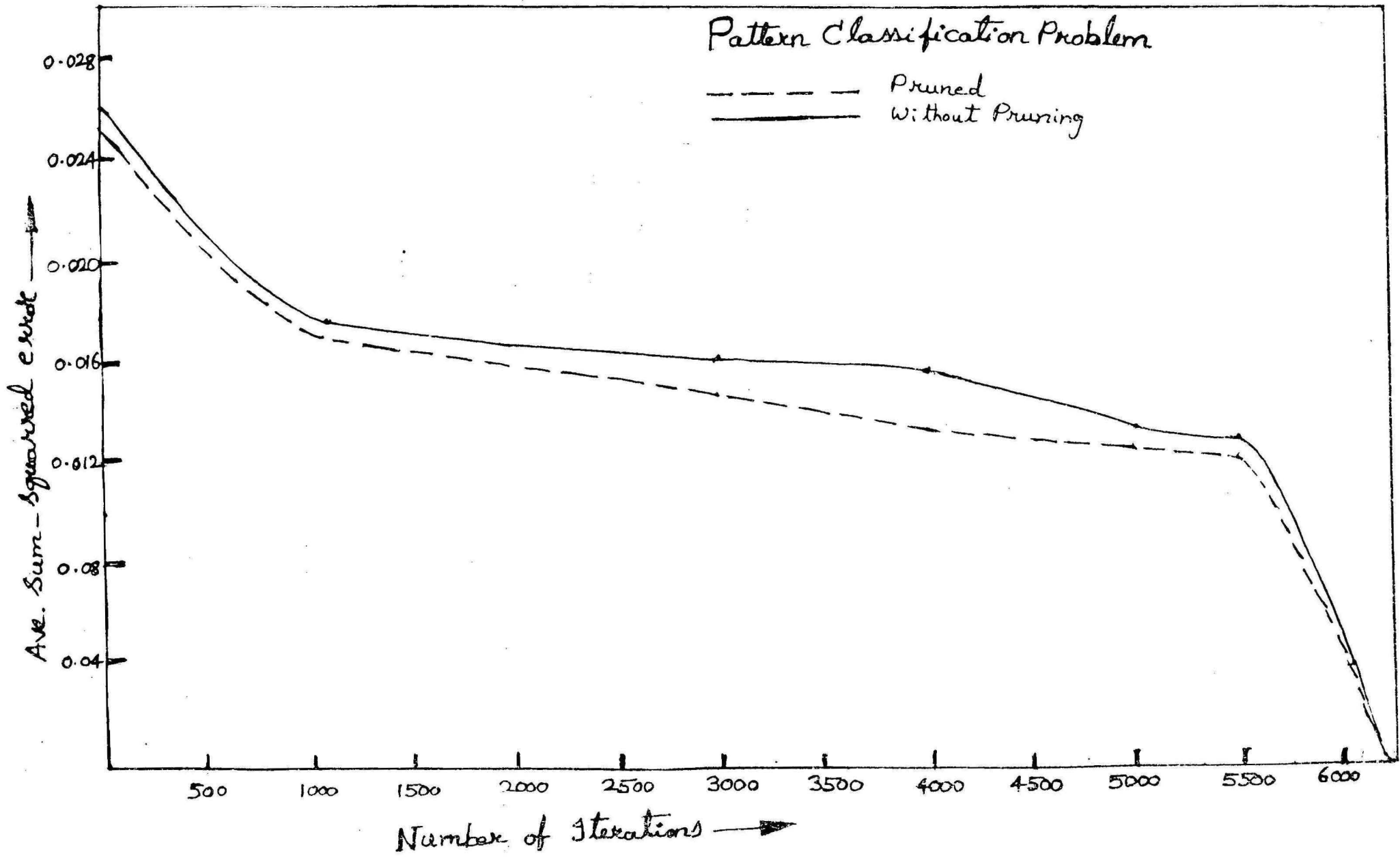
**Table 2**  
**Results of Handwritten Character Recognition**  
**(with Pruning)**

Pruned Network (64-5-3):

Number of connections	= 280	Alphabets trained = {A,B,G,R,S,X}	$\eta = 0.15$
Number of i/p nodes	= 64	Alphabets tested	$\alpha = 0.075$
Number of hidden nodes	= 5	A - 001 R - 100	No. of training
Number of o/p nodes	= 3	S - 101 X - 110	patterns = 120

Number of Iterations	Ave sum-squared Error	Output obtained	Desired Output	CPU Time (Min:Sec)	% of Failures	% of Disasters
50	0.066052	0.2052 0.8464 0.8780	0 1 1	01:56	75%	6.25%
		0.7473 0.1471 0.6627	1 0 0			
		0.4001 0.1063 0.8855	1 0 1			
		0.9427 0.6138 0.0118	1 1 0			
100	0.012451	0.1415 0.9364 0.9391	0 1 1	03:40	18.75%	6.25%
		0.6504 0.3001 0.4795	1 0 0			
		0.7603 0.0938 0.9189	1 0 1			
		0.9347 0.8665 0.0025	1 1 0			
300	0.002326	0.0615 0.9748 0.9669	0 1 1	11:12	12.5%	6.25%
		0.7011 0.2449 0.5600	1 0 0			
		0.9084 0.0385 0.9669	1 0 1			
		0.9964 0.5636 0.0002	1 1 0			
400	0.001628	0.0515 0.9791 0.9717	0 1 1	14:33	12.5%	< 6.25%
		0.7241 0.2215 0.6088	1 0 0			
		0.9244 0.0323 0.9726	1 0 1			
		0.9979 0.5186 0.0001	1 1 0			

F.7 page 5



Graph 1.2

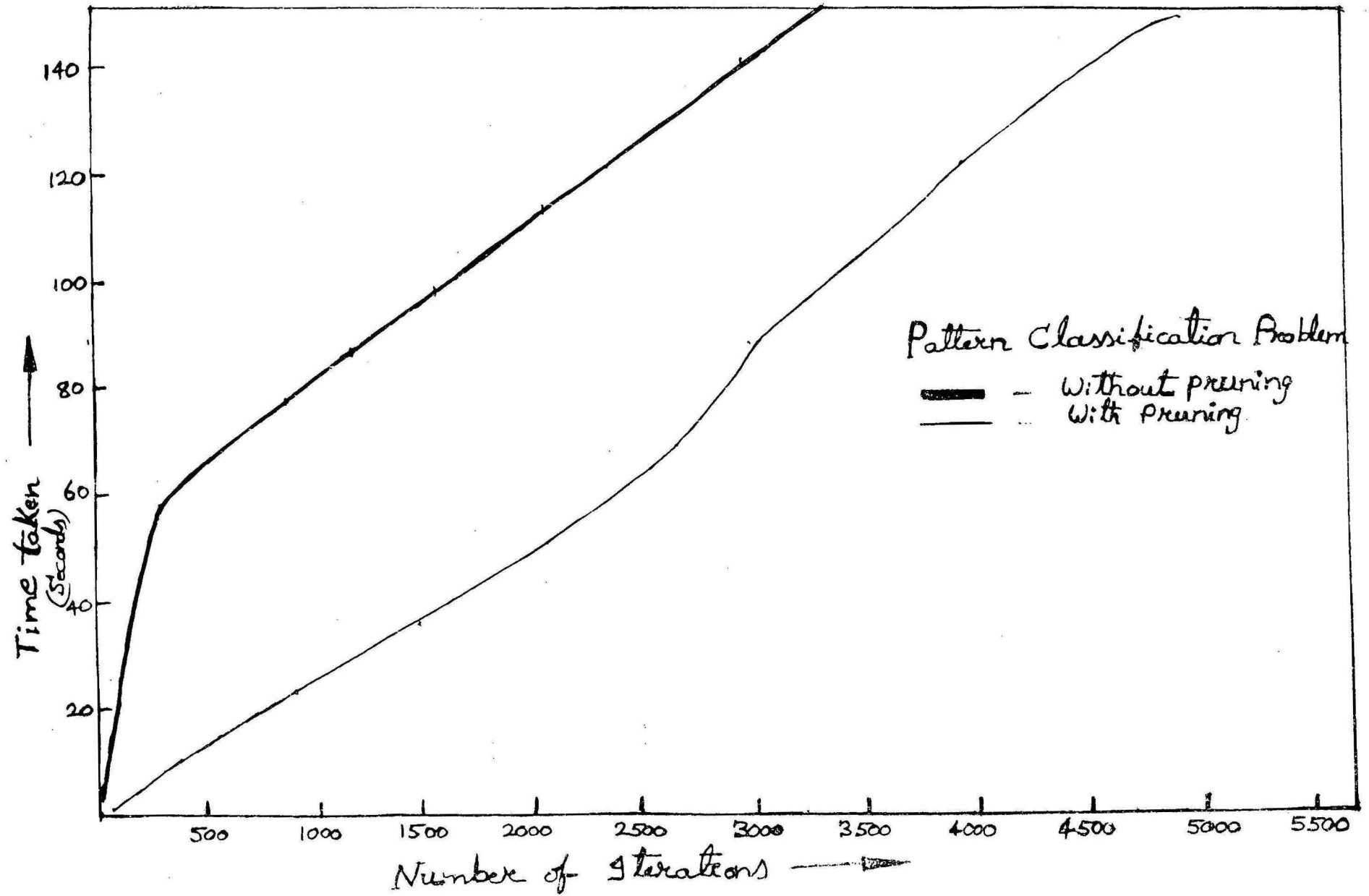


Table 3  
Results of Pattern Classification Problem  
(without pruning)

Original Network (2-2-1):  
 Number of connections = 9       $\eta = 0.05$   
 Number of i/p nodes = 2       $\alpha = 0.5$   
 Number of hidden layer nodes = 2      Number of training patterns = 24  
 Number of o/p nodes = 1

Number of Iterations	Ave sum-squared Error	Output obtained	Desired Output	CPU Time (Min:Sec)	% of Failures	% of Disasters
10	0.247501	0.5137	0	00:01	100%	50%
		0.5365	1			
		0.5344	1			
		0.5340	0			
100	0.243222	0.5316	0	00:2	150%	0%
		0.5557	1			
		0.5414	1			
		0.5264	0			
1200	0.172491	0.3613	0	01:26	25%	0%
		0.9671	1			
		0.7381	1			
		0.2893	0			
3000	0.144416	0.3419	0	02:21	25%	0%
		0.9921	1			
		0.9668	1			
		0.3315	0			
4000	0.135619	0.3510	0	02:52	25%	0%
		0.9944	1			
		0.9405	1			
		0.3386	0			

contd..

5000	0.131583	0.3419	0	03:23	25%	0%
		0.9923	1			
		0.9667	1			
		0.3315	0			
6000	0.123995	0.3128	0	03:05	< 25%	0%
		0.9894	1			
		0.9818	1			
		0.3069	0			
10000	0.016687	0.0003	0	05:10	0%	0%
		0.9995	1			
		0.9987	1			
		0.2271	0			

---

Table 4

**Results of Pattern Classification Problem  
(with Pruning)**

Pruned to (Network 2-1-1):  
 Number of connections pruned = 3       $\eta = 0.05$   
 Number of i/p nodes = 2       $\alpha = 0.5$   
 Number of hidden layer nodes = 1      Number of training patterns = 24  
 Number of o/p nodes = 1

Number of Iterations	Ave sum-squared Error	Output obtained	Desired Output	CPU Time (Min:Sec)	% of Failures	% of Disasters
10	0.247601	0.5466	0	00:01	100%	50%
		0.5462	1			
		0.5442	1			
		0.5432	0			
100	0.241955	0.5299	0	00:02	100%	50%
		0.5581	1			
		0.5415	1			
		0.5222	0			
1200	0.172559	0.3610	0	00:31	75%	0%
		0.9660	1			
		0.7378	1			
		0.2894	0			
3000	0.160310	0.3143	0	01:28	25%	0%
		0.9959	1			
		0.7975	1			
		0.2976	0			
4000	0.156061	0.2943	0	02:02	< 25%	0%
		0.9979	1			
		0.8266	1			
		0.2943	0			

contd..

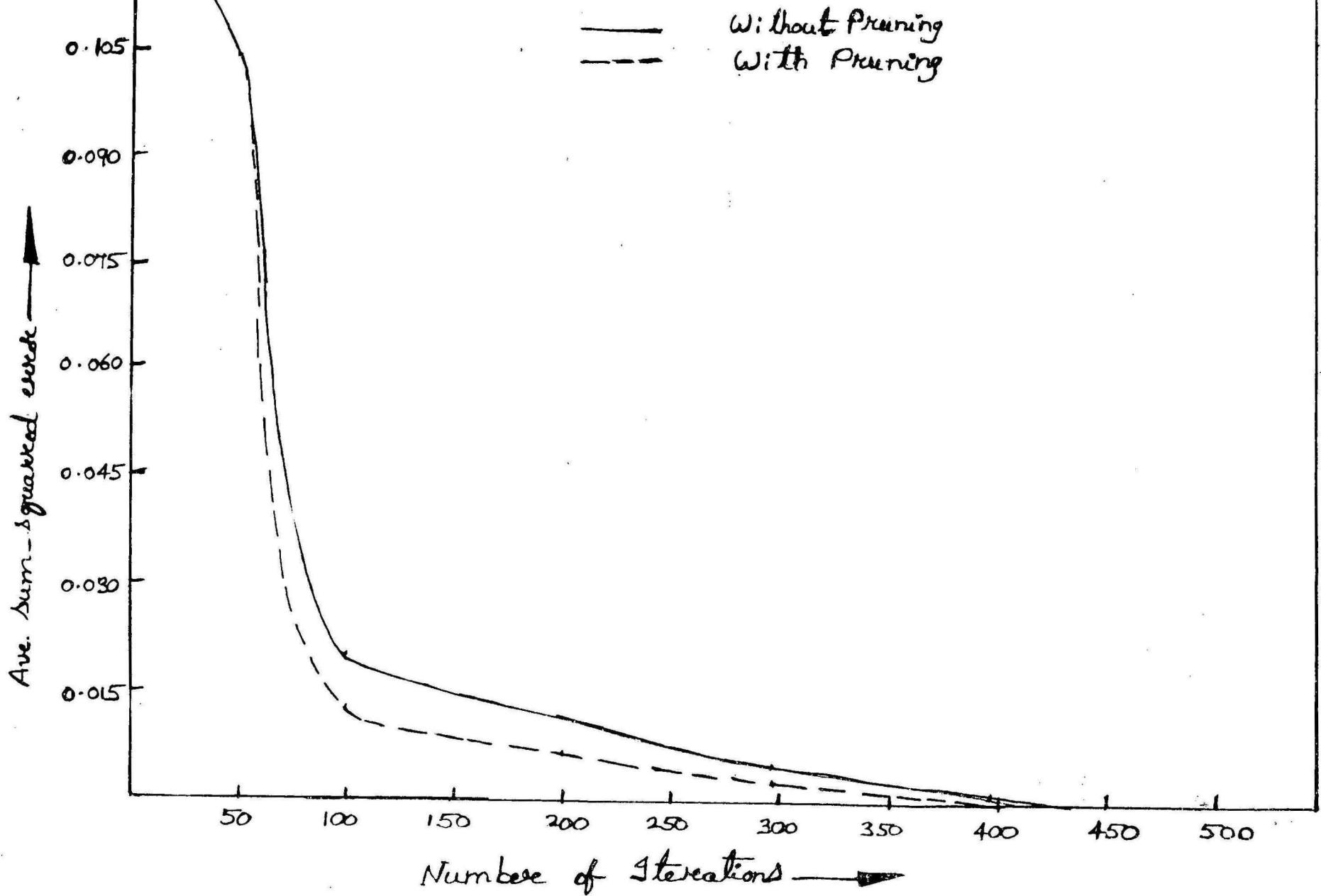


5000	0.130326	0.0529	0	02:29	25%	0%
		0.9989	1			
		0.8632	1			
		0.3560	0			
6000	0.124805	0.0136	0	03:04	25%	0%
		0.9989	1			
		0.8782	1			
		0.3665	0			
10000	0.0017451	0.0015	0	05:09	< 25%	0%
		0.9993	1			
		0.9218	1			
		0.3548	0			

---

# Handwritten Char. Recognition

Graph 2.1

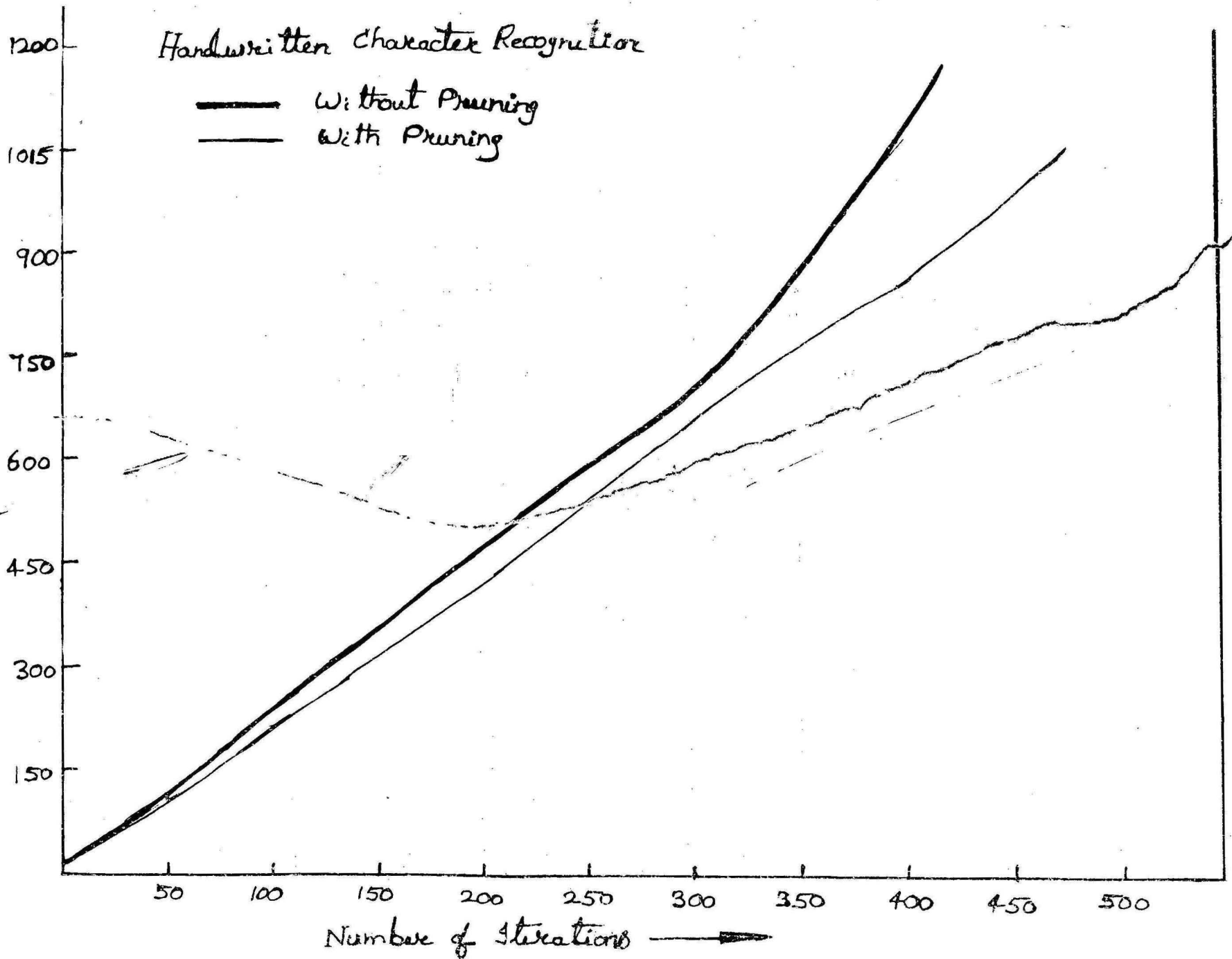


# Handwritten Character Recognition

— Without Pruning  
— With Pruning

Graph 2.2

Time taken (Seconds)



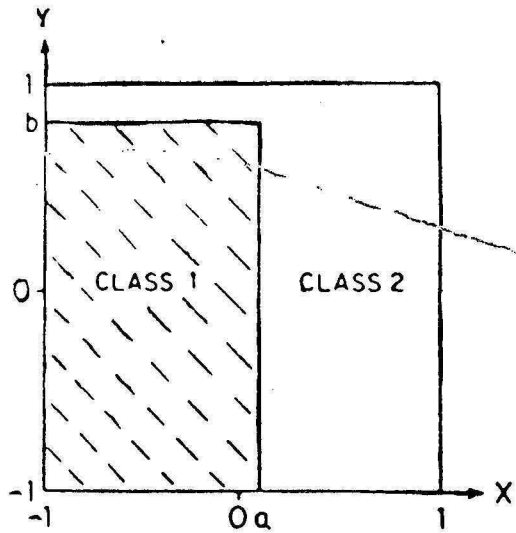


fig.4.1 Pattern  
classification Problem

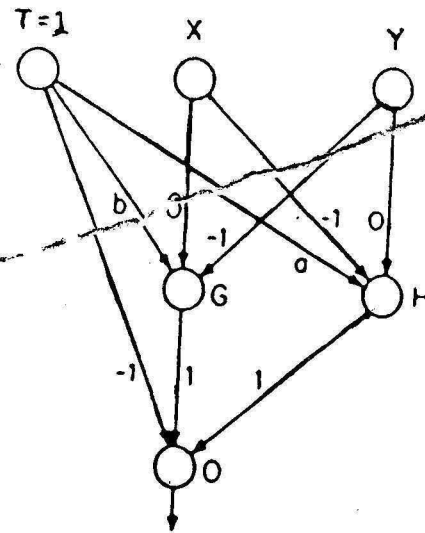


fig.4.2 The Network

## CONCLUSION

A generic Backpropagation training algorithm is implemented for neural network applications and is tested with handwritten character recognition and pattern classification problem. The pruning procedure for back propagation is implemented in C on DEC VAX 11/780 and the experimental results of pattern classification problem and handwritten character recognition were studied before and after pruning. Unlike the other pruning procedures, the implemented pruning procedure does not interfere with the training.

It is found that the learning time and forward calculation times are reduced with the pruning without much affecting the performance of the network. The noisy training improved the performance of the network to a considerable extent. Though the average sum-squared error is increased a bit with pruning, the reduced size of the network makes hardware implementation of the neural networks easy.

The extension of the character recognition work would be to cover both lower, upper case letters, digits and to a number of control characters, before tackling well-formed cursive writing. One can imagine the possibility of customising the network to a particular user through recognition of his own writing by incremented learning.

## REFERENCES

1. Ehud D. Karnin, "A Simple Procedure for Pruning Back- Propagation Trained Neural Networks", IEEE Trans. Neural Networks. Vol 1. No. 2, (June 1990), pp. 239-242.
2. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation", in Parallel Distributed Processing, Vol. 1, D. E. Rumelhart and I. C. McClelland Eds. Cambridge, M.A: MIT Press, (1986), pp. 318-362.
3. J. Sietsma and R. J. F. Dow, "Neural Net Pruning - Why and How?", in Proc. IEEE Int. Conf. Neural Networks, Vol. 1 (San Diego, CA), (1988), pp. 325-332.
4. J. Siestma and R. J. F. Dow, "Creating Artificial Neural Networks that Generalize", Neural Networks, Vol 4, (1991), pp. 67-69.
5. P. D. Wasserman, "Neural Computing - Theory and practice", Van Nostrand Reinhold, NY, (1989).
6. K. G. Schweller and A. L. Plagman, "Neural nets and Alphabets: Introducing Students to Neural Networks", SIGCSE Bulletin, Vol.21, No. 3 (Sept 1989), pp. 2-7.
7. R. P. Lippman, "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, (April 1987), pp. 4-21.
8. Charles C. Tappert, Ching Y. Suen and Tasu Wakashara, "The State of the art in On-line Handwriting Recognition", IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 12., No. 8, (Aug 1990), pp. 787-808.

9. Micheal Conrad, "Molecular Computer Design: A Synthetic Approach to Brain Theory", Real Brains-Artificial Mind, J. L. Cost and A. Karlquist (Eds), North-Holland.
10. I. C. Whitefield, "Neurocommunications : An Introduction", Wiley-Interscience publication, Aven, Great Britain, 1984.
11. Russel C, Eberhart and Roy W. Dobbins, "Artificial Neural Network Tools", Columbia, M D., 1988.
12. David J. Montana and L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms", in Eleventh Int. Joint Conf. on AI, (Aug 1989), Detroit, Michigan, vol. 1, USA , pp. 762-767.
13. S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with backpropagation", Advances in Neural Information Procesing I. D. S. Toretzky Ed. Morgan Kaufmann, 1989, pp. 177-185.
14. David Bailey and Donna Thompson, "How to Develop Neural Network Applications", AI Expert, (June 1990), pp. 38-47.