

# **PATTERN RECOGNITION USING NEURAL NETWORKS**

*Project Report submitted in partial fulfillment of the requirement for the  
degree of B.Tech*

## **BACHELOR OF TECHNOLOGY IN ELECTRONICS AND COMMUNICATION ENGINEERING**

By  
**DHRUV BATRA           131018**  
**GUNDEEP SINGH       131044**  
**ACHMN SHUKLA       131104**

UNDER THE GUIDANCE OF

**Dr. Neeru Sharma**



JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT  
May, 2017

# TABLE OF CONTENTS

|  | <b>Page<br/>Number</b> |
|--|------------------------|
| <b>DECLARATION BY SUPERVISOR</b>                       | <b>iii</b>             |
| <b>SUPERVISOR'S CERTIFICATE</b>                        | <b>iv</b>              |
| <b>ACKNOWLEDGEMENT</b>                                 | <b>v</b>               |
| <b>LIST OF FIGURES</b>                                 | <b>vi</b>              |
| <b>LIST OF TABLES</b>                                  | <b>vii</b>             |
| <b>ABBREVIATIONS USED</b>                              | <b>viii</b>            |
| <b>ABSTRACT</b>  | <b>ix</b>              |
| <br>   |                        |
| <b>CHAPTER-1</b>                                       |                        |
| <b>INTRODUCTION</b>                                    | <b>1</b>               |
| <b>1.1 What Is Machine Learning?</b>                   | <b>1</b>               |
| <b>1.2 Classifications</b>                             | <b>3</b>               |
| <b>1.2.1 Supervised Learning</b>                       | <b>3</b>               |
| <b>1.2.2 Learning Multiple Class</b>                   | <b>4</b>               |
| <b>1.2.3 Unsupervised Learning</b>                     | <b>6</b>               |
| <b>1.2.4 Reinforced Learning</b>                       | <b>6</b>               |
| <b>1.3 Model Selection Procedure</b>                   | <b>7</b>               |
| <b>1.3.1 Cross Validation</b>                          | <b>7</b>               |
| <b>1.3.2 Structural Risk Minimization</b>              | <b>9</b>               |
| <b>1.3.3 Minimum Description Length</b>                | <b>9</b>               |
| <br>   |                        |
| <b>1.4 Organization of the report</b>                  | <b>10</b>              |
| <b>CHAPTER-2</b>                                       |                        |
| <b>LITERATURE REVIEW</b>                               | <b>11</b>              |
| <b>2.1 Multilayer Perceptron</b>                       | <b>11</b>              |
| <b>2.1.1 Neurons</b>                                   | <b>11</b>              |
| <b>2.1.2 Understanding The Brain</b>                   | <b>12</b>              |
| <b>2.1.3 Neural Networks &amp; Parallel Processing</b> | <b>13</b>              |

|   |           |
|---|-----------|
| <b>2.2 The Perceptron</b>                         | <b>14</b> |
| <b>2.3 Training Perceptron</b>                    | <b>15</b> |
| <b>2.4 Learning Boolean Function</b>              | <b>16</b> |
| <br>  |           |
| <b>CHAPTER-3</b>                                  |           |
| <b>IMPLEMENTATION</b>                             | <b>19</b> |
| <b>3.1 Back Propagation</b>                       | <b>19</b> |
| <b>3.2 Training Procedure</b>                     | <b>20</b> |
| <b>3.2.1 Improving Convergence</b>                | <b>20</b> |
| <b>3.2.2 Overtraining</b>                         | <b>21</b> |
| <b>3.2.3 Structuring The Network</b>              | <b>22</b> |
| <br>  |           |
| <b>3.3 Tools For Design</b>                       | <b>23</b> |
| <b>3.3.1 Raspberry pi</b>                         | <b>23</b> |
| <b>3.3.2 Processor</b>                            | <b>24</b> |
| <b>3.4 Steps in Optical Character Recognition</b> | <b>25</b> |
| <br>  |           |
| <b>3.5 Input</b>                                  | <b>27</b> |
| <br>  |           |
| <b>3.6 Training</b>                               | <b>27</b> |
| <br>  |           |
| <b>3.7 Testing and Output</b>                     | <b>28</b> |
| <br>  |           |
| <b>CHAPTER-4</b>                                  |           |
| <b>RESULTS AND CONCLUSION</b>                     | <b>29</b> |
| <b>REFERENCES</b>                                 | <b>30</b> |

## DECLARATION

We hereby declare that the work reported in the B-Tech thesis entitled “**Pattern Recognition Using Neural Networks**” submitted at **Jaypee University of Information Technology, Waknaghat India**, is an authentic record of our work carried out under the supervision of **Dr. Neeru Sharma**. We have not submitted this work elsewhere for any other degree or diploma.

DHRUV BATRA (131018)  
GUNDEEP SINGH KALRA (131044)  
ACHMN SHUKLA (131104)

Department of Electronics and communication

Jaypee University of Information Technology, Waknaghat, India

May 2, 2017

## **SUPERVISOR'S CERTIFICATE**

This is to certify that the work reported in the B-Tech. thesis entitled “**Character Recognition Using Neural Network**”, submitted by **Dhruv Batra (131018), Gundeep Singh (131044), Achmn Shukla (131104)** at **Jaypee University of Information Technology, Wagnaghat, India**, is a bonafide record of their original work carried out under my supervision. This work has not been submitted elsewhere for any other degree or diploma

(Signature of Supervisor)

Dr. Neeru Sharma

2 May, 2015

## **ACKNOWLEDGEMENT**

We have taken efforts in this project. However, it would not have been possible without the kind support and help of our guide and many other individuals. I would like to extend my sincere thanks to all of them.

We are highly indebted to Dr. Neeru Sharma (Assistant Professor, JUIT) for her guidance and constant supervision as well as for providing necessary information regarding the project and also for her immense support in completion of the project.

We would like to express our gratitude towards Mr Kamlesh for regularly helping us and making everything available during our difficult times.

We would like to express our special gratitude and thanks to Department of Electronics and Communication, Jaypee University of Information and Technology for providing us with all the resources and amazing faculty members who have always been our guiding light.

Our thanks and appreciations also go to my colleague for giving in their important reviews about the project and people who have willingly helped me out with their abilities.

## LIST OF FIGURES

| <b>Figure Number</b> | <b>Caption</b>                         | <b>Page Number</b> |
|----------------------|--|--------------------|
| 1.1                  | Training set for “family car”          | 4                  |
| 1.2                  | Example of Hypothesis class            | 4                  |
| 1.3                  | Instance example from 3 classes        | 5                  |
| 1.4                  | Function Description                   | 8                  |
| 1.5                  | Error vs. Polynomial error             | 8                  |
| 1.6                  | Error vs. Polynomial error             | 9                  |
| 2.1                  | Typical Neural Network                 | 12                 |
| 2.2                  | A simple perceptron                    | 14                 |
| 2.3                  | K parallel perceptrons                 | 15                 |
| 2.4                  | Perceptron training algorithm          | 16                 |
| 2.5                  | Perceptron implementing AND gate       | 17                 |
| 2.6                  | XOR gate geometric interpretation      | 18                 |
| 3.1                  | Mean square error vs. Hidden units     | 21                 |
| 3.2                  | Mean square error vs. training epochs  | 22                 |
| 3.3                  | Weight sharing under network structure | 23                 |
| 3.4                  | Raspberry Pi III                       | 24                 |
| 3.5                  | Raspberry Pi pin out diagram           | 25                 |
| 3.6                  | Steps in OCR                           | 26                 |
| 3.7                  | Input data                             | 27                 |
| 3.8                  | Example training set                   | 28                 |
| 4.1                  | Output                                 | 29                 |

## LIST OF TABLES

|                                      |    |
|--------------------------------------|----|
| Table 2.1: AND gate input and output | 17 |
| Table 2.2: XOR gate input and output | 17 |



## **ABBREVIATIONS USED**

|      |                                      |
|------|--------------------------------------|
| ANN  | Artificial Neural Network            |
| CPU  | Central Processing Unit              |
| GPU  | Graphics Processing Unit             |
| HDMI | High-Definition Multimedia Interface |
| HOG  | Histogram of Oriented Gaussians      |
| MDL  | Minimum Description Length           |
| MIMD | Multiple instructions multiple data  |
| MLP  | Minimum Description Length           |
| OCR  | Optical Character Recognition        |
| SRM  | Structural Risk Minimization         |
| SVM  | Support Vector Machine               |

## **ABSTRACT**

Character recognition is just one part of the pattern recognition existing in the world. The major advantage of doing the pattern recognition using artificial intelligence and unsupervised learning is that with the use of correct data set you can teach it to recognize every pattern that exists. Our mind can do pattern recognition or handwritten character deciphering very efficiently and easily, simply because our mind is made up of a large set of neural networks.

Doing it in real time is another challenge but we have considered challenges as stages of success rather than considering them as hurdles in achieving our goal. Thus using non biological neural networks we have implemented character recognition using raspberry PI. The problem still exists because of the competition between the efficiency and speed.

A camera is used for image acquisition in the real time and after filtering the noisy data and preprocessing it is passed on. Feature extraction is another important part of the procedure in which differentiation between the characters is done. It helps recognizing the characters. The characters in the photo are stored as an image of matrix of pixels. Back propagation algorithm is used to optimize the results achieved so that our output is closer to the results expected. Weights and biases of the neural network are automatically adjusted according to the back propagation which further helps in improving the results.

# CHAPTER 1

## INTRODUCTION

### 1.1 What Is Machine Learning?

We are in the period of "Big data". Once just organizations used to have data and the data was put away and handled on computer centers. With the landing of PCs and the widespread utilization of wireless communications, we all moved towards becoming producers of data. Each time we rent a movie, purchase an item, visit a site page, post on the online networking sites, and compose a blog or, even when we simply walk or drive around, we are creating data. Each one of us is a generator as well as a customer of data. We want to have services and products specialized for us. We want our needs to be understood and interests to be anticipated. For example, a supermarket chain stores the details of each and every transaction: date, client id, merchandise purchased and their sum, total money spent, etc. This leads to production of a great amount of data every day. To maximize sales and profit, the supermarket chain must be able to anticipate which client is probably going to purchase which item. Likewise client wants to find the products, best matching his/her needs.

This task is not apparent. We don't know precisely which individuals are probably going to purchase this dessert or the following book of this writer, see this new movie, or visit this city. Client behavior changes in time and by geographic area. Yet, what we know is that it is not totally arbitrary. Individuals don't go to supermarkets and purchase things arbitrarily. There are certain patterns in the data which are to be found and used for predicting the future transactions of the clients. To tackle a problem on a computer, we require an algorithm. A set of instructions that should to be carried out on the input to produce a desired output is called an algorithm. For instance, an algorithm for finding the smallest number in a set of numbers. The input is a set of numbers and the output is the smallest number. For similar tasks there might be different algorithms but we are keen on finding the most effective one.

For a few tasks, we don't have an algorithm. Predicting client behavior is one. Another case is separating spam messages from legal ones. The input is an email document, which is a file of characters. The output is a yes/no demonstrating whether the message is spam or not. However, we don't have a means by which to change the input to the output. What is considered spam changes with time and from person to person. But what we lack in knowledge we compensate for in data. We can easily incorporate a large number of case messages, some of which we know to be spam, and we "learn" what constitutes spam

from them. In other words, we need the computer to extract automatically the algorithm for this task. There is no reason to learn how to find the smallest number since we have many algorithms devised for that. Yet there are numerous applications for which we don't have an algorithm but have loads of data. Identifying the process completely is not possible, but a decent and helpful estimation can be built. In spite of the fact that identifying the complete process may not be possible, identifying certain patterns or regularities may be helpful. This is the niche of machine learning. Such patterns may help us understand the procedure, or to make predictions and these predictions can be relied upon assuming the near future to be almost same as the time of collecting data. Utilization of machine learning strategies to huge databases is called data mining. In data mining, a huge volume of data is processed to build a basic model with significant use, for instance, having high predictive precision. Its application territories are abundant: In finance, banks break down their past information to assemble models to use in credit applications, fraud discovery, and stock markets. In manufacturing, learning models are utilized for optimization, control, and troubleshooting. In medicine, learning projects are utilized for medical diagnostics. In telecommunications, call patterns are examined for system enhancement and boosting the quality of service (QoS). In science, a large amount of data in physics, astronomy, and biology can only be analyzed fast enough by computers. The World Wide Web is vast and continually growing. Therefore, manual searching of required information is not possible.

Machine learning is a database problem as well as a part of artificial intelligence. A system in a changing environment should have the ability to learn from the experience to be called intelligent. Prediction for every single conceivable circumstance is not necessary if the system itself learns and adjust to such changes. Machine learning is likewise useful in finding solutions to many problems in vision, speech recognition, and robotics. For instance, take recognition of faces. Daily we meet our friends or relatives, or see their photographs and can recognize them effortlessly. If the procedure is asked, we can't clarify thus not able to compose a computer program for this. Additionally, we realize that a face image is not just a random collection of pixels, the face has a structure, and it is symmetric. A face has eyes, nose, and mouth, situated in specific spots. Every individual's face is a pattern made out of a specific blend of these. A learning program, by analyzing sample face images of a person, captures the pattern particular to the individual and recognizes by checking for this pattern in a given image. This is one example of pattern recognition.

Another example is optical character recognition, which is recognition of character codes from their images. It is an interesting case when the characters are written by hand—for

instance, amounts on checks or postal address on envelopes. Different individuals have distinctive handwriting styles and there may be many possible images for the same character. In spite of the fact that writing is a human invention, a system which is as accurate as human reader is yet to be designed. We don't have a formal description of "A" that covers all 'A's. We take samples from writers and learn of the A- from these samples. What we want is to extract the commonness in all those distinct 'A's from the examples by extracting the most common feature that is there in each A. The machine can then be trained accordingly.

The theory of statistics is utilized as a part of building mathematical models in machine learning. There is a twofold part of computer science: First, in training, an efficient algorithm is utilized to tackle the issue of optimization, and furthermore to store and process the large amount of data we generally have. Second, once a model is learned, the representation and algorithmic solution for inference must also be efficient.

## **1.2 Classification**

Machine Learning is classified depending on the learning algorithms used. There are majorly three types of machine learning –Supervised learning, Unsupervised learning and Reinforced learning.

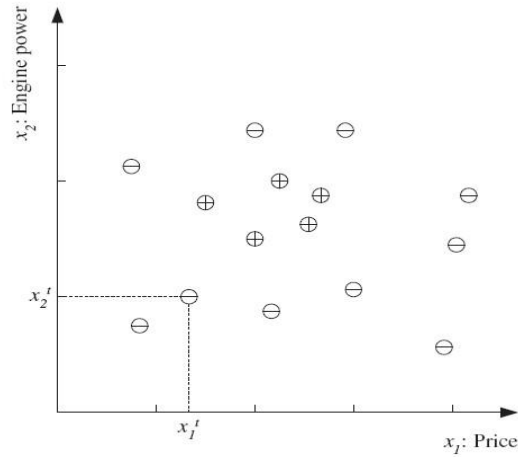
### **1.2.1 Supervised Learning**

The machine learning task of working out a function from labeled training data, which consist of a set of training examples is called supervised learning. Each example under supervised learning is a pair of an input object (typically a vector) and the desired output value (also called the supervisory signal). By analyzing the training data through a supervised learning algorithm we formulate a function which can be used for mapping new examples. For inconspicuous instances, an optimal situation will take into account the algorithm to accurately decide the class labels. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.[1]

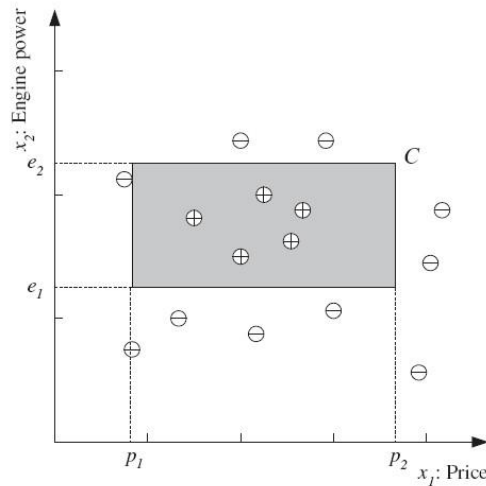
#### **EXAMPLE**

Suppose we want to learn about family cars and create a class, C, which describes them. A set of cars is taken and shown to a group people whom we gather to conduct the survey. Their responses are gathered and the results are then simulated. Positive examples are those to which the people points at as what they believe is a family car and the negative examples are all other cars. Then class learning is done in which we gather the description of positive examples and group those descriptions and call that class as “family cars”. Doing this, we can make a forecast: Given a car that we have not seen before, we will be able to say whether it is a

family car or not by checking with description learned. Or, then again we can do learning extraction: This review might be supported by a car company, and the point is to comprehend what the individuals' desire from a family car.



**Fig 1.1** Training set for the class of a “family car.” The coordinates of the point indicate the price and engine power of that car. ‘+’ denotes (a family car), and ‘-’ denotes (not a family car); it is another type of car. [3]



**Fig 1.2** Example of a hypothesis class. Space where each instance  $t$  is a data point at coordinates  $(x_1^t, x_2^t)$  and its type, namely, positive versus negative, is given by  $r^t$  (see figure 1.1). After analysis of the data, we may have reason to believe that for a car to be a family car, its price and engine power should be in a certain range [3]

### 1.2.2 Learning Multiple Classes

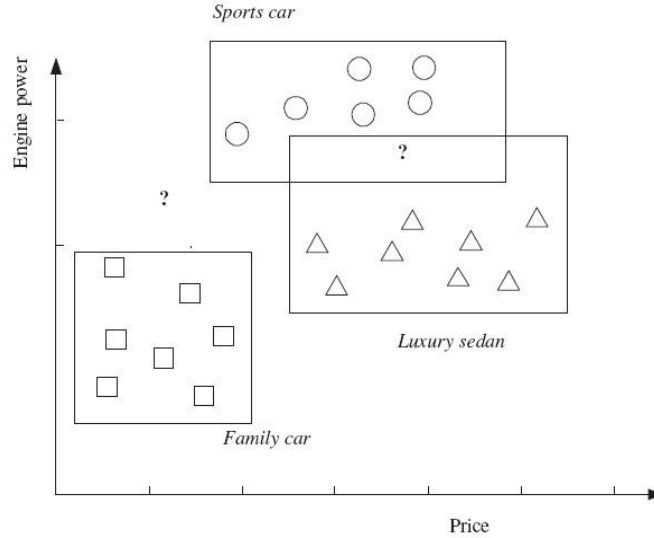
In the previous example of learning a class for family cars, positive examples belong with the class family cars and the negative examples to every other cars. This is a two-class problem. In the general case, we have  $K$  classes signified as  $C_i, i = 1, \dots, K$  and an input instance belongs with one and precisely one of them. The training set is now of the form

$$X = \{x^t, r^t\}_{t=1}^N \tag{1.1}$$

where  $r$  has  $K$  dimensions and

$$r_i^t = \begin{cases} 1, & x^t \in C_i \\ 0, & x^t \in C_j, j \neq i \end{cases} \quad (1.2)$$

An example is given in figure 1.3



**Fig 1.3** Instances from three classes: family car, sports car, and luxury sedan. [3]

The boundary isolating the instances of one class from the instances of every single different class is learnt in machine learning for the classification. In this manner, a  $K$ -class classification problem is taken as  $K$  two-class problems. The training examples belonging to  $C_i$  are the positive instances of hypothesis  $h_i$  and the examples of all other classes are the negative instances of  $h_i$ . Thus in a  $K$ -class problem, we have  $K$  hypotheses to learn such that

$$h_i(x^t) = \begin{cases} 1, & x^t \in C_i \\ 0, & x^t \in C_j, j \neq i \end{cases} \quad (1.3)$$

The total empirical error takes a sum over the predictions for all classes over all instances:

$$E(\{h_i\}_{i=1}^K | X) = \sum_{t=1}^N \sum_{i=1}^K 1(h_i(x^t) \neq r_i^t) \quad (1.4)$$

Ideally only one of  $h_i(x)$ ,  $i = 1, \dots, K$  is 1, for a given  $x$ , and we can choose a class. We cannot choose or reject a class when none, or two or more,  $h_i(x)$  is 1. The classifier rejects such doubtful cases. In the example of learning a class for family cars, only one hypothesis was used to model only the positive examples. Sometimes building two hypotheses is preferable, one for the positive instances and the other for the negative instances, assuming a structure for the negative instances also. Separation family cars from sports cars

is an example; each class has a structure of its own. The advantage is that if the input is a luxury sedan, we can reject the input as both hypotheses decide negative.

### **1.2.3 Unsupervised Learning**

The correct values for mapping of input to the output is provided by a supervisor when supervised learning is used. But in unsupervised learning there is just input information and no supervisor. Finding the regularities in the information is the primary point here. There are more frequent occurrences of certain patterns than others, and such patterns need to be identified. This method is called density estimation in statistics. Clustering is one of the strategies used for density estimation where discovering clusters or groupings of info is the aim. For an organization with an information of past clients, the client information includes the transactions by the clients with the organization in past. The organization may want to see the distribution of the profile of its customers to see describe frequently visiting clients. Along these lines clients with similar attributes are allocated the same group by the clustering model; this is called customer segmentation. On formation of such groups, organizations may decide strategies specific to these groups; this is known as customer relationship management. Machine learning techniques are likewise utilized as a part of bioinformatics.

### **1.2.4 Reinforced Learning**

In some applications, a sequence of actions determines the output of a system. A single right action is not important in such cases but a sequence of right actions, i.e., the policy. There is no such thing as the best action; the action is good if it is a part of a good policy. The goodness of policies must be accessed by the machine learning program and generate a policy by learning from past good action sequences. Such learning reinforcement techniques are called reinforcement learning algorithms. A decent example is game playing where a solitary move by itself is not that essential; it is the sequence of right moves that is good. A move is good if it is a part of a good game playing policy. Game playing is a research area imperative in both artificial intelligence and machine learning. This is because games are easy to describe and in the meantime, they are very hard to play well. A game like chess has few guidelines yet very difficult to play because of a huge number of moves in the game with an extensive number of conceivable moves at each state. On learning algorithms that knows how to play games well, they can be applied to applications with more obvious economic utility.

Another application range of reinforcement learning is a robot exploring an environment in search of an objective area. The robot can choose and move in any direction at a given instance



of time. But after various trial runs, the right sequence of actions to reach the objective state from an underlying state should be learned which helps the robot in doing the task as quickly as possible and without hitting any of the obstacles.

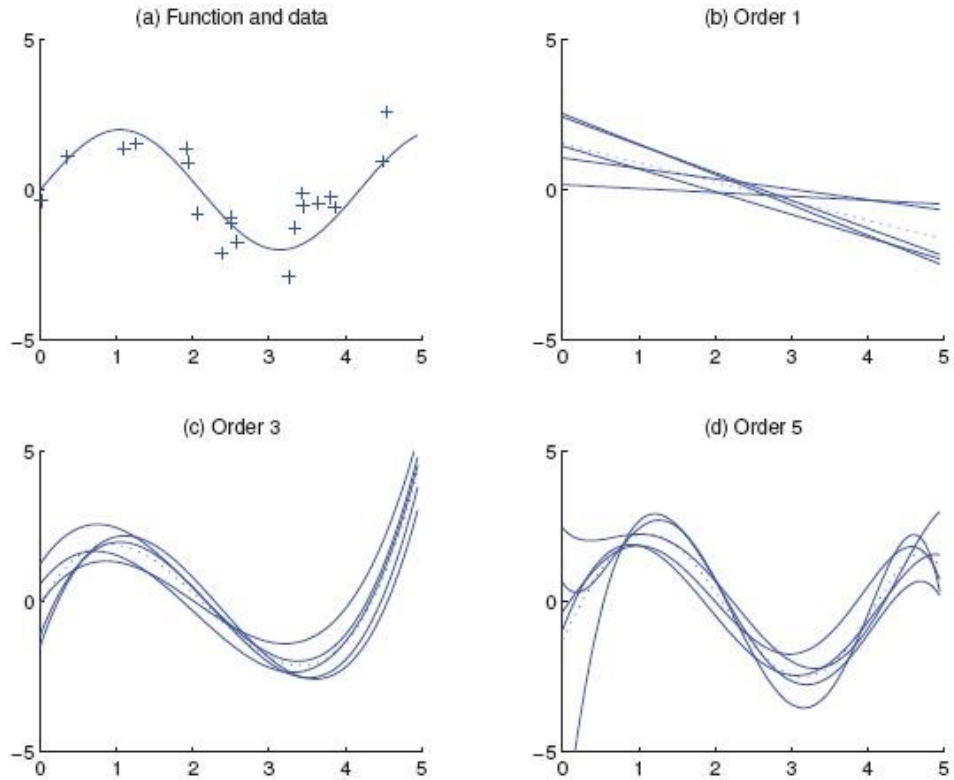
An example is a robot equipped with a camcorder having inadequate data and thus at any time is in a partially observable state. So it must choose its action taking into record this uncertainty; for instance, it may not know its exact location in a room but rather just that there is a wall to its right. Simultaneous operation of multiple agent may be required in some tasks. An example is a group of robots playing soccer.

### **1.3 Model Selection Procedures**

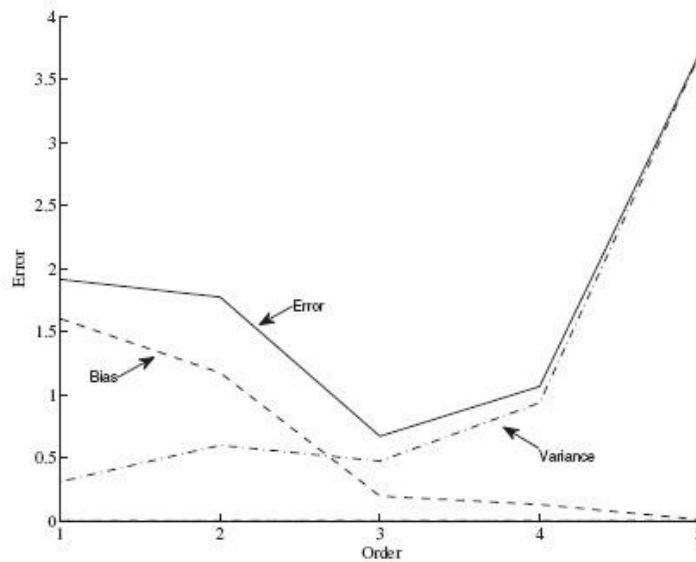
There are various strategies we can use to calibrate the model complexity.

#### **1.3.1 Cross Validation**

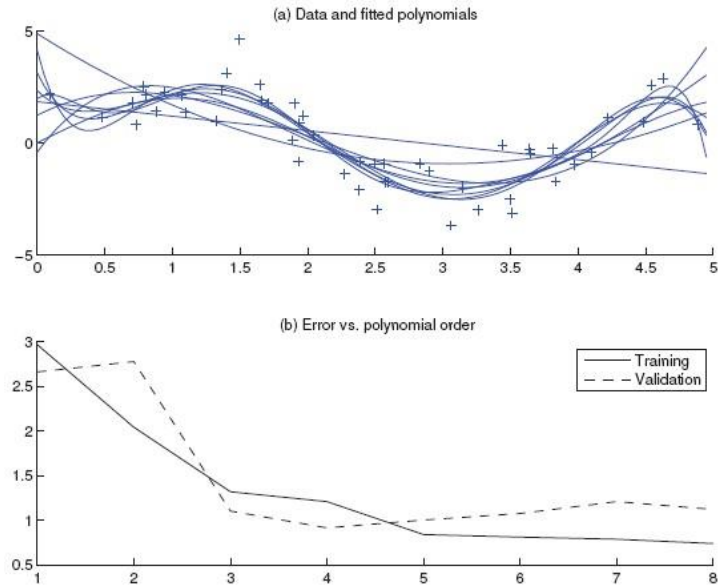
Cross validation technique is used in finding the optimal complexity for a model. The bias and fluctuation for a model can't be calculated. Thus total error is calculated by first separating the given dataset into two parts: training set and validation set; then training candidate models of various complexities on the training set and last testing their error on the validation set left out amid training. On increasing the model complexity, training error diminishes. But the error on the validation set diminishes only to a specific complexity level, after which there is no further diminishing; the error can even increase if there is noise in the data. This “elbow” corresponds to the optimal complexity level (see figure 1.6). Practically the bias and thus the error can't be computed as method used in figure 1.5; the validation error of figure 1.5 is estimate of this and also the variance of noise : Even having a correct model with no bias and sufficiently large data for variance to be immaterial does not guarantees zero validation error; there may be some nonzero validation error still present. The approval mistake of figure 1.6 is not as V-shaped as the error in figure 1.5 because of former using more training data which constrains the variance. It is found in figure 1.5 that a fifth-order polynomial acts as a third-order polynomial where there is more data – it is not as exact at the two extremes where there is less data. [2]



**Fig 1.4** (a) Function,  $f(x) = 2\sin(1.5x)$ , and one noisy ( $N(0, 1)$ ) dataset sampled from the function. Five samples are taken, each containing twenty instances. (b), (c), (d) are five polynomial fits, namely,  $g_i(\cdot)$ , of order 1, 3, and 5.



**Fig 1.5** Order 1 has the smallest variance. Order 5 has the smallest bias. As the order is increased, bias decreases but variance increases. Order 3 has the minimum error.



**Fig 1.6(a)** Training data and fitted polynomials of order from 1 to 8.  
**(b)** Training and validation errors as a function of the polynomial order. The “elbow” is at 3.

### 1.3.2 Structural Risk Minimization

Structural risk minimization (SRM) utilizes an arrangement of models minimization ordered in terms of their complexities. A case is polynomials of increasing order. The number of free parameters gives the complexity. VC dimension can be used as another measure of model complexity. SRM finds the model simplest in terms of order and best in terms of empirical error on the data. [2]

### 1.3.3 Minimum Description Length

Minimum description length (MDL) is description length in view of an information theoretic measure. MDL of a dataset is characterized as the shortest description of the data which can be easily understood. In the event of data being simple, it has a short complexity; for instance, if it is a sequence of '0's, we can simply state "0" and the length of the sequence. No description shorter than the data can be found if the data is totally random. A suitable model for the data has a solid match to the data, and rather than the data, we can send/store the model description. Out of the considerable number of models portraying the data, we need to have the least complex model with the goal that it fits the shortest description. So we again have a trade-off between keeping the model simple and good description of data.

## **1.4 Organization of report**

**Chapter 2:** This chapter is dedicated to literature review where we have discussed about how the human brain works and how by simulating the human brain behavior we can solve the real time problems.

**Chapter 3:** Various steps involved in implementing the proposed work is explained in this chapter. It contains information about the microprocessor we are using and the steps involved.

**Chapter 4:** Results are shown in this chapter and the work is concluded here.

## CHAPTER 2

### LITERATURE REVIEW

The term 'neural network' came up in attempts to find mathematical representations of information processing in biological systems. Without a doubt, it has been extensively utilized in covering a wider scope of various models, many of which have been the subject of claims regarding their biological credibility. In pattern recognition biological realism force entirely unnecessary constraints. Accordingly, neural systems are viewed as efficient models for statistical pattern recognition. Our study will thus be confined to the particular class of neural networks, the multilayer perceptron.[1]

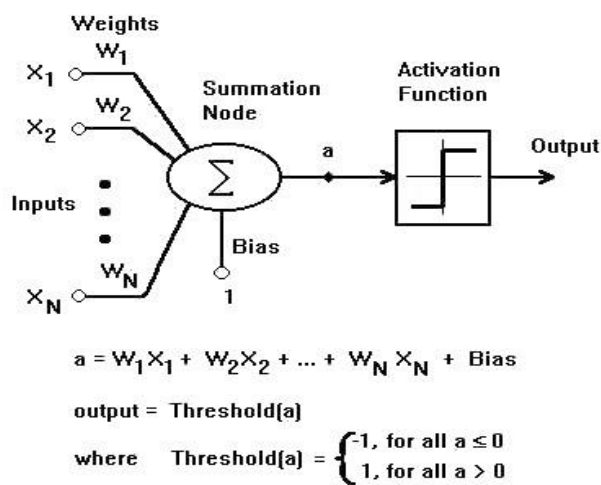
#### 2.1 Multilayer Perceptron

As animals have the ability of recognizing various things and sense out of large amount of visual information, without much effort. Thus, understanding these tasks done by animals using simulation techniques can be of high practical use for development of the system. Thus the study and simulation of Artificial Neural Network necessary. ANN models, have been majorly influenced by the human brain. Cognitive scientists and neuroscientists are the ones who understand the functioning of the and build models of the natural neural networks in the brain and make simulation studies. We believe that artificial intelligence may help us build better computer systems. The brain processes information and has some amazing abilities that cannot be compared to engineering systems—for example, vision, speech recognition, and learning, to just name three. These applications if implemented on machines in the right manner have great economic opportunity. Understanding how these functions are performed by the brain, and bringing in solutions to these tasks in the form of algorithms does make the task easier.

##### 2.1.1 Neurons

The human brain is amazingly different from a computer. The brain is composed of a very large ( $10^{11}$ ) number of processing units, namely, neurons, which operates in parallel as compared to 1 processor of the computer. These processing units are much simpler and slower than a processor in a computer. The large connectivity of these neurons is what provides the brain with its computational power and makes it different from a computer. A neuron in the brain have connections with around 10,000 other neurons, each of which is called a synapse.

In the brain, neural network comprises of both the memory and processor; neurons do the processing, and the memory is in the synapses between the neurons while in a computer, the processor is active and the memory is separate and passive. In Neural Network, a simple computation is performed by each node (neuron) and a signal from one node to another, labeled by a number called the “connection strength” or weight, is conveyed by a connection. For different weights used, the network evaluates different functions. In a network with initial weights, and given that we know what is to be done by the network, in order to get the values of the weight that will be required for the desired task, a learning algorithm must be used. Algorithm in which the system learns itself certifies the computer system to be called Artificial Neural Network. [4]



**Fig 2.1:** Typical Neural Network

Designing of a typical pattern recognition system uses two pass, the first pass is a feature extractor that finds specific features within the data to solve the desired task. The second pass is the classifier which is more of general purpose and neural network can be employed to train them. Clearly, more design effort is required by the feature extractor as it always should be hand-made based on what the application is trying to achieve.

### 2.1.2 Understanding the Brain

There are three levels in understanding an information processing system, called the levels of analysis:

1. *Computational theory* corresponding to the goal of computation and an abstract definition of the task.
2. *Representation and algorithm* corresponding to the representation of the input and the output and about the algorithm’s specification.
3. *Hardware implementation* corresponding to the actual physical realization of the system.

Let us take sorting for an example. The aim is to order a given set of elements. Integers may be used as representation, and Quicksort as the algorithm. After compilation, the suitable code for a specific processor sorting numbers depicted in binary is one hardware implementation. The thought is that for a similar process theory, there could be multiple representation and algorithms manipulating symbols in that representation. Similarly, for any given representation and algorithm, there could also be multiple hardware implementations. We are able to use one amongst varied sorting algorithms, and even a similar rule will be compiled on computers with totally different processors and result in different hardware implementations. The brain is one hardware implementation for learning or pattern recognition. If from this particular implementation, we can do reverse engineering and extract the representation and the algorithm used, and if from that in turn, we can get the computational theory, we can then use another representation and algorithm, and in turn a hardware implementation more suited to the means and constraints we have. One hopes our implementation will be cheaper, faster, and more accurate. [4]

### **2.1.3 Neural Networks and Parallel Processing**

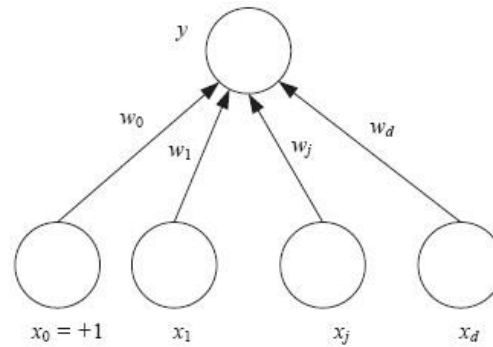
Since the 1980s, computer systems with thousands of processors have been commercially available. The software for such parallel architectures, however, has not advanced as quickly as hardware. The reason for this is that almost all our theory of computation up to that point was based on serial, one-processor machines. We are not able to use the parallel machines we have efficiently because we cannot program them efficiently.

There are mainly two paradigms for parallel processing: In single instruction, multiple data (SIMD) machines, all processors execute the same instruction but on different pieces of data. In multiple instruction, multiple data (MIMD) machines, different processors may execute different instructions on different data. SIMD machines are easier to program as there's just one program to write down. However, issues seldom have such a daily structure that they'll be parallelized over a SIMD machine. MIMD machines are more general, however it's not a simple task to write down separate programs for all the individual processors; extra issues are associated with synchronization, information transfer between processors, and so forth. SIMD machines are also easier to build, and machines with more processors can be constructed if they are SIMD. In MIMD machines, processors are more complex, and a more complex communication network should be constructed for the processors to exchange data arbitrarily.

The problem now is to distribute a task over a network of such processors and to determine the local parameter values. This is where learning comes into play: We do not need to

program such machines and determine the parameter values ourselves if such machines can learn from examples. Thus, artificial neural networks are a way to make use of the parallel hardware we can build with current technology and—thanks to learning—they need not be programmed. Therefore, we also save ourselves the effort of programming them.

## 2.2 The Perceptron



**Fig 2.2** A Simple Perceptron  $x_j, j = 1, \dots, d$  are the input units.  $x_0$  is the bias unit that always has the value 1.  $y$  is the output unit.  $w_j$  is the weight of the directed connection from input  $x_j$  to the output.[9]

The perceptron is the basic processing element. It has inputs that may come from the environment or may be the outputs of other perceptrons. Associated with each input,  $x_j \in \mathfrak{R}$ ,  $j=1, \dots, d$ , is a connection weight, or synaptic weight  $w_j \in \mathfrak{R}$  and the output,  $y$ , in the simplest case is a weighted sum of the inputs (see figure 2.2):

$$y = \sum_{j=1}^d w_j x_j + w_0 \quad (2.1)$$

$w_0$  is the intercept value to make the model more general; it is generally bias unit modeled as the weight coming from an extra bias unit,  $x_0$ , which is always +1. We can write the output of the perceptron as a dot product

$$y = w^T x \quad (2.2)$$

where  $w = [w_0, w_1, \dots, w_d]^T$  and  $x = [1, x_1, \dots, x_d]^T$  are augmented vectors to include also the bias weight and input.

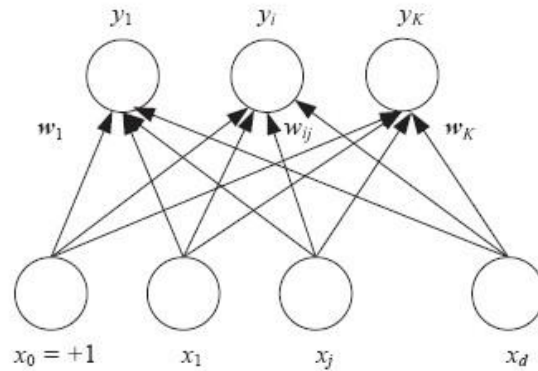
During testing, with given weights,  $w$ , for input  $x$ , we compute the output  $y$ . To implement a given task, we need to learn the weights  $w$ , the parameters of the system, such that correct outputs are generated given the inputs.

When  $d = 1$  and  $x$  is fed from the environment through an input unit, we have

$$y = wx + w_0 \quad (2.3)$$



which is the equation of a line with  $w$  as the slope and  $w_0$  as the intercept. Thus this perceptron with one input and one output can be used to implement a linear fit. With more than one input, the line becomes a (hyper) plane, and the perceptron with more than one input can be used to implement multivariate linear fit.[9]



**Fig 2.3** K parallel perceptrons.  $x_j, j = 0, \dots, d$  are the inputs and  $y_i, i = 1, \dots, K$  are the outputs.  $w_{ij}$  is the weight of the connection from input  $x_j$  to output  $y_i$ .

$$y_i = \sum_{j=1}^d w_{ij}x_j + w_{i0} = w_i^T x \quad (2.4)$$

$$y = Wx \quad (2.5)$$

When there are  $K > 2$  outputs, there are  $K$  perceptrons, each of which has a weight vector  $w_i$  where  $w_{ij}$  is the weight from input  $x_j$  to output  $y_i$ .  $W$  is the  $K \times (d+1)$  weight matrix of  $w_{ij}$  whose rows are the weight vectors of the  $K$  perceptrons.

### 2.3 Training a Perceptron

The perceptron defines a hyper plane, and also the neural network perceptron is simply some way of implementing the hyper plane. Given a data sample, the weight values can be calculated offline and then when they are plugged in, the perceptron can be used to calculate the output values.[5]

In training neural networks, we generally use online learning where we are not given the whole sample, but we are given instances one by one and would like the network to update its parameters after each instance, adapting itself slowly in time. Such an approach is interesting for a number of reasons:

1. It saves us the cost of storing the training sample in an external memory and storing the intermediate results during optimization. An approach like support vector machines may be quite costly with large samples, and in some applications, we may prefer a simpler approach

where we do not need to store the whole sample and solve a complex optimization problem on it.

2. The problem may be changing in time, which means that the sample distribution is not fixed, and a training set cannot be chosen a priority. For example, we maybe implementing a speech recognition system that adapts itself to its user.

3. There may be physical changes in the system. For example, in a robotic system, the components of the system may wear out, or sensors may degrade.

In online learning, the error function is not written over the whole sample but on individual instances. Start from random initial weights, adjust the parameters a little bit at each iteration to minimize the error, and do not forget what we have learned previously. If this error function is differentiable, we can use gradient descent.

For example, in regression the error on the single instance pair with index  $t$ ,  $(x^t, r^t)$  is

$$E^t(w|x^t, r^t) = \frac{1}{2}(y^t - r^t)^2 = \frac{1}{2}(r^t - (w^t x^t))^2 \quad (2.5)$$

and for  $j = 0, \dots, d$ , the online update is

$$\Delta w_{ij}^t = \eta(r_i^t - y_i^t)x_j^t \quad (2.6)$$

where  $\eta$  is the learning factor, which is gradually decreased in time for convergence. This is known as stochastic gradient descent.

This equation has the form

$$Update = LearningFactor \cdot (DesiredOutput - ActualOutput) \cdot Input$$

```

For  $i = 1, \dots, K$ 
  For  $j = 0, \dots, d$ 
     $w_{ij} \leftarrow \text{rand}(-0.01, 0.01)$ 
Repeat
  For all  $(x^t, r^t) \in \mathcal{X}$  in random order
    For  $i = 1, \dots, K$ 
       $o_i \leftarrow 0$ 
      For  $j = 0, \dots, d$ 
         $o_i \leftarrow o_i + w_{ij}x_j^t$ 
      For  $i = 1, \dots, K$ 
         $y_i \leftarrow \exp(o_i) / \sum_k \exp(o_k)$ 
      For  $i = 1, \dots, K$ 
        For  $j = 0, \dots, d$ 
           $w_{ij} \leftarrow w_{ij} + \eta(r_i^t - y_i)x_j^t$ 
Until convergence

```

**Fig 2.4** Perceptron training algorithm implementing stochastic online gradient descent for the case with  $K > 2$  classes.[3]

## 2.4 Learning Boolean Functions

Here, the inputs are binary and the output is 1 if the corresponding function value is true and 0 otherwise. Therefore, it can be seen as a two-class classification problem. As an example, two inputs AND, the table of inputs and required outputs is given in table 2.1.

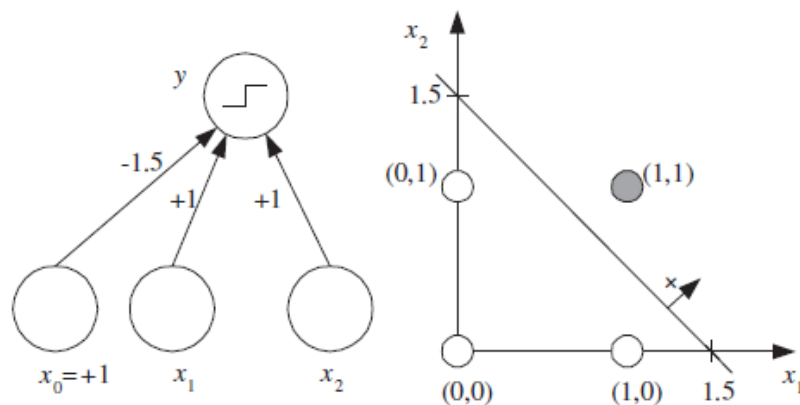
| $x_1$ | $x_2$ | $r$ |
|-------|-------|-----|
| 0     | 0     | 0   |
| 0     | 1     | 0   |
| 1     | 0     | 0   |
| 1     | 1     | 1   |

**Table 2.1** AND gate

and an example of a perceptron that implements AND, and its geometric interpretation in two dimensions is given in figure 2.5. The discriminant is

$$y = s(x_1 + x_2 - 1.5) \quad (2.7)$$

that is,  $x = [1, x_1, x_2]^T$  and  $w = [-1.5, 1, 1]^T$ .



**Fig 2.5** The perceptron that implements AND and its geometric interpretation.

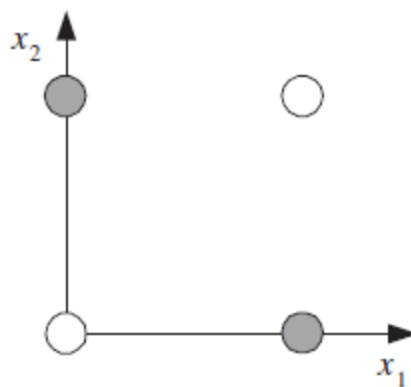
Though Boolean functions like AND and OR are linearly separable and are solvable using the perceptron, certain functions like XOR are not. The table for XOR is given in table 2.2.

| $x_1$ | $x_2$ | $r$ |
|-------|-------|-----|
| 0     | 0     | 0   |
| 0     | 1     | 1   |
| 1     | 0     | 1   |
| 1     | 1     | 0   |

**Table 2.2** XOR gate

and its geometric representation is given in fig 2.6

A perceptron having a single layer of weights can only approximate linear functions of the input and cannot solve problems like the XOR, where the discriminant to be estimated is nonlinear. Similarly, a perceptron cannot be used for nonlinear regression. This limitation does not apply to feed forward networks with intermediate or hidden layers between the input and the output layers. If used for classification, such multilayer perceptrons (MLP) can implement nonlinear discriminants and, if used for regression, can approximate nonlinear functions of the input.



**Fig 2.6** XOR problem is not linearly separable. We cannot draw a line where the empty circles are on one side and the filled circles on the other side.[3]

## CHAPTER 3

### IMPLEMENTATION

#### 3.1 Back propagation Algorithm

Training a MLP is the same as training a perceptron with the only difference of the output being a nonlinear function of the input due to the nonlinear basis function in the hidden units. The hidden units is taken as inputs, while the second layer as a perceptron and it is already know how to update the parameters,  $v_{ij}$  in this case, given the inputs  $z_h$ . Chain rule is used to calculate the gradient for the first layer weights  $w_{hj}$ :

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}} \quad (3.1)$$

This looks as the error is propagating back to the inputs from the output  $y$  and hence the name back propagation. A known, desired output for each input value is requires in calculating the loss function gradient. Thus considered to be a supervised learning method; nonetheless, it is likewise utilized as a part of some unsupervised network. It is a generalization of the delta rule to multi-layered feed forward networks, made possible by utilizing the chain rule to iteratively compute gradients for each layer. The activation function used by the artificial neurons must be differentiable.

The back propagation learning algorithm can be divided into two phases: propagation and weight update. [11]

##### ***Propagation Phase***

Each propagation involves the following steps:

1. Forward propagation of an input of a training pattern through the neural network to generate the propagation's output activations.
2. Backward propagation of the propagation's output activations through the neural network using the training pattern target to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

##### ***Weight update Phase***

For each weight-synapse follow the following steps:

1. Multiply its output delta and input activation to get the gradient of the weight.
2. Subtract a ratio (percentage) from the gradient of the weight.

The speed and learning quality is determined by this ratio (percentage) and is called the learning rate. Higher ratio corresponds to faster training of the neuron, while the lower ratio to more accurate training. The sign of the gradient of a weight indicates where the error

is increasing, this is why the weight must be updated in the opposite direction. Repeat phase 1 and 2 until the performance of the network is satisfactory.

## 3.2 Training Procedures

### 3.2.1 Improving Convergence

There are various advantages of gradient descent. Being straightforward is one. Also it is local; in particular, the adjustment in a weight utilizes just the estimations of the presynaptic and postsynaptic units and the error. At the point when web-based training is utilized, storage of the training set is not necessary and can adapt as the task to be learned changes. Hence hardware implementation is possible on account of these reasons. But gradient descent converges slowly. When learning time is vital, utilization of more modern streamlining techniques can be done. Two frequently straightforward techniques are utilized to enhance the performance of the gradient descent considerably, making this method feasible in real applications.

#### Momentum

Let us say  $w_i$  is any weight in a multilayer perceptron in any layer, including the biases. At each parameter update, successive  $\Delta w_t^i$  values may be so different that large oscillations and slow convergence may occur.  $t$  being the time index that is the epoch number in batch learning and the iteration number in online learning. The idea is to take a running average by incorporating the previous update in the current change as if there is a momentum due to previous updates:

$$\Delta w_t^i = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1} \quad (3.2)$$

$\alpha$  is generally between 0.5 and 1.0. This approach is useful when online learning is used. The disadvantage is that the past  $\Delta w_i^{t-1}$  values should be stored in extra memory.

#### Adaptive Learning Rate

The magnitude of change to be made in the parameter, in gradient descent, is determined by the learning factor  $\eta$ . It is generally taken between 0.0 and 1.0, mostly less than or equal to 0.2. It can be made adaptive for faster convergence, where it is kept large when learning takes place and is decreased when learning slows down:

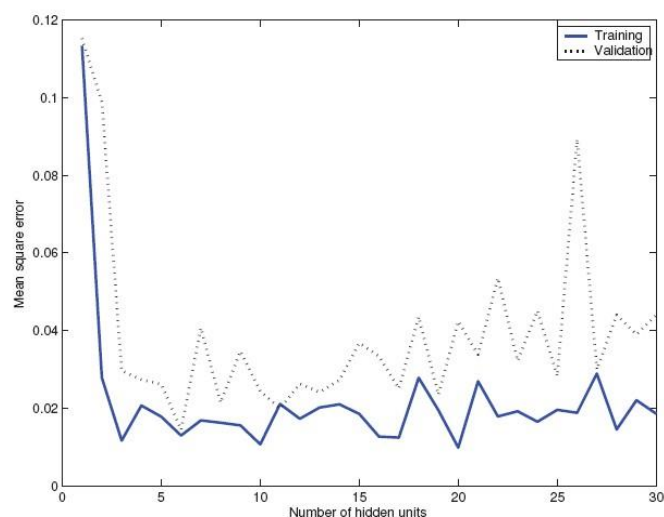
$$\Delta \eta = \begin{cases} +a & \text{if } E^{t+\tau} < E^t \\ -b\eta & \text{otherwise} \end{cases} \quad (3.3)$$

Thus if the error on the training set decreases  $\eta$  is increased by a constant amount; but decreased geometrically if it increases. Due to oscillation of  $E$  from one epoch to another, it is better to take the average of the past few epochs as  $E^t$ .

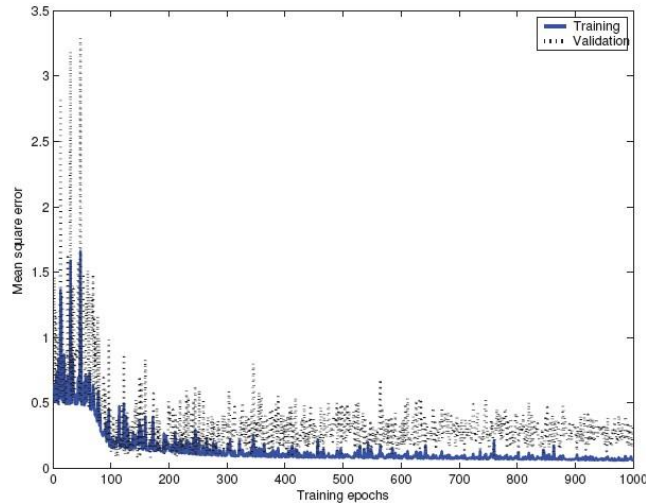
### 3.2.2 Overtraining

Consider a MLP having  $d$  inputs,  $H$  hidden units, and  $K$  outputs has  $H(d+1)$  weights in the first layer and  $K(H+1)$  weights in the second layer. Both the space and time complexity of an MLP is  $O(H \cdot (K + d))$ . Training time complexity is  $O(e \cdot H \cdot (K + d))$ , where  $e$  denotes the number of training epochs. In an application,  $d$  and  $K$  are predefined. To tune the complexity of the model we play with the  $H$  parameter. From previous discussions it is known that an over complex model memorizes the noise in the training set and does not generalize to the validation set. In an MLP, when the number of hidden units is large, the generalization accuracy deteriorates (see fig 3.1), and the bias/variance dilemma also holds for the MLP, as it does for any statistical estimator.

When training is prolonged, a similar behavior happens: The error on the training set decreases as more training epochs are made, but the validation set error starts to increase beyond a certain point (see fig 3.2). Initially all the weights are close to 0 and thus have little effect. As training continues, the most important weights start moving away from 0 and are utilized. But if continued further, almost all weights are updated away from 0 and effectively become parameters. To assess expected error, the same network is to be trained a number of times starting from different initial weight values, and the average of the validation error is computed.



**Fig 3.1** As complexity increases, training error is fixed but the validation error starts to increase and the network starts to over fit.



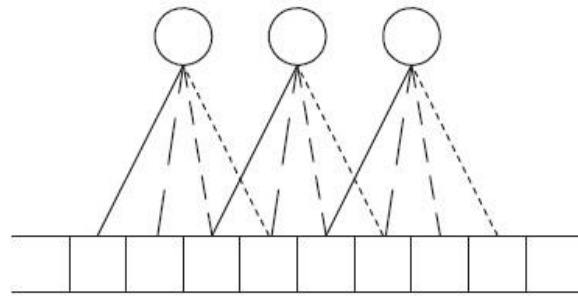
**Fig 3.2** As training continues, the validation error starts to increase and the network starts to over fit.[10]

### 3.2.3 Structuring the Network

In a few applications, we may believe that the input has a local structure. For instance, in vision we realize that adjacent pixels are related and there are local features like edges and corners. In such a situation when designing the MLP, hidden units are not connected to all input units because not all inputs are correlated. Rather, we define hidden units that define a window over the input space and are connected to only a small local subset of the inputs. This declines the number of associations and accordingly the number of free parameters.

This can be repeated in successive layers where each layer is connected to a small number of local units below and checks for a more complicated feature by combining the features below in a larger part of the input space until we get to the output layer. For example, the input may be pixels. By looking at pixels, the first hidden layer units may learn to check for edges of various orientations. Then by combining edges, the second hidden layer units can learn to check for combinations of edges—for example, arcs, corners, line ends—and then combining them in upper layers, the units can look for semi-circles, rectangles, or in the case of a face recognition application, eyes, mouth, and so forth. This is the example of a hierarchical cone where features get more complex, abstract, and fewer in number as we go up the network until we get to classes. Such an architecture is called a convolutional neural network where the work of each hidden unit is considered to be a convolution of its input with its weight vector; an earlier similar architecture is the neocognitron.





**Fig 3.3** In weight sharing, different units have connections to different inputs but share the same weight value (denoted by line type).

Weight sharing can further reduce the number of parameters. In visual recognition, features like oriented edges may be present in different parts of the input space. So instead of defining independent hidden units learning different features in different parts of the input space, we can have copies of the same hidden units looking at different parts of the input space.

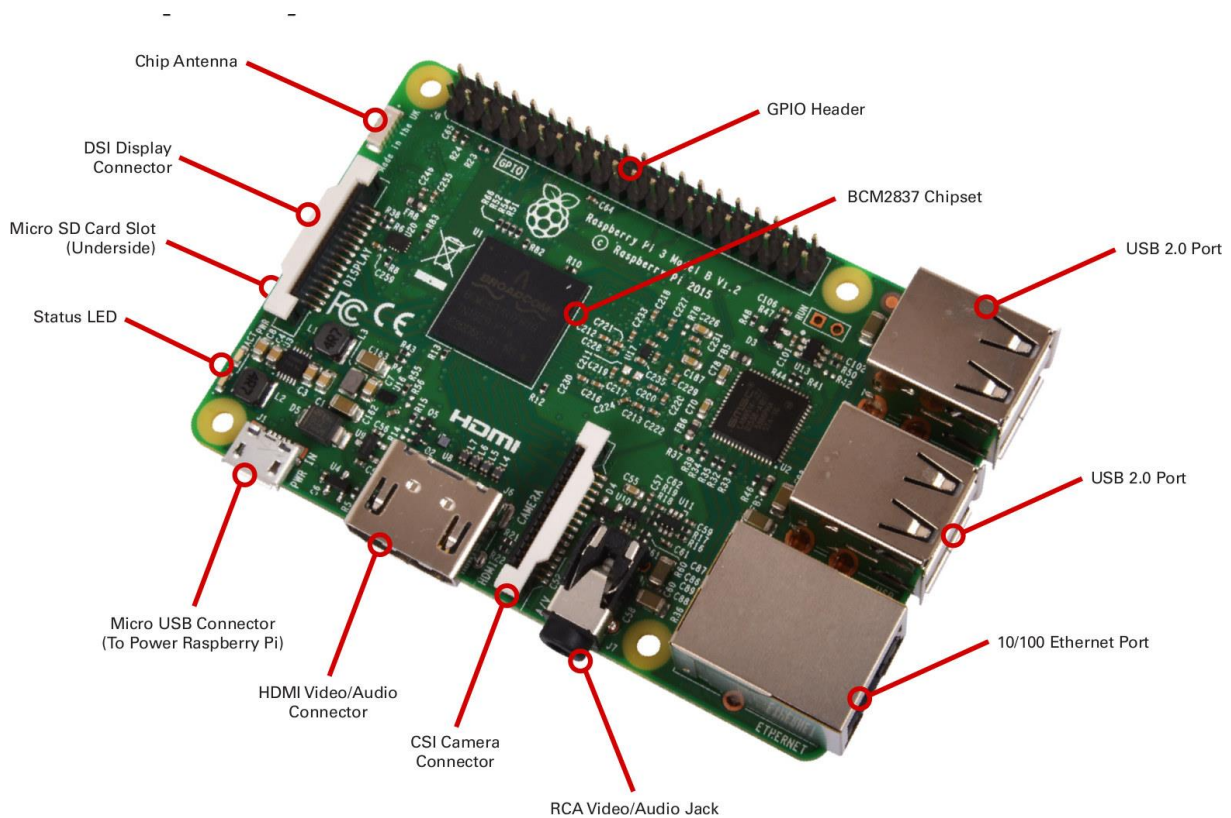
### 3.3 Tools for design

#### 3.3.1 Raspberry Pi

Raspberry Pi is a credit card-sized computer powered by the Broadcom BCM2835 system-on-a-chip (SoC). This SoC includes a 32-bit ARM1176JZFS processor, clocked at 700MHz, and a Videocore IV GPU. It also has 256MB of RAM in a POP package above the SoC. It is powered either by a 5V micro USB AC charger or at least 4 AA batteries (with some special circuitry). The Raspberry Pi is a small, barebones computer developed by The Raspberry Pi Foundation, a UK charity. Their intention was to provide low-cost computers and free software to students. Their ultimate goal is to take computer science education to a new level and they hope that this small, affordable computer will be a tool that enables that. The Raspberry Pi is slower than a modern laptop or desktop but is still a complete Linux computer and can provide all the expected abilities that implies, at a low-power consumption level. It is open hardware, with the exception of the primary chip on the Raspberry Pi, the Broadcom SoC, which runs many of the main components of the board—CPU, graphics, memory, the USB controller, etc. Many of the projects made with a Raspberry Pi are open and well-documented as well and are things you can build and modify yourself.

The Raspberry Pi was originally designed for the Linux operating system, and many Linux distributions now have a version optimized for the Raspberry Pi. Two of the most popular options are Raspbian, which is based on the Debian operating system, and Pidora, which is based on the Fedora operating system. For beginners, either of these two work well; which one

we choose to use is a matter of personal preference. A good practice might be to go with the one which most closely resembles an operating system we are familiar with, in either a desktop or server environment. There are, of course, lots of other choices. OpenELEC and Rasp BMC are both operating system distributions based on Linux that are targeted towards using the Raspberry Pi as a media center. There are also non-Linux systems, like RISC OS, which run on the Pi. Some enthusiasts have even used the Raspberry Pi to learn about operating systems by designing their own.



**Fig. 3.4** Raspberry Pi 3

### 3.3.2 Processor

The Broadcom BCM2835 SoC used in the first generation Raspberry Pi is somewhat equivalent to the chip used in first generation smartphones (its CPU is an older ARMv6 architecture), which includes a 700 MHz ARM1176JZF-S processor, VideoCore IV graphics processing unit (GPU), and RAM. It has a level 1 (L1) cache of 16 KB and a level 2 (L2) cache of 128 KB. The level 2 cache is used primarily by the GPU. The SoC is stacked underneath the RAM chip, so only its edge is visible. The Raspberry Pi 2 uses a Broadcom BCM2836 SoC with a 900 MHz 32-bit quad-core ARM Cortex-A7 processor (as do many current smartphones), with 256 KB : shared L2 cache while

the Raspberry Pi 3 uses a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, with 512 KB shared L2 cache.

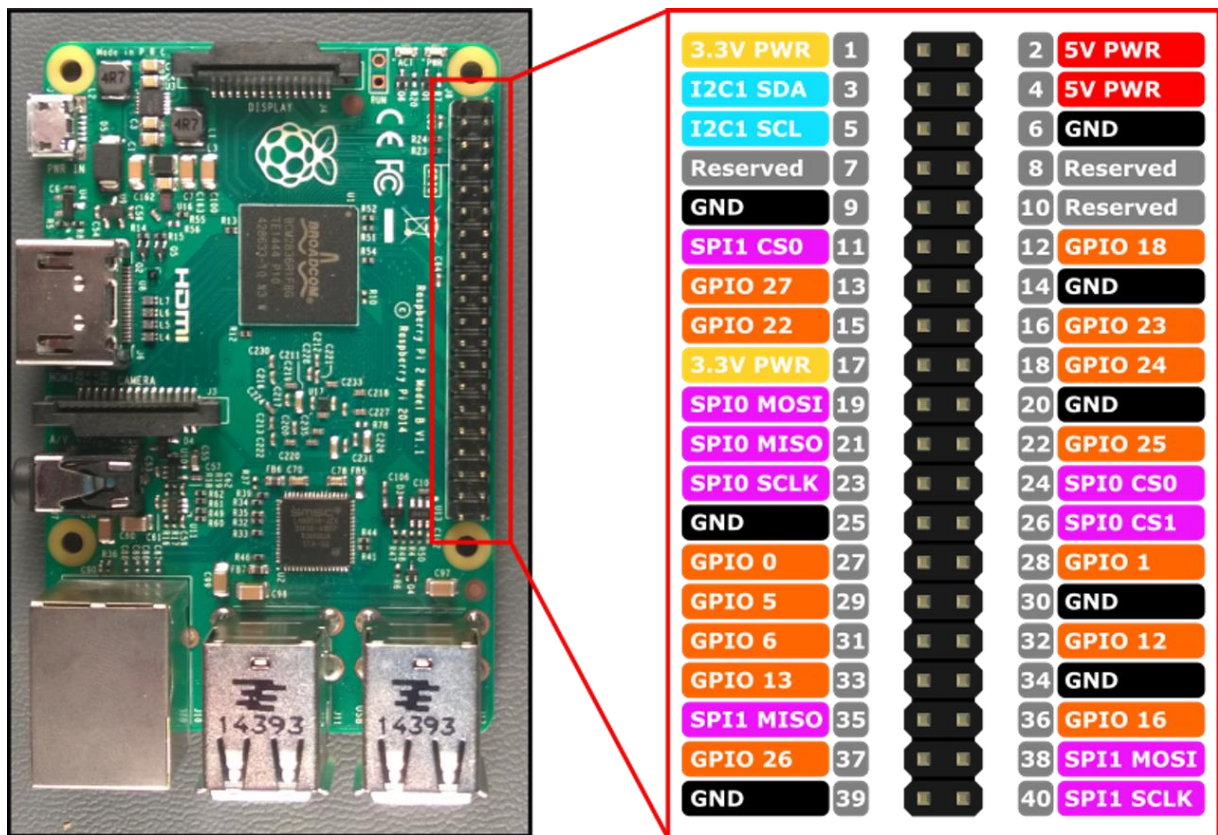


Fig.3.5 Raspberry PI 3 Pin OUT Diagram

### 3.4 Steps in optical character recognition

Optical character recognition involves various phases from image acquisition to the classification and recognition (fig 3.6). The output of a step is the input to the next step. The task of preprocessing relates to the removal of noise and variation in handwritten word patterns. Preprocessing is further broken down into smaller tasks such as noise removal, binarization, thinning, edge detection, etc. to enhance the quality of images and to correct distortion.

#### 3.4.1 Image Acquisition

A camera/scanner is used in this phase to take an input image. The acquired image must be of a specific format such as JPEG; PNG; BMP etc. and may be colored or gray scale.

### 3.4.2 Preprocessing

It is a series of operations performed on the acquired image. It is a necessary step in character recognition as it enhances the image and removes the noise which may cause errors in the corresponding steps. The various operations are:

- (1) **Filtering:** Due to uneven writing surface and/or low sampling rate, there may be some noise introduced in the acquired image. This operation aims at removing this noise by using various types of filters.
- (2) **Binarization:** This operation converts the grayscale image into binary image.
- (3) **Edge Detection:** Character boundaries are characterized by the edges. So the detection of edges is useful in character recognition. It reduces the amount of data by filtering the useless data while preserving the important structural properties of the image.

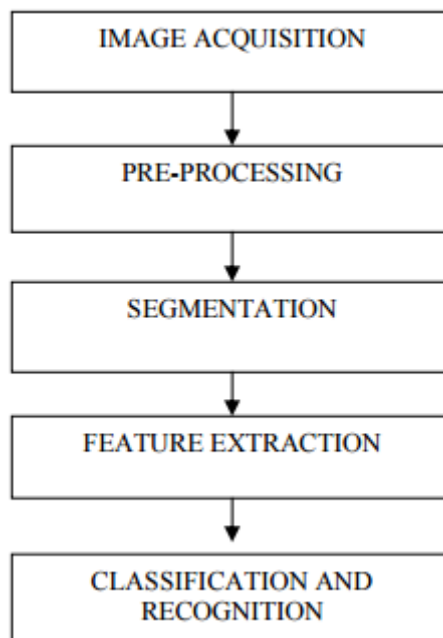


Fig 3.6 Stages in OCR

- (4) **Thresholding:** Grey scale images are represented as binary images by setting a threshold value and everything that lies above this threshold is set to 1 and everything below is set to 0.
- (5) **Skew Detection:** There can be skewness involved while scanning the input image. The skewness should be detected and removed as it tends to reduce the accuracy of the document. The skewed lines are made horizontal making use of the skew angle.[13]

### 3.4.3 Segmentation

It is the most important phase in character recognition. In this phase, spaces are provided in between the individual characters in the image. [14]

### 3.4.4 Feature Extraction

Feature extraction aims at extracting a set of features which maximizes the recognition rate with least amount of effort. We know that every individual has different handwriting but there are certain features in each character which remains the same throughout. These features can be extracted and utilized in the recognition process. [15]

### 3.4.5 Classification and Recognition

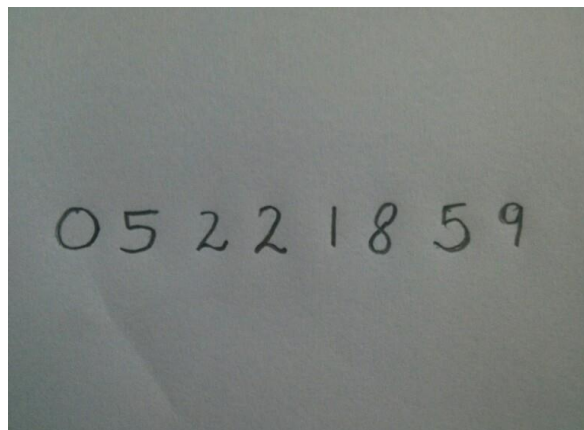
It is the decision making part which uses the extracted features to classify the characters and give the output.

## 3.5 Input

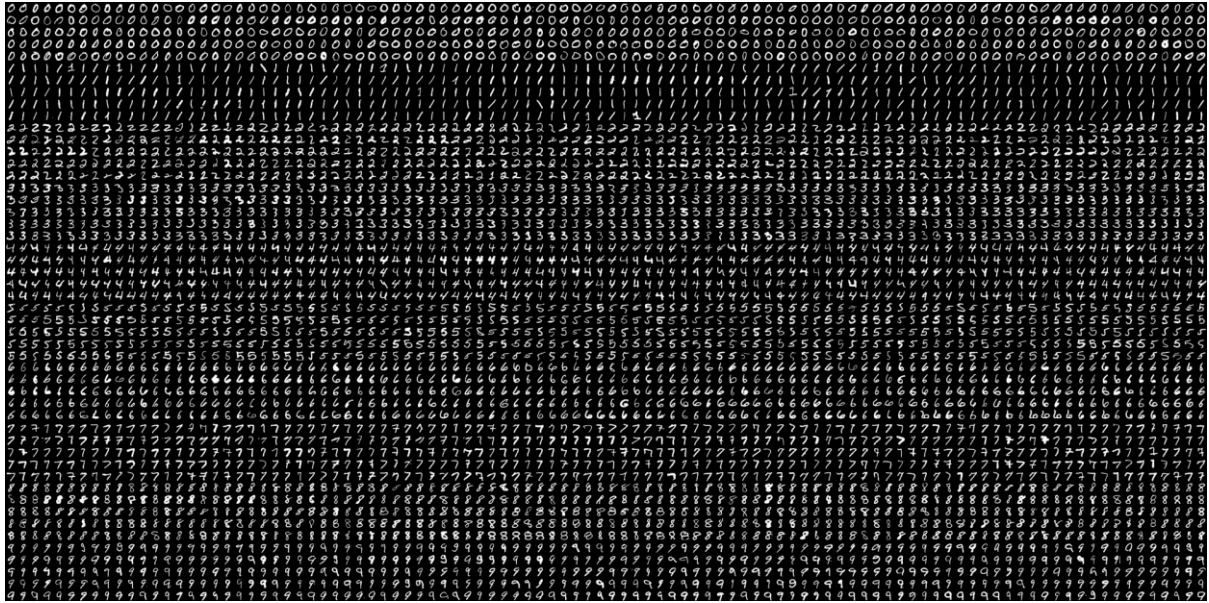
Handwritten data (fig 3.7) is taken and image acquisition is done by the camera using the PI. Since it is using unsupervised learning so the network is first trained using a sample set (fig 3.8).

## 3.6 Training

The first step is to create a database of handwritten digits. The popular MNIST database of handwritten digits is used which is a set of 70000 samples of handwritten digits where each sample consists of a grayscale image of size 28×28. We will use sklearn datasets package to download the MNIST database from mldata.org. This package makes it convenient to work with toy databases. Two python scripts will be written; one for training the classifier and the second for testing it.



**Fig 3.7** The input data



**Fig 3.8** An example training set for unsupervised learning

For training we will implement the following steps –

1. Calculate the HOG features for each sample in the database.
2. Train a multi-class linear SVM with the HOG features of each sample along with the corresponding label.
3. Save the classifier in a file.



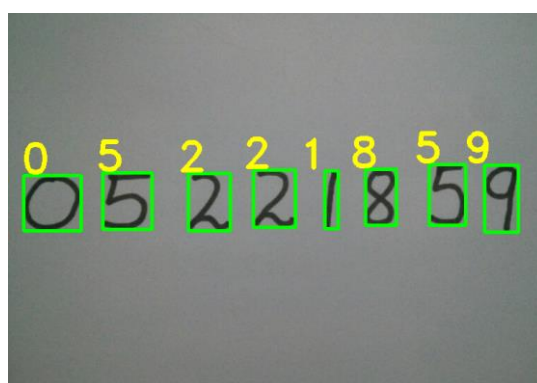
## CHAPTER 4

### RESULTS AND CONCLUSIONS

The output after the compilation of code and processing is done on the input is shown in fig

4.1. There are a few assumptions, we have assumed in the testing images –

1. The digits should be sufficiently apart from each other else they will interfere in the square region around each digit. In that case, we create a new square image and then copy the contour in that square image.
2. For the images used in testing, fixed thresholding worked pretty well which is not the case in most real world images. There, we need to use adaptive thresholding.
3. In the pre-processing step, we only did Gaussian blurring. In most situations, on the binary image we will need to open and close the image to remove small noise pixels and fill small holes.



**Fig 4.1** The output

The characters from 0 to 9 are recognized by the project. It can detect these character with an accuracy of about 95%. In future, it can be extended to recognize the alphabets and the special characters. Many different methods have been explored by the scientists over past few decades. A variety of approaches have been proposed and tested by researchers in different parts of the world. No OCR in this world is 100% accurate. The recognition accuracy of the neural networks proposed here can be further improved. The number of character set used for training is reasonably low and the accuracy of the network can be increased by taking more training character sets. This approach can be used for recognition of Hindi, Bengali etc. characters.


## REFERENCES

- [1]. Bishop, C. M. 1995. *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press.
- [2]. Webb, A., and K. D. Copsey. 2011. *Statistical Pattern Recognition*, 3rd ed. New York: Wiley
- [3]. *Introduction to Machine Learning 3 Edition* by Ethem Alpaydin
- [4]. Aha, D. W., ed. 1997. Special Issue on Lazy Learning. *Artificial Intelligence Review*
- [5]. Aha, D. W., D. Kibler, and M. K. Albert. 1991. "Instance-Based Learning Algorithm." *Machine Learning* 6:37–66.
- [6]. Atkeson, C. G., A. W. Moore, and S. Schaal. 1997. "Locally Weighted Learning." *Artificial Intelligence Review*
- [7]. Stanfill, C., and D. Waltz. 1986. "Toward Memory-Based Reasoning." *Communications of the ACM*.
- [8]. Abu-Mostafa, Y. 1995. "Hints." *Neural Computation*
- [9]. Ash, T. 1989. "Dynamic Node Creation in Backpropagation Networks." *Connection Science*.
- [10]. Battiti, R. 1992. "First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method." *Neural Computation*
- [11]. Durbin, R., and D. E. Rumelhart. 1989. "Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks." *Neural Computation*
- [12]. Hertz, J., A. Krogh, and R. G. Palmer. 1991. *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley.
- [13]. Le Cun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. "Backpropagation Applied to Handwritten Zipcode Recognition." *Neural Computation*
- [14]. <https://www.ijsr.net/archive/v2i1/IJSR13010129.pdf>
- [15]. <http://yann.lecun.com/exdb/publis/pdf/matan-90.pdf>



## Submission Info

|                  |                         |
|------------------|-------------------------|
| SUBMISSION ID    | 807645821               |
| SUBMISSION DATE  | 01-May-2017 12:15       |
| SUBMISSION COUNT | 1                       |
| FILE NAME        | Project_Report_Achmn... |
| FILE SIZE        | 2.13M                   |
| CHARACTER COUNT  | 46815                   |
| WORD COUNT       | 8363                    |
| PAGE COUNT       | 40                      |
| ORIGINALITY      |                         |
| OVERALL          | 27%                     |
| INTERNET         | 24%                     |
| PUBLICATIONS     | 4%                      |
| STUDENT PAPERS   | 13%                     |
| GRADEMARK        |                         |
| LAST GRADED      | N/A                     |
| COMMENTS         | 0                       |
| QUICKMARKS       |                         |



LIBRARIAN  
LIBRARY RESOURCE CENTER  
Jaypee University of Information Technology  
Waknaghat, Distt. Solan (Himachal Pradesh)  
Pin Code: 173204