POLITECNICO DI TORINO

Department of Control and Computer Engineering

Masters of Science in Mechatronics

Master Thesis Report

Experimental Validation of Back Propagation Algorithms for Pattern Recognition

Supervisor:	Co-supervisor:	Candidate:
Prof. Paolo Ernesto Prinetto	Dr. Giuseppe Airo Farulla	Sajjad Ali
(Politecnico di Torino)	(Politecnico di Torino)	S213855

March 2017

ABSTRACT

Technology has offered so many good things to the human race, from organ transplants in medical, the miracles of biometrics, Artificial Intelligence and to the communication systems, science and technology have always amused us. However, the best thing that I found good about technology is that it can help those who are an inevitable part of our societies; here I am talking about people with sensory disabilities. They need more attention, more focus and more help than normal humans do. This proposed work is a part of a bigger project that deals with developing intelligent systems for people with sensory disabilities. An Artificial Neural Networks based Optical Character recognition system has to be developed that will be used for learning and education of visually impaired people

This proposed thesis entitled "Experimental validation of Back Propagation Algorithms for Pattern Recognition" is a completely experimental work based on ANNs and Back Propagation algorithm, which required many test and validation sessions. In this thesis, we focus on feedforward neural networks trained by using the Backpropagation (BP) algorithm, which is a widely used method of training. This research work consists all the major activities from creating data sets (Both training and validation), training the ANN and then testing it. Another major activity was the optimization of the parameters of the ANN and BP algorithm. The results acquired after optimization are very interesting and helpful for the future works. This proposed thesis will prove to be a big help in carrying out the next phases of the project in a fast manner.

Keywords: Optical Character Recognition, Artificial Neural Networks, Back Propagation Algorithm

ACKNOWLEDGEMENTS

It has been my privilege to work in a research environments with a wonderful array of mentors and intellectuals.

Thanks to my thesis supervisor Prof. Paolo Ernesto Prinetto for his support. His amicableness and vigorousness impress me a lot. I have benefited enormously from working with him.

Thanks to Prof. Anna Capietto, for her continues support during my stay at UNITO.

Thanks to Prof. Nadir Murru for his support and guidance at every step, and for helping me out whenever I needed him.

Thanks to Prof. Giuseppe Airò Farulla for suggesting this research work to me and encouraging me for every good thing that I did during this research. His help and support has always been there.

For the past two years, I am grateful to study in Politecnico di Torino, with excellent faculty in a stimulating academic atmosphere. All of these have a great impact on my development in my research work. I would like to thank all faculties for their considered guidance and perspectives.

Thanks to my parents, who keep on supporting me faithfully in my life.

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

Chapter 1: Introduction

Many forms of technology, both "high" and "low" can help individuals with learning disabilities to capitalize on their strengths and bypass, or compensate for, their disabilities. Such technologies are called "Assistive Technologies". All the technologies that can enhance the performance of people with disabilities can be conceptualized as Assistive technologies. As defined by the Individuals with Disabilities Education Act Amendments of 1997, assistive technology is "any item, piece of equipment, or product system . . . that is used to increase, maintain, or improve functional capabilities of individuals with disabilities".

Assistive technology is an umbrella term that comprises assistive, adaptive, and rehabilitative devices for people with disabilities and encompasses the process used in selecting, locating, and using them. Assistive technology upholds greater independence by enabling people to accomplish tasks that they were formerly unable to achieve, or had great difficulty accomplishing, by providing enhancements to, or changing methods of interacting with, the technology needed to accomplish such tasks.

This proposed thesis is a part of a big project in the setting of the agreement between I.Ri.Fo.R./UICI (Institute for Research, Education and Rehabilitation/Italian Union of Blind and Low Vision people) and the University of Turin, Which is related to assistive technologies for visually impaired people. The idea is to develop an Optical Character Recognition (OCR) system, which can help the people with visual impairments in education and learning. An OCR transforms a not-editable format (usually not accessible with screen reader and braille display) into an editable one such that it can be made accessible. The main objective of this OCR would be automatically generating texts and

mathematical formulas for people with visual disabilities. Prior to this work, a Japanese software development firm has already launched an OCR of this kind but it works on Vector scaling algorithm. The Japanese OCR known as **InftyReader** is the unique one that performs automatic recognition of documents containing also formulae. Its performances are not ever optimal (for instance, bold characters are not correctly recognized and some further errors are performed in the recognition of mathematical formulae). Our OCR is based on Artificial Neural Networks (ANNs) and Back-Propagation ((BP) Algorithm.

From the very beginning, the idea was to develop a model that works as a human nervous system, which consists of neurons and works exactly in the same way as a human brain does and have more or less same information processing capabilities that a human brain can have. Therefore, we must understand the anatomy and the essential properties of the nervous system (i.e. Biological Neural Networks). After successfully understanding the properties of a human brain, we are able to design abstract models of the Artificial Neural Networks (ANNs).

We discuss the perceptron that in theory is a mathematical counterpart of biological neuron. One can perform very simple tasks using single perceptron, but for carrying out complex operations, we need a network of multiple neurons. Multiple layer neural networks are discussed in detail in Chapter 2. An example of XOR problem is also discussed in this chapter.

A prior study on this topic was related to developing an ANN using BP algorithm that could perform character recognition and people succeeded to do so but validation of the ANN was inevitable to proceed further. This thesis will prove to be a crucial step for doing so. BP algorithm consists of some important parameters, which should be optimized for

the best performance of ANN and in this proposed thesis, optimization of those parameters has been performed to validate the ANN. It is important to discuss the parameters of ANN with BP algorithm like learning rate (eta), number of hidden neurons (N), the interval in which the weights are initialized (h) and the problems related to these parameters. The parameters must be optimized to get the best performance of the ANN.

The thesis contains all the steps of validating an ANN such as, creating training sets, training the ANN and finally, testing it. A well-known issue about the creation of OCRs using ANNs is the tuning of some parameters that affect performances. Moreover, it is also difficult to find data sets that can be used to train an ANN for character recognition.

Therefore, in this thesis, we start to solve these problems without considering the mathematical symbols, since the results obtained only considering English characters are useful for addressing the previous problems also in presence of formulae.

First Chapter is Introduction, which comprises a brief introduction of the thesis and the related projects. We can regard this chapter as a summary of the whole thesis.

Second chapter is an overview on Artificial Neural Networks (ANNs), which can give a lot of information on ANNs and their biological counter parts, the information about neurons and perceptron is very interesting. Furthermore, it is discussed, how to generate Multi-layer Feed Forward Networks using multiple perceptron. XOR Problem is also discussed in this chapter.

In third chapter, i.e., Training Algorithms and ANNs, we discuss supervised and unsupervised training, and then we focus on supervised training and discuss Back-

Propagation Algorithm in detail, with notations and a brief Proof of BP algorithm. Then we discuss the problems that one can face when using BP algorithm.

Fourth chapter, i.e., Application to Character Recognition, comprises of the steps involved in character recognition and MatLAB functions that we use for creating data sets, training and testing ANNs and experimental results. It has all the detailed results in form of tables and surface graphs.

In fifth chapter, we conclude our report, with some interesting results and future work that is to be done in the concerned project.

.

Chapter 2: An Overview of Neural Networks

2.1. Biological and Artificial Neural Networks

From the very beginning, the idea was to develop a model that works as a human nervous system, which consists of neurons and works exactly in the same way as a human brain does and have more or less same information processing capabilities that a human brain can have. Therefore, we must understand the anatomy and the essential properties of the nervous system (i.e. Biological Neural Networks). After successfully understanding the properties of a human brain, we will be able to design abstract models of the artificial neural networks.

From the studies, it is evident that a human brain is a complex organ that works on the principle of "control through communication" between neurons. Neurons are interconnected through wire like veins, so, it makes a humongous network of billions of neurons. Neurons are slower than logic gates that we use in our computing devices; they can achieve stimulus or reaction times of few milliseconds whereas logic gates switch in nanoseconds. However, the brain can solve the problems that even computers are not able to solve. Let us now try to understand the phenomenon of information transfer from one neuron to another neuron in detailed fashion.

The sensory system of a human is may be a data transforming system. The tangible inputs, i.e., signals from the environment, can be coded and transformed that bring out those fitting reactions. Moreover, Biological Neural networks are self-organizing frameworks; also, every single neuron is a fragile, self-organizing structure skilled for information processing of the data at very large scale.

2.1.1.Nervous system

The ability to cop up with your surroundings, to see, to hear, to feel, to smell relies on your nervous system. So eventually, whatever you perceive from the environment is possible because of a complete, complex communication system that connects your brain with other body parts. The major source of this connectivity are interconnected nerve cells and Glial cells i.e. Neurons and Glia. (Byrne, J. H. 2016)

Neurons are essential practical units of the sensory system; furthermore, they produce electrical signals known as action potentials, which permit them to transmit data in long distances. Glia also plays an important role in stabilizing the nervous system but they work to support neurons.

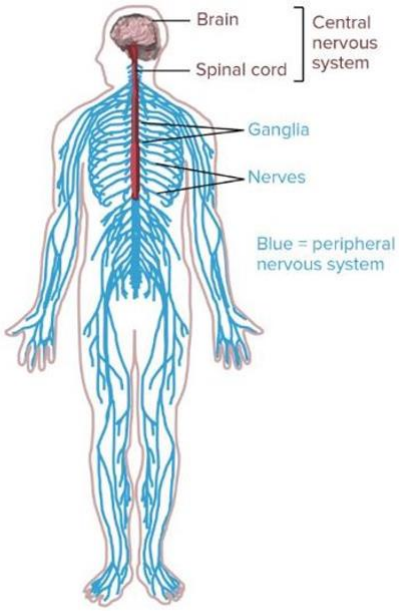

Figure 2-1 Nervous System Diagram

Before discussing the structure of neuron in details, let us have brief introduction of human nervous system. (Kendel et al., 1995)

In humans and other species, the sensory system can be comprehensively classified under two sections: Central sensory system and the Peripheral sensory system.

The Central sensory system (CNS) comprises of the mind and the spine. It may be in the CNS that analyses all the information. (Purves, D. et al., 1997).

Peripheral sensory system (PNS), which comprises of the neurons and parts of neurons discovered outside of the CNS, incorporates sensory neurons and motor neurons

2.1.2. Classes of Neurons

According to their functions, neurons present in a nervous system can be classified into 3 types

- Sensory Neurons
- Motor Neurons
- Interneurons

2.1.2.1. Sensory Neurons

Sensory neurons get majority of the data regarding what is happening inside and outside of the body and pass on that information to the CNS for processing. For example, if you accidently touch a hot plate, the sensory neurons at the outer layer of your skin will pass the information to your CNS that plate was hot. . (Nicholls, J. G. et al., 2001)

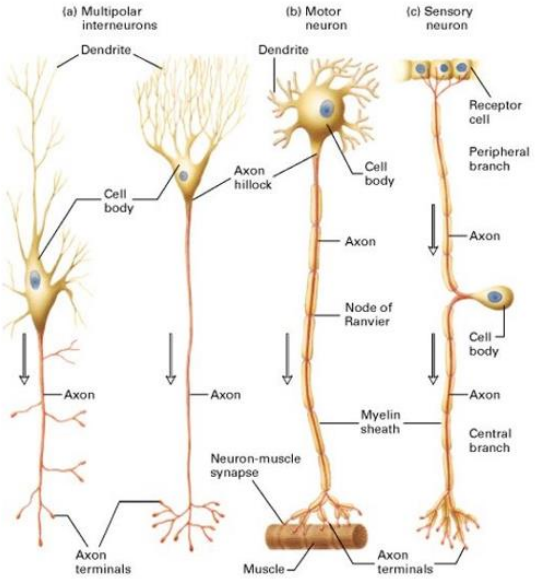

Figure 2-2 Structures of a) Multipolar Interneurons
b) Motor neurons and c) Sensory neurons

2.1.2.2. Motor Neurons

The primary function of motor neurons is to get information from other neurons and convey that information to your glands, muscles and organs. For instance, when you incidentally touched the hot plate, sensory neurons passed the information about plate being hot to CNS, now CNS will process that information and figure out that touching a hot plate can be injurious, so it will transmit this info to the motor neurons with the help of interneurons and in response you will avoid touching the hot plate. (Swift, A. 2015)

2.1.2.3. Interneurons

Interneurons can only be found in CNS, and they are used to pass on sensed or processed information to either CNS or PNS. Let's continue with the same example, so when you touched hot plate, sensory neurons will carry the info to interneurons in

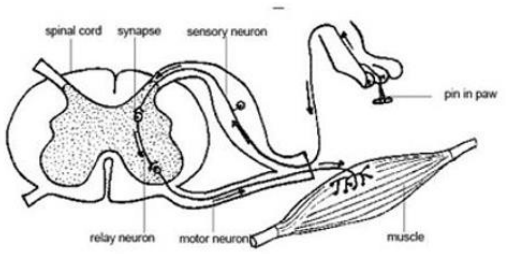

Figure 2-3 Interneuron Structure

CNS, after processing some other interneurons will pass the processed information to motor neurons, as a result you avoid touching the hot plate. On the same time, other interneurons would transmit the signal up the spinal cord to neurons in the brain, where it would be perceived as pain. (Swift, A. 2015)

2.2. Structure of a Single Neuron

From the brief study of the classes of neurons, when can figure out the following basic functions of a neuron.

- Reception of the signals
- Integration of incoming signals to determine weather information should pass or halt
- Transmit the signals to target cells, glands, tissues or organs

We can further study these functions in the anatomy of a neuron. (Reece, J. B. et al., 2011)

2.2.1. Anatomy of a neuron.

As other cells present in the human body, neurons also have a physique called the soma. The core of the neuron can be found in the soma. Neurons require processing a considerable measure of proteins and neuronal proteins are synthesized in the soma.

2.2.1.1. Nerve Process (Dendrites and Axon)

Nerves can be classified into Dendrites and Axon these nerves transmit and receive signals from brain, spinal cord and different organs with the help of nerve impulses. Dendrites carry impulses toward the nerve cell body, and axons carry impulses away

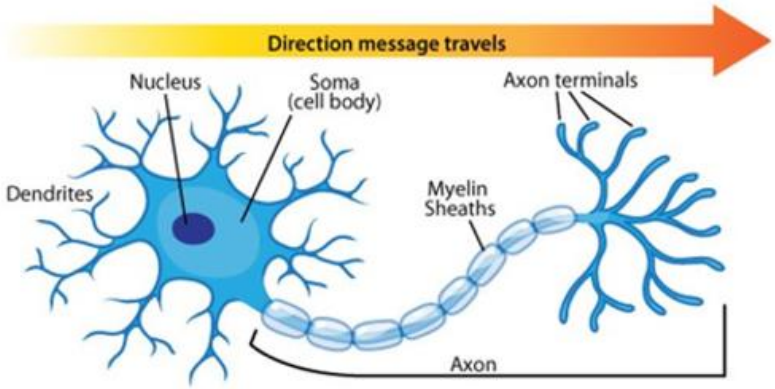

Figure 2-4: Cell Body

from the cell body. Therefore, we can say that dendrites are receiving ends of the cell

body and Axons are receiving ends of the cell body. As a final step, the signal leaves through the synapse to be passed along to the next nerve cell. Like most other cells in the body, neurons also have a nucleus, which holds the cell's DNA. (Sadava, D. E. et al., 2009)

2.2.1.2. The Synapse

Neurons have specific projections called dendrites and axons. As we already know that dendrites bring information to the cell body and axons take information away from the cell body. (Sadava, D. E. et al., 2009)

Information from one neuron flows to another neuron across a synapse. The synapse is a small gap separating neurons. The synapse consists of:

- a presynaptic ending that contains neurotransmitters, mitochondria and other cell organelles
- a postsynaptic ending that contains receptor sites for neurotransmitters
- a synaptic cleft or space between the presynaptic and postsynaptic endings

Now when we have a basic understanding of how a biological neuron works, let us jump to a mathematical model of a neuron, also called a **perceptron**.

2.3. Mathematical Model of a single neuron (Perceptron)

A Perceptron can be regarded as the most basic unit of a neural network. A parametric system have inputs, a network or a control block that processes the parameters, and off course the outputs. Therefore, we can see a single perceptron as a parametric system and study its model.

Figure 1-5 shows how to conduct a learning process for a parametric system. (*R. Rojas*: Neural Networks, Springer-Verlag, Berlin, 1996)

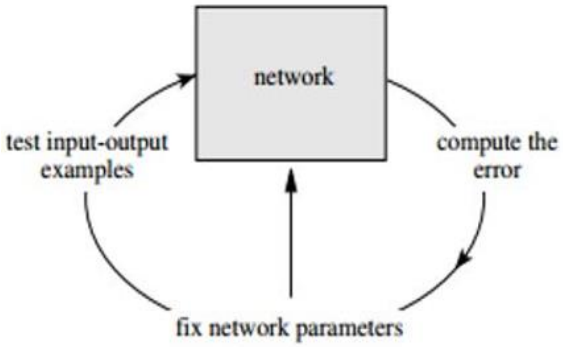

Figure 2-5 Learning Process of parametric system

Perceptron are the simplest data structures for the study of Neural Networks. We can perceive a perceptron as a node of an interconnected network. The links between the nodes not only show the relationship between the nodes but also transmit data and information, called a signal or impulse. The perceptron is a simple model of a neuron (nerve cell).s

A perceptron model have *n* inputs and an output. The inputs are "carried" within the perceptron by weighted connections (that emulates the role of synapses), as shown in figure 2-6. Thus, the perceptron receives *n* inputs, i.e. any input is multiplied by the weight over

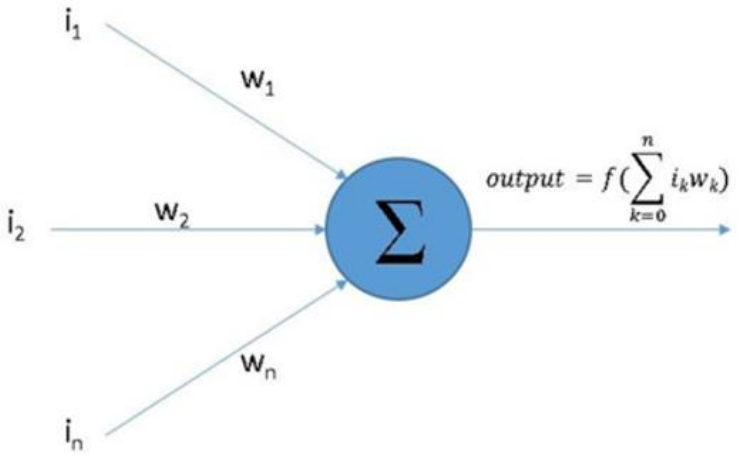

Figure 2-6: A perceptron model with *n* inputs i.e. $i_1,\ldots,i_n$, weights weights $w_1,\ldots,w_n$ over the connection, and activation function *f*

the connection that transports it. Then, the perceptron sums all the weighted inputs and applies an output. The input values are logical i.e. true or false, 1 or 0, but they can be any real number as well. The output of the perceptron, however, is always logical (1 or 0). When the output is true, the perceptron is said to be firing.

One very common activation function is the threshold function. All of the weights attached to inputs are simply multiplied by the input value and then added with other weighted inputs to get the final sum, which finally is given to activation function to see if it greater than or less than threshold value. Threshold is the most important components of the perceptron. It decides on the bases of a given function that weather perceptron can fire or not. Therefore, it fires whenever the following equation is true, where w_j and i_j represent the weights and the inputs respectively for j=1,…,n and *t* is the threshold value.

$$I_1W_1 + I_2W_2 + \ldots + I_nW_n = t$$

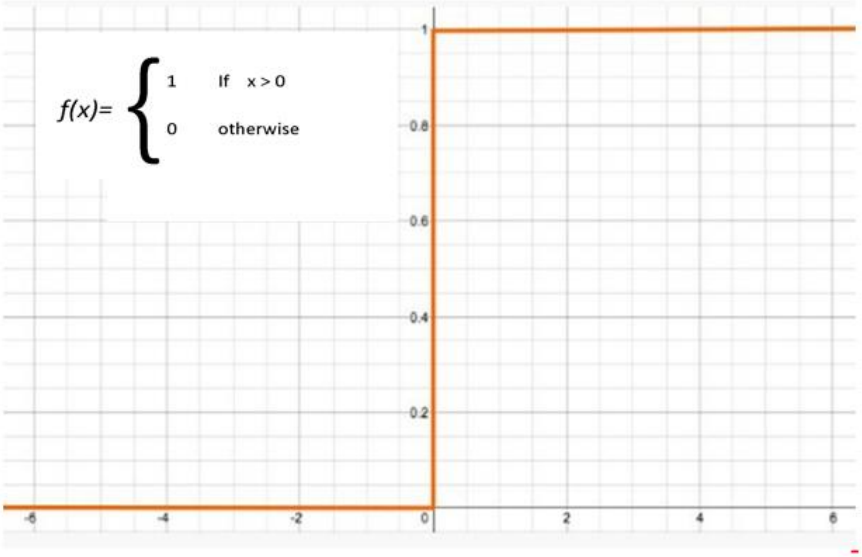

Figure 2-7: Graphical Representation of threshold Function for t=0

Sometimes the perceptron involves another parameter called bias. A bias value allows you to shift the activation function to the left or right, which may be critical for successful learning.

Let us now take a deep look on how a perceptron can solve a simple problem. We will try to construct simple networks that can solve NOT, AND and OR problems, it is than possible to easily solve any logical operation using these three operations.

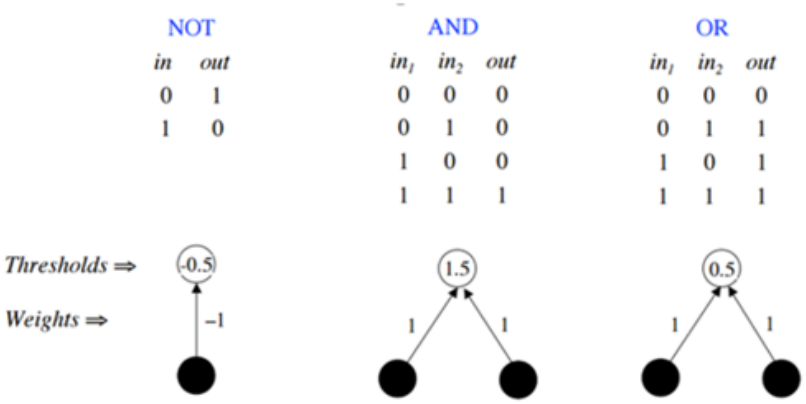

Figure 2-8: Fundamental Logic gates implementation using NNs

In figure 2-8, we can see the implementation of the basic logic gates, the only important thing to notice in the figure is the selection of weights and threshold value. For NOT gate, we know the output should be exactly inverted; therefore, weight equals -1 is multiplied with the input. Moreover, threshold must be set in order to get the inverted output. We have set -0.5 as threshold, so when input is 0, the weighted input is 0 as well, which is less greater than -0.5, so the output must be1 and vice versa. In the same way we can study OR and AND gates as well. (*John A. Bullinaria*, 2015)

Moreover the perceptron is a model too simple to solve complex problems (for instance it cannot solve the XOR problem),but if we consider multiple perceptrons working together, we can address very complex problems. Let us see how to model a NN in the following topic.

2.4. Artificial Neural Networks

From previous studies we can sum up that, the network of multiple perceptron can be called an Artificial Neural Network or ANN. Single perceptron is not capable of doing miracles, it can solve very basic problems but when then complexity increases, we must introduce more perceptron. Therefore, we use a network of perceptron to solve bigger problems, it the same as the brain does but as we are dealing in mathematics, so, we must model it. However, before starting to model a simple ANN, we must talk about the ANN topologies

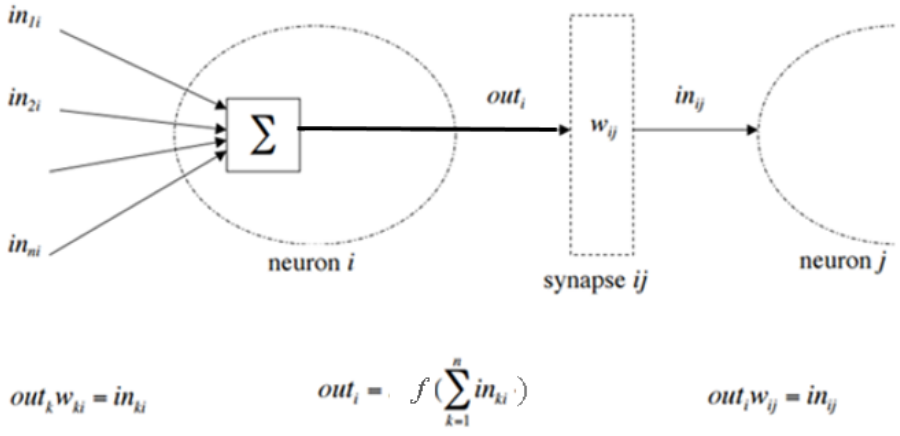

Figure 2-9: ANN structure: Two neurons (named i and j) connected by the synapse *ij* and weight w_{ij}. The activation function for neuron *i* is defined by *f*

and structures.

Usually we are going to have multiple neurons with different indices (k, i or j) and the activation flows between them via synapses with strengths w_{ki} and w_{ij}.

We need to take care of all the activation and weights without introducing any complication. Let us address some of these complications.

- Every neuron have some inputs and outputs. However, output of one neuron can be input of another.
- Usually, the networks are built up on layers of neurons, so we number the neurons separately in each layer, but we have to distinguish the weights and activations for different layers.
- We will found different labels in different books and references for inputs, outputs, activations and weights. Like in figure 2-8, inputs are labeled as $in_{1i,}$ in_{2i} and output $out_{i.}$

2.4.1. ANN Topologies and Structures

According to activation flows, we can classify ANNs into following structures:

- **Single Layer feedforward ANNs:** they have just one input layer and one output layer; there is no feedback connection as well.
- **Multi-Layer feedforward ANNs:** They have one input layer, one output layer and one hidden layer for processing units. There is no feedback connection, and the hidden layer is always between input and output layer.

- **Recurrent ANNs:** A recurrent ANN has at least one feedback connection; it can have hidden units but not necessarily.

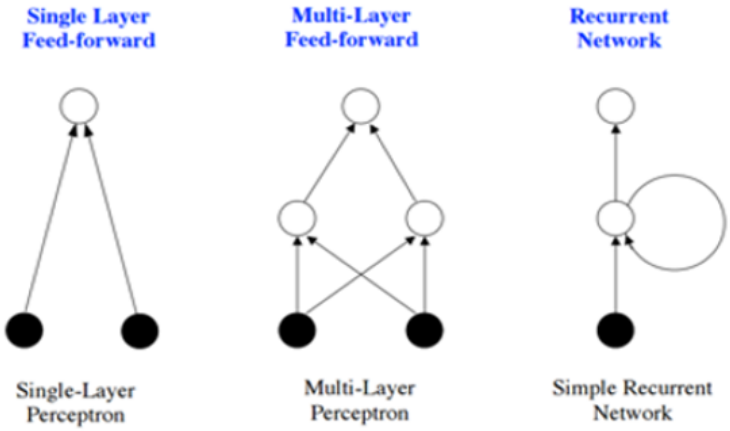

Figure 2-10: ANN structures and topologies w.r.t activation flow

In this thesis we mainly focus on multi layered feed forward networks. A milti layered feed forward neural network consists of three layers of neurons i.e. Input layer, Hidden layer and output layer. The neuron belonging to same layer are not connected with each other but they are connected with all the neurons that belong to next or previous layer. The neurons in input layer are just there to transport the inputs into NN. That is why they do not carry any weights. (*John A. Bullinaria*, 2015)

Figure 2-11 can give a better understanding of what is said above.

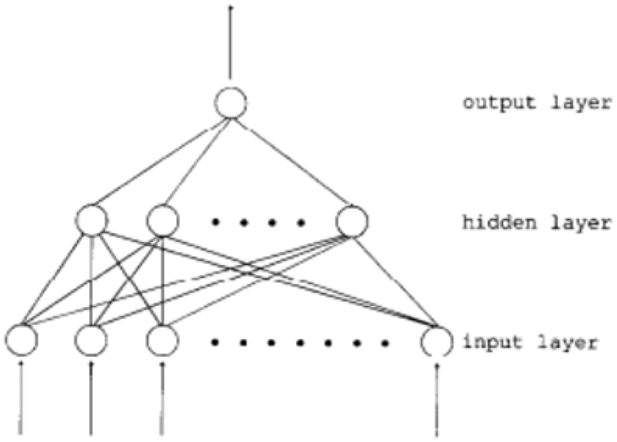

Figure 2-11: A multilayered feed forward NN with three layers

The only reason for adding this information is that when we start building out neural network, we must know, what to avoid and what to adopt. Now, we discuss an example of an ANN, which solves a complex problem more complex than NOT, AND, OR gates.

2.5. ANN and the XOR Problem

Two input Boolean operations are among the building blocks of understanding the learning process of neural computation. Only two of them show a strong level of complexity i.e. XOR and inverted XOR. (*Richard Bland, 1998*)

In this section, we will see how to solve a classic XOR problem using ANN. I mentioned the XOR problem as a classic because it has its history with neural computing. This is the

first problem that scientist faced when working on a single perceptron. Therefore, XOR problem opened the door for ANN and multiple layer neural computing.

i_1	i_2	output
0	0	0
0	1	1
1	0	1
1	1	0

Table 2-1: Truth table for XOR Problem

In the previous section, we saw that we could solve many simple problems with a single perceptron. However, for performing XOR operation, a single perceptron was not enough, therefore, we are obliged to opt for multiple perceptron. Let us now analyze the problem. We have a two binary input problem get the output in a particular way i.e. if we have same inputs the output should be zero and if we have different input, output should be one.

We can use a multi-layer feedforward NN to solve this problem. In figure 2-11, we have a complete solved XOR problem with chosen weights, thresholds and the activation function as well. As we can see in the figure above, 0, 1 in gray, 2, 3 in blue and 4 in green, are the labels assigned to the neurons. All the 1 and -1 are the weights over the connection. Moreover, the weights and the threshold of neuron 2 are set in a way that it performs inverted i_1 + inverted i_2 operation. On the other hand neuron 3 is doing an OR operation. Lastly, neuron 4 is doing an AND operation. (*Richard Bland, 1998*)

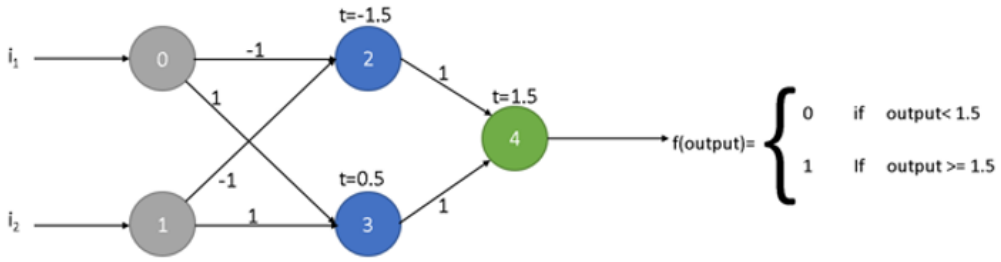

Figure 2-12: XOR problem using Neural Network

- $i_1 = 0$, $i_2 = 0$: According to our network, when both inputs are zero, neuron 2 will give 1, otherwise it will be zero. On the other hand, neuron 3 will give out 0. Neuron 4 is performing an AND operation. Therefore, now we have 2 inputs 1 and 0, so output will be zero.
- $i_1 = 1$, $i_2 = 1$: Now, neuron 2 gives out 0 and neuron 3 gives out 1, the AND operation for these inputs will be zero.
- $i_1 = 0$, $i_2 = 1$: if any input is zero, neuron 2 produces output equal to 1 and neuron 3 will give out 1 if any input is 1, so we have AND operation of true inputs that is 1.
- $i_1 = 1$, $i_2 = 0$: if any input is zero, neuron 2 produces output equal to 1 and neuron 3 will give out 1 if any input is 1, so we have AND operation of true inputs that is 1.

	1.5		
	1	1	
-1.5		0.5	
-1	-1	1	1

Table 2-2 Tabular Representation of figure 2-12

2.6. Conclusion

In this chapter, we understood the biological and mathematical perspectives of neural networks. We came up with a model of a single perceptron, then we discussed about how to implement simple neural networks with the help of a single neuron. Finally, we jumped to the more complex XOR problem. Now in the next chapters, we will use this knowledge to understand the required algorithms and to develop our neural network, which will perform character and formula recognition.

Chapter 3: Training Algorithm & Artificial Neural networks

3.1. An Overview on Training Approaches

As we have seen in the previous chapter, the behavior of the ANN is mainly determined by the weights associated to the connection. If we have to approach a complex problem, surely we need a complex ANN (with many layers and neurons). Thus, it is impossible to find "a prior" the correct values of the weights, so that the ANN performs as we desire. Therefore, it is useful to randomly initiate these weights and then adjust their value by means of a training algorithm. Training a neural network can be approached in two ways: in fact, training can be Supervised or Unsupervised. Supervised training requires both training input and the corresponding desired output to be given as data of the training algorithm. In unsupervised training instead, we usually give only set of inputs to ANN, which then elaborates patterns of "knowledge" from data. (*S. B.Maind. et al., 2014)*

3.1.1. Supervised Training

For what concerns supervised training, we provide the network with both the training inputs and the corresponding desired outputs. Then our network processes and try to establish a comparison between the actual outputs and the desired outputs. As previously said, we initialize the weights randomly, so we have to find the correct weights for each connection between adjacent perceptron to decrease an error measure so that the final error (nearly) reaches zero (*D. Anderson, et. al., 1992*). The input data set is called a training set. Therefore, we process same training set for many iterations to get the desired outputs. The

most common among the supervised algorithms until date is the back-propagation algorithm and its modified forms. We will discuss about it later in this chapter.

3.1.2. Unsupervised or Adaptive Training

In this form of training algorithm, the network is provided with just the inputs. Therefore, the network must itself decide what features to use to cluster the input data. This process is referred to as adaption (*D. Anderson, et. al., 1992*). Therefore, we can say that it is more of a probability and identification problem in which we use different identification tools for machine learning. The development of these kinds of algorithm that enable a machine to automatically process text and language has always been a great challenge in Artificial intelligence, Expert systems and Machine learning.

ANNs for unsupervised training can be generally used to take better representations of the inputs, for example, provided a test of text documents an ANN can map that document to a real valued vector in such a way that resulting vector is similar like the documents in its content. Therefore, by using clustering techniques we can use our ANN with unsupervised learning. We will not discuss this topic in detail, as we are more focused in supervised learning (*D. Anderson, et. al., 1992*).

3.2. Network Selection

Based on the type of application, we choose a befitting algorithm for training our network. This is a crucial part as we should decide what kind of algorithm we have to use before designing out neural network and after deciding we carry forward everything accordingly.

Some of the design considerations include determining the number of input and output nodes to be used, the number of hidden layers in the network and the number of hidden nodes used in each hidden layer. The number of input nodes is typically taken to be the same as the size of state variables. The number of output nodes is typically the number that identifies the general category of the state of the system. Each node constitutes a processing element and it is connected through a set of weighted links to other elements. In the past, there was a general practice of increasing the number of hidden layers, to improve training performance.(*Imran Shafi et. al*) Keeping the number of layers at three and adjusting the number of processing elements in the hidden layer, can achieve the same goal. A trial-and-error approach is usually used to determine the number of hidden layer processing elements, starting with a low number of hidden units and increasing this number as learning problems occur. Even though choosing these parameters is still a trial-and-error process, there are some guidelines that can be used, (i.e., testing the network's performance). It is a common practice to choose a set of training data and a set of Validation data that are statistically significant and representative of the system under consideration. The training data set is used to train the ANN, while the validation data is used to test the network performance, after the training phase finishes (*Magali R. G. et al, 2003*)

Most applications of neural networks fall into the following five categories:-

- **Prediction**
- **Classification**
- **Data association**
- **Data conceptualization**

- **Data filtering**

Network Type	Network Approach	Use of the Network
Prediction	Back-propagation - Delta Bar Delta - Extended delta bar delta - Directed random search - Higher order Neural Networks - Self Organizing Map into Backpropagation	Use input values to predict some output (e.g. pick the best stocks in the stock market, predict the weather, identify people with cancer risks and optical character recognition)
Classification	Learning vector quantization Counter-propagation - Probabilistic neural network	Use input values to determine the classification (e.g. is the input the letter A? is the blob of video data a plane and what kind of plane is it?)
Data association	Hopfield - Boltzmann Machine - Hamming network Bidirectional associative memory -Spatio-temporal pattern recognition	Like classification but it also recognizes data that contains errors (e.g. not only identify the characters that were scanned but also identify when the scanner wasn't working properly)
Data conceptualization	Adaptive resonance Network Self organizing map	Analyze the inputs so that grouping relationships can be inferred (e.g. extract from a data base the names of those most likely to buy a particular product)
Data filtering	Recirculation	Smooth an input signal (e.g. take the noise out of a telephone signal)

Table 3-1: Network selection table

3.3. Back Propagation Algorithm

This thesis is dealing with a neural network that is designed using back propagation (BP) algorithm, therefore, we will discuss more about that and keep other approaches aside.

In this section, we consider a fully connected network trained using BP algorithm with pattern wise approach. A single hidden layer network with an enough hidden neurons is sufficient for approximating the input-output relationship. Hence, in our analysis, we consider the network with three layers having *N(k)* neurons in each k-th layer. The different phases in the learning algorithm are discussed in this section. (S*uresh et al.2005*)

The BP algorithm is a supervised learning algorithm, and is used to find suitable weights, such that for a given input pattern, the network output should match with the target output. The algorithm is divided into three phases, namely, forward phase, error BP phase, and weight update phase.

Let us consider a neural network with *L* layers. Let *N(i)* be the number of neurons in the layer i, for i = 1,…,*L* and $\underline{w_{ij}}^{(k)}$ be the weight of the connection between the i-th neuron in the layer k and the j-th neuron in the layer k - 1. An ANN is trained over a set of inputs so that it provides a fixed output for a given training input. Let us denote X the set of training inputs. An element $\underline{x} \in X$ is a vector, Let $\underline{a}_i^{(k,x)}$ be the output of the i-th neuron in layer *k* when an input $\underline{x}$ is processed by the ANN and $\underline{y}^{(x)}$ is the desired output of ANN when $\underline{x}$ is processed. The goal is to determine the values of the weights so that, $\underline{a}^{(L,x)} != \underline{y}^{(x)}$. In the next section, we will see notations and the working of BP algorithm.(Giuseppe *Airo Farulla, et al.)*

3.3.1. Notations

L is the number of layers of the N.N.

N(k) is the number of neurons in k-th layer

X is the set of all training inputs; where an element $\underline{\boldsymbol{x}}$ in *X* is a vector whose length is equal to *N(1),* i.e. $|\underline{\boldsymbol{x}}| = N(1)$

Y is the set of desired outputs; where an element $\underline{\boldsymbol{y}}$ in *Y* is a vector whose length is equal to *N(L),* i.e. $|\underline{\boldsymbol{y}}| = N(L)$

$\underline{\boldsymbol{y}}^{(x)}$ is the desired output when the N.N. processes the training input $\underline{\boldsymbol{x}}$

$\underline{\boldsymbol{w_{ij}}}^{(k)}$ is the weight of the connection between neuron *i* in layer *k* and neuron j in layer *k*-1

$\boldsymbol{w}^{(k)}$ is the matrix with the dimension *N(k)* x *N(k-1)*; its entries are all the weights of the connections between (*k*-1)-th and *k*-th layer

$\boldsymbol{f}$ is the activation function

$\underline{\boldsymbol{z}}^{(1,\underline{x})} := \underline{\boldsymbol{x}}^{T}$ i.e. is the training input written as a column vector

$\underline{\boldsymbol{a}}_1^{(1,\underline{x})} := \underline{\boldsymbol{z}}^{(1,\,x)}$, $a_1^{(1,\underline{x})}$ is the output of the first neuron in first layer when you gave $\underline{\boldsymbol{x}}$ as input of the N.N.

For all $\boldsymbol{k} = \mathbf{2},\ldots,L$

$\underline{\boldsymbol{z}}^{(k,\underline{x})} := \boldsymbol{w}^{(k)} * \underline{\boldsymbol{a}}^{(k-1,\underline{x})}$

$\underline{\boldsymbol{a}}^{(k,\underline{x})} := \boldsymbol{f}(\underline{\boldsymbol{z}}^{(k,\underline{x})})$

$\underline{\boldsymbol{a}}^{(L,\underline{x})}$ is the final output of the N.N. when you give $\underline{\boldsymbol{x}}$ as input of the N.N.

$\underline{\boldsymbol{a}}_j^{(k-1,\underline{x})}$ is the output of the ***j-th*** neuron in the ***(k-1)-th*** layer when you gave $\underline{x}$ as the input

3.3.2. The BP Algorithm

We would like to find all the weights, such that, we have $\underline{\boldsymbol{a}}^{(L,\underline{x})} = \underline{\boldsymbol{y}}^{(\underline{x})}$ for all the training inputs. Let us see the steps of the algorithm.

$\underline{\boldsymbol{w_{ij}}}^{(k)}$ are randomly initialized in the interval (-h/2,h/2)

while $\exists\ \underline{\boldsymbol{x}} \in X$: $\underline{\boldsymbol{a}}^{(L,\underline{x})} != \underline{\boldsymbol{y}}^{(\underline{x})}$

for $\underline{\boldsymbol{x}} \in X$,

do

$\underline{\boldsymbol{z}}^{(1,\underline{x})} = \underline{\boldsymbol{a}}^{(1,\underline{x})} = \underline{\boldsymbol{x}}$

for k=2,…..,L

do

$\underline{\boldsymbol{z}}^{(k,\underline{x})} = \boldsymbol{w}^{(k)} * \underline{\boldsymbol{a}}^{(k-1,\underline{x})}$

$\underline{\boldsymbol{a}}^{(k,\underline{x})} = f\,(\underline{\boldsymbol{z}}^{(k,\underline{x})})$

end for

$\underline{\boldsymbol{d}}^{(L,\underline{x})} = (\underline{\boldsymbol{a}}^{(L,\underline{x})} - \underline{\boldsymbol{y}}^{(\underline{x})}) \odot f\,(\underline{\boldsymbol{z}}^{(L,\underline{x})})$, where $\odot$ is the component wise product

for $k = L\text{-}1$,….,2

do

$\underline{\boldsymbol{d}}^{(k,\underline{x})} = [(\boldsymbol{w}^{(k+1)})^{\mathrm{T}} * (\underline{\boldsymbol{d}}^{(k+1,\underline{x})})] \odot f\,(\underline{\boldsymbol{z}}^{(k,\underline{x})})$

end for

end for

for $k=L,\ldots..,2$

do

$$\boldsymbol{w}^{(k)} = \boldsymbol{w}^{(k)} - \eta/\,|\underline{\boldsymbol{x}}| \sum_{\underline{x} \in X} (\underline{\boldsymbol{d}}^{(k,\underline{x})} * (\underline{\boldsymbol{a}}^{(k-1,\underline{x})})^{\mathrm{T}})$$

where η is a parameter called learning rate.

end for

end while

We can observe that in BP algorithm, the quantity, $\underline{\boldsymbol{d}}^{(L,\underline{x})} = (\underline{\boldsymbol{a}}^{(L,\underline{x})} - \underline{\boldsymbol{y}}^{(\underline{x})}) \odot f(\underline{\boldsymbol{z}}^{(L,\underline{x})})$ is essentially the error performed by our ANN when it processes the input $\underline{\boldsymbol{x}}$ for the current values of the weights i.e. it is the measure of how much the output of the ANN is far from the desired output. The following lines in the algorithm "back propagate" such an error in the previous layer. Later, we briefly see the origin of the equation that updates the values of the weights in the BP algorithm.

Now for the sake of simplicity, we will consider $k=L$, and we prove that updating the values of the weights using following equation decreases the error of our ANN

$$\boldsymbol{w}^{(L)new} = \boldsymbol{w}^{(L)} - \eta/\,|\underline{\boldsymbol{x}}| \sum_{\underline{x} \in X} [(\underline{\boldsymbol{a}}^{(L,\underline{x})} - \underline{\boldsymbol{y}}^{(\underline{x})})\,(\underline{\boldsymbol{a}}^{(k-1,\underline{x})})^{\mathrm{T}} \odot f(\underline{\boldsymbol{z}}^{(L,\underline{x})})]$$

Given the set of initial weights that are randomly sampled, we want to change them in order to get $\underline{\boldsymbol{a}}^{(L,\underline{x})} = \underline{\boldsymbol{y}}^{(\underline{x})}$ for all the training inputs $\underline{\boldsymbol{x}}$. Thus, we would like to minimize following cost function:

$$\boldsymbol{C}(w) = 1/2n \sum_{\underline{x} \in X} \| \underline{\boldsymbol{y}}^{(\underline{x})} - \underline{\boldsymbol{a}}^{(L,\underline{x})} \|^2 \qquad (1)$$

Where n = $|\underline{\boldsymbol{x}}|$ and *w* shows that the cost function depends on the weights that we currently have assigned.

We use a quadratic cost function because it easily shows how to make small changes in the weights to get an improvement in the cost function. The function $\boldsymbol{C}$ is essentially a measure of the error performed by the neural network with the current weights. So, we want to find a way to reduce such an error gradually.

In general if, we have a function *g* with *n* variables i.e. $g = g(x_{1\ldots}\, x_n)$ it is well known that,

$$\Delta g = g(x_1{}^{new}{}_{\ldots}\, x_n{}^{new}) - g(x_{1\ldots}\, x_n) \quad \cong$$

$$\cong \quad \partial g/\partial x_1\,(x_1{}^{new} - x_1) + \partial g/\partial x_2\,(x_2{}^{new} - x_2) + \ldots.. + \partial g/\partial x_n\,(x_n{}^{new} - x_n) \qquad (2)$$

The gradient vector is defined as follows:

$$\underline{\nabla} g := (\partial g/\partial x_{1\ldots}\, \partial g/\partial x_n)^T$$

Where $\underline{\nabla} g$ is a column vector, Hence we can write eq(2) as the dot product of $\underline{\nabla}\, g$ and $\Delta\, \underline{\boldsymbol{x}}$ i.e.

$$\Delta g \cong \underline{\nabla}\, g \bullet \Delta\, \underline{\boldsymbol{x}}$$

Where, $\Delta\, \underline{\boldsymbol{x}} = (x_1{}^{new} - x_1,\, x_2{}^{new} - x_{2\ldots}\, x_n{}^{new} - x_n)$ and is a row vector.

Thus we easily (and quite trivially) choose a value for $\Delta\, \underline{\boldsymbol{x}}$ so that $\Delta g < 0$, in this way we have that $g(x_1{}^{new}{}_{\ldots}\, x_n{}^{new}) < g(x_{1\ldots}\, x_n)$. If we think to the cost function $\boldsymbol{C}$, this means that we have reduced the error of our ANN, so we can choose,

$$\Delta \underline{\boldsymbol{x}} = - \eta \underline{\nabla} g \; ; \qquad (3)$$

where η is a positive constant named the learning rate. From eq(3) we have

$$\Delta g = - \eta \| \underline{\nabla} g \|^2 < 0$$

If we define the vector $\underline{\boldsymbol{x}}^{new} = (\boldsymbol{x}_1^{new}, \ldots, \boldsymbol{x}_n^{new})$ and $\underline{\boldsymbol{x}} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$

$$x^{new} = x - \eta * \underline{\nabla} g$$

If we apply this formula to eq(1), we get

$$w^{new} = w - \eta /2n * 2 \sum_{\underline{x} \in X} \| \underline{\boldsymbol{y}}^{(\underline{x})} - \underline{\boldsymbol{a}}^{(L,\underline{x})} \| * (\underline{\boldsymbol{a}}^{(L-1,\underline{x})})^{\mathrm{T}} \odot f(\underline{\boldsymbol{z}}^{(L,\underline{x})}),$$

Remembering that $\underline{\boldsymbol{a}}^{(L-1,\underline{x})} = f(\underline{\boldsymbol{z}}^{(L,\underline{x})}) = \boldsymbol{w}^{(L)} \; \underline{\boldsymbol{a}}^{(L-1,\underline{x})}$

3.4. Overview on identification of BP Algorithm parameters

The capacity of an ANN to generalize well on hidden data depends on a variety of factors, most important of which is the matching of the network complexity with the degree of freedom or information that is essential in the training data. Matching of this information with complexity is critical as it allows ANNs to generalize properly to a set of possible different inputs. A measure of the complexity of a structure is its number of free or adjustable parameters, which for a feedforward neural network is the number of synaptic weights. Clearly, the ability of a feedforward neural network in learning the samples of the training set is proportional to its complexity. (*E. J. Teoh, et al.)*

The BP algorithm is the most used approach when it comes to using ANNs for optical character recognition because of its ability to process huge data sets (*Sheetal et. al*), but these approaches have some drawbacks as well, and in this section, we will discuss those drawbacks regarding some parameters. Eventually, the goal of this thesis is to find out the optimal values of these parameters for optical character recognition with multiple data sets.

3.4.1. Learning Rate (η)

Training a neural network using an algorithm such as described in the section 3.3 usually requires a lot of time on large, composite problems. Such algorithms typically have a learning rate parameter that defines how much the weights can change in response to an experimental error on the training set. The choice of this learning rate can have a significant impact on the generalization accuracy as well as on the complexity and thus duration of training. Almost anyone who has used such training algorithms has been faced with the problem of choosing the learning rate, but there is rarely much guidance on what value to use, since the best value to use depends on peculiarities of the single task (*D. Randall Wilson, et al.).*

When using a gradient descent-learning algorithm, the error gradient (or an approximation thereof) is calculated at the current point in weight space, and the weights are changed in the opposite direction of this gradient to minimize the error described by means of an error function. However, although the gradient may indicate what direction the weights should be moved, it does not specify how far the weights may safely be moved in that direction before the error quits decreasing and starts increasing again. Therefore, a learning rate that is too large often moves too far in the "correct" direction, resulting in overshooting a valley

or minimum in the error surface, thus hurting accuracy. Because of this effect, when using a learning rate too big the training will cause longer training; this is so, because the ANN is continually overshooting its objective and tends to "unlearn" what it has already learned, thus requiring expensive backtracking or causing unproductive oscillations. This instability often causes poor generalization accuracy as well, since the weights can never settle down enough to move all the way into a minimum before bouncing back out again. (*Wilson, D. R. et.al*)

Once the learning rate is small enough to avoid such overcorrections, it can proceed in a relatively smooth path through the error landscape, finally settling in a minimum. Reducing the learning rate further can make this path smoother, and doing so can significantly improve generalization accuracy. However, there comes a point at which reducing the learning rate any more simply wastes time, resulting in taking many more steps than necessary to take the same path to the same minimum.(*Jacobs, R. A, et al.)*

3.4.2. Number of neurons in hidden layer (N)

Optimizing the number of hidden layer neurons for building an AFNN to solve the problem remains one of the mysterious tasks in this research area. Setting too few hidden units causes high training errors and high generalization errors due to under-fitting, while too many hidden units results in low training errors but still high generalization errors due to over fitting (*Shuxiang Xu, et al. 2008*). The only one advice given for setting the number of neurons in the hidden layer is to take a number between the numbers of neuron in last and first layer, see (*Rojas, Neural Networks, A Systematic Introduction, 1996*). However,

this consideration is not supported by theoretical studies and could not be effective for approaching some problems.

3.4.3. Weight Initialization in BP Algorithm

Weight initialization has been widely recognized as one of the most effective approaches in speeding up the training of neural network. (Jim *Y.F. Yam, et. al*) Initializing of weights effects the performance of neural networks largely. For that purpose, researchers around the world have proposed many methods of initializing the weights as ANNs are used in solving very complex problems, so every problem is heavily effected by initialization of weights. The most used method until date is random weight initialization, even if many techniques have been recently developed. In spite of this, random initialization is still the most common method due to its simplicity. In the present norm, we use random initialization. (*Nadir Murru, et. al*)

3.4.4. Interval in which weights can be sampled

This is another parameter that is to be optimized in our case. This is the interval between h and –h in which we sample our weights. Actually, our sampling interval is set to –h/2, h/2, but the real problem that we face is determining the optimal value of h, in which our algorithm gives the best fit of the desired output.

3.5. Conclusion

In this chapter, we discussed about the training algorithms that we use to train ANNs. Furthermore, we classified those algorithms into supervised and unsupervised algorithms. Then we focused on our point of interest, i.e., Back Propagation algorithm. Then we gave

a brief proof of the BP algorithm with the notations that we use. At last, we discussed the problems that we face, when we build a neural network that is based on BP algorithm. In the next chapter, we will discuss the outcomes of our experiments and try to find the optimal values of the different parameters of our algorithm.

Chapter 4: Application to Character Recognition

In this Chapter, the Optical Character Recognition (OCR) system based on Artificial Neural Networks (ANNs) is discussed. ANNs are frequently employed to solve sample-recognition problems. One of these is, indeed, character recognition. The research work behind this thesis aims at recognizing printed characters by projecting them on different sized grids. The first step in the recognition process is the image acquisition, often followed by noise filtering, smoothing and normalization of the image. In this work, we resort to ANNs trained using the Back Propagation algorithm. In the proposed Character recognition system, each typed English letter is represented by a string of binary numbers that are used as input to a simple feature extraction system whose output, in addition to the input, are fed to an Artificial Neural Networks. Afterwards, the Feed Forward Algorithm gives insight into the enter workings of a neural network followed by the Back Propagation Algorithm which compromises Training, Error calculation, and Modifying Weights. (Prasad et al.) We will perform sensitive analysis on some parameters of BP algorithm and try to optimize them for the best performance of our ANN.

4.1. Introduction

OCR system converts the image obtained by scanning a text or a document into machine editable format. Recognition of printed characters is itself a challenging problem, especially when real world scenario (image may be noisy or degraded) is considered. In addition, variability in font types and sizes makes recognition a difficult task. A good character recognition approach must eliminate the noise after reading binary image data, smooth the image for better recognition, extract features efficiently, train the system and

classify patterns-(S. Barve). The aim of this thesis is to use ANNs to simplify the development of a character recognition application, still ensuring high quality of recognition and good performance. (Rókus, et al.). Our goal is to show the precision and speed of character recognition depending on the parameters of the implemented neural network later in this chapter; we will discuss every step of application to character recognition in detail.

4.2. Character Recognition with MatLAB

The algorithm that we are currently working on is completely designed using MatLAB. The problem definition of character recognition is not that simple, therefore, we have to define a path, and by following that path, we will achieve our desired goals. We can define the whole character recognition problem in to following major steps.

- Creating Training and Validation Data sets
- Training
- Testing

4.2.1. Creating Training and Validation Data

To train and use ANNs properly, we have created sets of training data and validation data, which contain training inputs and corresponding desired outputs. We started from .png files of all the English alphabets and we converted them into binary matrices in which each column defines a character. We used '**create_training_set_test**' function to create our training and validation data. We create training and validation data in the same way,

therefore, for the sake of simplicity; we will just focus on creating training data. Let us discuss in detail how this function works. The function can be called in MatLAB as follows,

[trainingSet, trainingOutput] = create_training_set_test(path, fontSize, nc, nr)

This function has four inputs and it returns two outputs. First input is the directory path of all the *.png files and each *.png file contains a character. The second input is font size, we have to describe the font size that we are working on, third and fourth inputs are number of columns and number of rows of the output binary matrices respectively. The outputs are training set and training output; these are two binary valued matrices, which contain our training data and corresponding desired output.

This single function does following tasks:

- **Preprocessing** – processing the data in the form needed.
- **Features extraction** – we minimize the needed data with saving only the needed information. This will give us a vector with binary values.

4.2.1.1. Preprocessing

The preprocessing consists of two steps:

- Binarization.
- Segmentation.

Our first goal is to convert that raw data into desired form. For this purpose, we use another function called '**image_preprocessing_test**'. This function can be called as follows:

[imbw,imbw_lin]= image_preprocessing_test(img,nc,nr)

It produces a binary inverted image starting from img. When nc and nr are provided, it produces also a linearized image (of size nc*nr) and memorizes it into the matrix named imgbw. At this point, we have our desired binary image in matrix form.

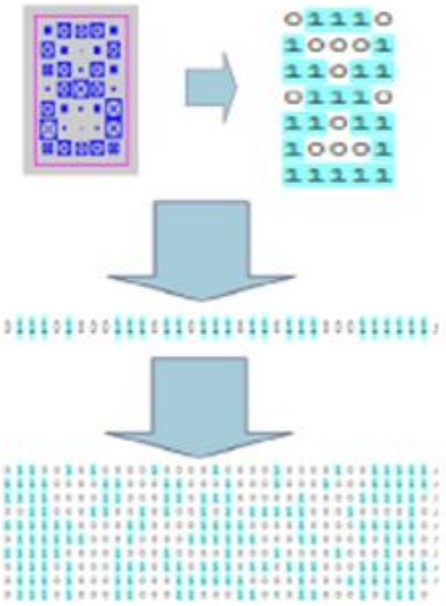

Figure 4-1: The transformation of input data into a binary matrix

The segmentation is the most important part of the preprocessing method to extract important details of every character used for the subsequent recognition task. After the segmentation, we have to decide which details are important for us. Next is the step of features extraction that we will discuss in next section.

4.2.1.2. Features Extraction

For feature extraction, we use another function inside '**image_preprocessing_test**' called **'extract_subm'**, which can be called in MatLAB as follows:

[imbw_lin, s] = extract_subm(imbw,nc,nr)

This function produces a (nc*nr) matrix by cropping zeros rows and columns and padding with zeros if needed. After this stage, we get a binary image of the characters as shown in figure 4-2.

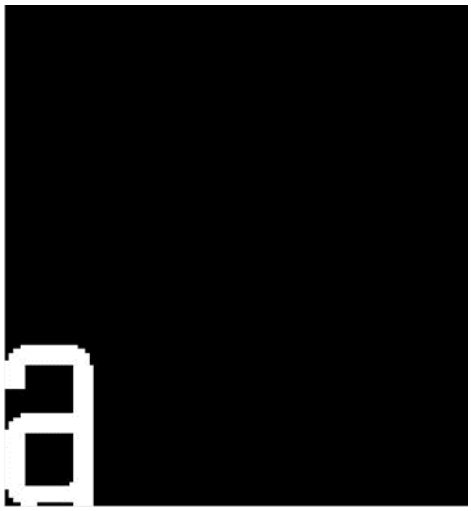

Figure 4-2: Binary image produced using create_training_set_test

After getting this binary image, which can also be represented as a binary matrix, we convert it into a column vector to use it as training data. For that purpose, we use another sub function inside extract_subm, which is called **'fix_dim'**. Which can be called in MatLAB as follows:

B = fix_dim(A,nc,nr)

It converts a matrix A of dimension nr x nc into a string of column vectors of size nc*nr. Finally, we can use that string of column vectors as our training set.

4.2.1.3. Issues in creating the training sets

Creation of the training sets is one of the most time-consuming phase when building an OCR. The first task after generating our segmented binary image is to find a way in which we can create training sets in a faster and reliable way (i.e., so that training sets do not have

noisy or mistaken data). To make this whole process faster we prepared another function that could create training sets in seconds, but it needed a reference training set. So eventually, we had to create at least one training set using the previous functions. We created multiple training sets using following specifications.

Number of fonts	106
Number of characters	26
nc	100,105,110115,120,125
nr	105,110,115,120,125,130
Font size	11

Table 4-1: Specification for creating training sets

Number of fonts	22
Number of characters	26
nc	100,105,110115,120,125
nr	105,110,115,120,125,130
Font size	11

Table 4-2: Specification for creating validation sets

We first created a training set of 26*106 characters with font size 11 for nr=105 and nc=100 and a validation set of 26*22 characters with font size 11 for nr=105 and nc=100, then we created all the other training and validation sets (for the different values of nr and nc depicted in the tables above), using another function that we designed especially for this purpose. We named that '**create_training_set_new'** which can be called in MatLAB as follows,

new_training_set = create_training_set_new' (trainingSet,nr,nc,p,q,M)

We can see that it gives a new training set whose elements are vectors of dimensions p*q starting from a training set whose elements are vectors of dimensions nr*nc.

- trainingSet is the output of create training set
- nr and nc are the dimensions of the trainingSet

- p and q are the dimensions of the new_training_set
- M is the number of columns of trainingSet.

The function simply append zeroes in the rows and columns of each vector in the previous training set. Thus, it only changes the size of the vectors from **nr*nc** to **p*q** by using zero padding. In conclusion, we created training sets containing 106*26 characters represented by vectors of length nc*nr, for the different values of nr and nc as depicted in Table 4-1. Similarly, we obtained different validation sets.

4.3. Training the ANN

This section deals with the training algorithm that we are using to train the network and the MatLAB functions associated with the algorithm.

Chapter 3 introduced the BP algorithm and we gave a brief proof of that as well, while this chapter deals with its MatLAB implementation.

The first function that we use for training is '**train_neural_network'**, the input and output arguments of this function are as follows:

[weights_12, weights_23, trainingStep, elapsedTime] = train_neural_network(starting_weights_12, starting_weights_23, trainingSet, trainingOutput, eta, maxStep)

- **starting_weights_12** is the initial matrix of weight between input and hidden layer and is an input matrix of dimensions N2xM**,** where N2 is the number of hidden neurons and M is the size of an input vector, i.e., nr*nc.

- **starting_weights_23** is the initial matrix of weights between hidden and output layer and is a matrix of dimension 26xN2, where 26 denotes the 26 alphabets whereas N2 is the number of neuron in the hidden layer.
- **trainingSet** is the training set that we created before.
- **trainingOutput** is the set of desired outputs.
- **eta** is the learning rate.
- **maxStep** is the maximum number of steps that our algorithm is allowed to take.

Initial weights are sampled between an interval (–h/2,h/2) which is also a parameter to be optimized. This function in return gives outputs as follows:

- **weights_12** is the final matrix of weights between input and hidden layer that will be used for validation.
- **weights_23** is the final matrix of weights between hidden and output layer that will be used for validation.
- **trainingStep** is the performance measure of the algorithm. i.e., is the number of steps necessary to accomplish the training.
- **elapsedTime** is the time taken by the network to train.

The **activation function** that we use for training our network is **hyperbolic tangent**-(Bekir K. et al.). Moreover, this function enforces back-propagation algorithm for training the ANN. Drawbacks of Back-propagation algorithm are well known in literature. Perhaps the best known is called “Local Minima”. This appears to be occurring because the algorithm tends to change the weights in such a manner as to cause the error to reduce, but the error may have to rise as part of a more general fall. If this is the case, the algorithm will be stuck (because it cannot go uphill) and the error will not decrease further. There are several

solutions to this problem. One is very simple and that is to reset the weights to different random numbers and try training again (this can also solve several other problems). Another solution is to add "momentum" to the weight change. This means that the weight change this iteration depends not just on the current error, but also on previous changes.

A sub function inside **'train_neural_network'** is used to define the activation function and we named it 'activation_function', which can be called in MatLAB as follows:

[y, yd] = activation_function(x, type)

This function returns the activation function y and its derivative **yd** computed on input **x**, whereas type refers to what kind of activation function we want.

In this section, section we explained the functions that we use for training, the experimental results after training will be discussed later in this Chapter.

4.4. Testing the ANN

In this phase, we test our ANN with some parameters that define the performance of the algorithm and ANN as well. The main function we use for testing our ANN is named '**test_neural_network'** and can be called in MatLAB as follows:

[testOutput, elapsedTime, pErr, nErr] = test_neural_network(weights_12, weights_23, validationSet, trainingOutput)

This function accepts input arguments as follows:

- **weights_12** is the final matrix of weights between input and hidden layer that we acquired after training and will be used for validation.
- **weights_23** is the final matrix of weights between hidden and output layer that we acquired after training and will be used for validation.

- **validationSet** is the input validation set that we created.
- **trainingOutput** is the correct outputs corresponding to each validation input that we created.

By giving these inputs, we get in return following output parameters:

- **testOutput** is the output that we compare with training output.
- **elapsedTime** is time taken to test the algorithm.
- **pErr** is the percentage error between training output and test output.
- **nErr** is number of errors occurred.

This MatLAB function gives us the two important performance measures i.e. **pErr** and **nErr**. We will focus more on **pErr**, because it gives a better understanding of error. The goal is to optimize these parameters, lower the value of these parameters better the precision of our algorithm. In the next section, we will discuss the experimental results and try to find out the optimal values of **eta, h, N2, nr** and **nc**.

4.5. Experimental Results

As we discussed prior in this chapter (see section 4.2.1.3.), we created multiple training sets with some desired specifications (see Table 4-1). In particular, we have 36 different training sets containing all the 26 English characters for 106 different fonts, and font size 11. An element of these training sets is a vector of length nr*nc. Each training set corresponds to a couple of values for nr and nc (see Table 4-1). Similarly, we have created 36 different validation sets, where the fonts involved are not the same fonts used for the training sets (see Table 4-2). We carried out multiple training and testing sessions with different values of parameters nr, nc, eta, h and N2. The

results that we got from these experiments were quite amazing. Let us discuss how we carried out those experiments and their results.

4.5.1. Optimization of nr and nc

In this section, we focus the attention only on parameters nr and nc. In fact, we want to investigate if the size of the elements in the training sets affects the performance of BP algorithm both in terms of steps for accomplishing the training and in terms of correct recognition among the characters in the validation set.

The specifications for the first experiment are as follows:

Number of fonts for training	106
Number of fonts for validation	22
Number of characters	26
nc	100,105,110115,120,125
nr	105,110,115,120,125,130
Font size	11
eta	0.8
h	0.8
N2	80
maxSteps	15000

Table 4-3: Specification for the first experiment

The task was to train the ANN over all the possible combinations of nr and nc, i.e., 36 different training sets and analyze number of steps to train the network and percentage of errors in the recognition of characters of the validation set, for the values of the parameters given in Table 4-3. Furthermore, we trained our algorithm for 10 iterations for each combination of nr and nc, i.e., for this test, we trained and tested the algorithm 360 times to extract these results. Table 4-4 summarizes experimental results after training and testing the algorithm.

Mean Training Steps and Percentage Error		
nr x nc	**No. Of Steps**	**pErr**
105x100	297.4	22.9
105x105	273.8	22.4
105x110	280.2	23.6
105x115	261.9	23.2
105x120	276.5	22.5
105x125	251.4	23.2
110x100	245.9	22.7
110x105	298.0	22.7
110x110	301.2	22.9
110x115	263.8	23.8
110x120	265.3	22.7
115x125	274.8	22.9
115x100	270.9	23.0
115x105	252.7	22.3
115x110	268.8	23.1
115x115	278.3	22.5
115x120	256.9	22.3
115x125	273.2	22.5
120x100	272.8	21.9
120x105	248.2	23.4
120x110	287.8	23.3
120x115	285.0	22.1
120x120	278.2	22.9
120x125	263.6	22.8
125x100	267.2	23.1
125x105	246.0	22.7
125x110	265.3	23.8
125x115	279.9	23.0
125x120	257.7	23.4
125x125	279.6	22.9
130x100	273.9	22.8
130x105	269.5	22.5
130x110	293.8	23.0
130x115	267.8	23.2
130x120	283.5	22.6
130x125	276.1	22.3

Table 4-4: Mean values of number of steps and pErr for eta=0.8, h=0.8 and N2=80

In Table 4-4, we can observe that the lowest value for number of steps occurs when nr=110 and nc=100, i.e., 245.9 and the second lowest is 246 for nr=125 and nc=105. Therefore, if we observe more, we can see that value of number of steps fluctuates between 245 and 300 steps, which is not so significant. This result for number of steps means that the values of nr and nc do not affect so much the results. Therefore, we can say that the values of nr and nc do not affect the result in a very much. Figure 4-3 provides a better understanding of these results. We obtained similar results using different values of eta, N2 and h.

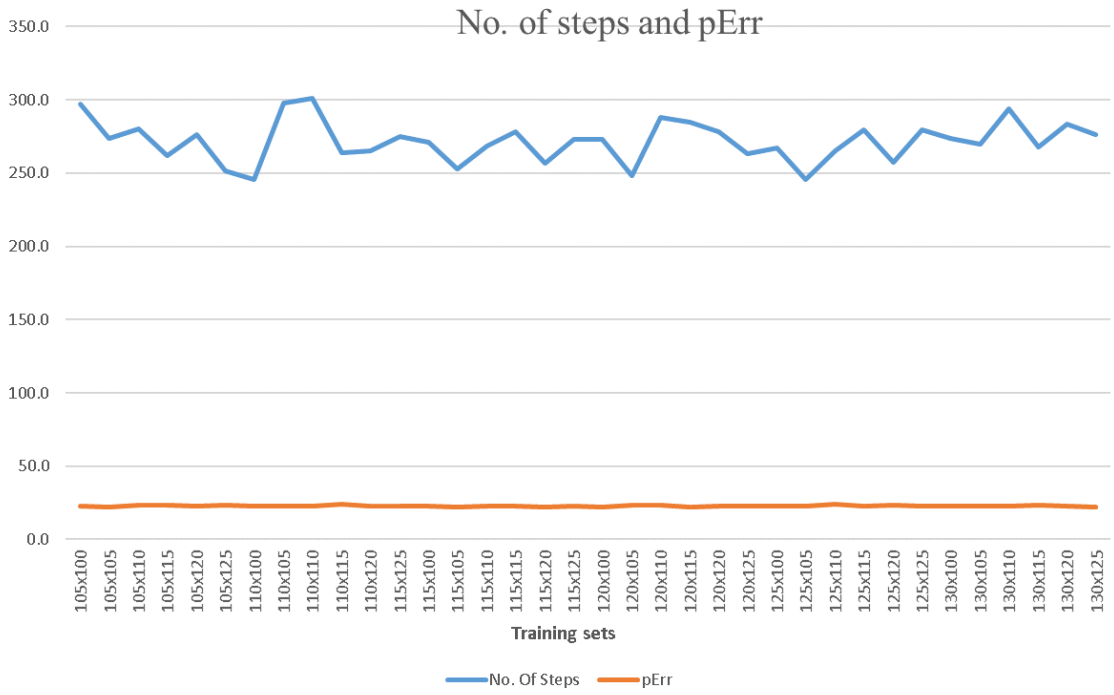

Figure 4-3: Graphical representation of Table 4-4

4.5.2. Optimization of parameters eta, N2 and h for font size 11

The specifications for the second experiment are as follows:

Number of fonts for training	106
Number of fonts for validation	22
Number of characters	26
Training sets (nr x nc)	105x100, 110x100, 120x105, 125x105
Font size	11
eta	0.4, 0.8, 1.2
h	0.4, 0.6, 0.8, 1, 1.2
N2	40, 60, 80, 100, 120
maxSteps	15000

Table 4-5: Specification for the optimization of eta, N2 and h for font size 11

From the previous test, we chose four random training sets keeping in mind the optimal values of performance measures. A complete specification of parameters is given in Table 4-5. This time we trained our ANN again for 10 iterations with different values of eta, h and N2 and got following results.

Mean values of Number of training steps for nr=105 and nc=100											
Eta	**h**		**Mean**	**Eta**	**h**		**Mean**	**Eta**	**h**		**Mean**
0.4	**0.4**	**100**	330.4	**9110**	**0.4**	**100**	213.3	**1.2**	**0.4**	**100**	525.5
		120	232.1			**120**	190.3			**120**	375.7
		40	1355.2			**40**	2704.2			**40**	1947.9
		60	560.0			**60**	649.1			**60**	1302.7
		80	359.5			**80**	288.8			**80**	490.4
	0.6	**100**	351.0		**0.6**	**100**	174.4		**0.6**	**100**	208.7
		120	268.0			**120**	132.9			**120**	208.7
		40	2218.0			**40**	1031.6			**40**	970.0
		60	736.0			**60**	430.6			**60**	384.5
		80	535.0			**80**	230.0			**80**	237.5
	0.8	**100**	414.9		**0.8**	**100**	177.7		**0.8**	**100**	208.8
		120	360.0			**120**	145.3			**120**	182.9
		40	4499.0			**40**	1329.8			**40**	913.4
		60	1059.9			**60**	469.0			**60**	380.3
		80	589.8			**80**	248.7			**80**	251.0
	1	**100**	465.6		**1**	**100**	259.0		**1**	**100**	230.8
		120	393.3			**120**	229.4			**120**	193.4
		40	6752.0			**40**	3490.2			**40**	1604.4
		60	1857.5			**60**	920.2			**60**	585.8
		80	657.0			**80**	452.3			**80**	329.6
	1.2	**100**	598.7		**1.2**	**100**	218.7		**1.2**	**100**	192.6
		120	515.8			**120**	184.1			**120**	168.8
		40	10870.0			**40**	2279.6			**40**	1079.3
		60	2231.0			**60**	745.1			**60**	439.3
		80	911.5			**80**	347.5			**80**	260.6

Table 4-6: Mean values of Number of Training Steps for nr=105 and nc=100 for font size 11

Mean values of percentage error for nr=105 and nc=100											
Eta	h		Mean	Eta	h		Mean	Eta	h		Mean
0.4	0.4	100	25.4	9110	0.4	100	12.9	1.2	0.4	100	9.6
		120	25.5			120	12.2			120	8.9
		40	20.9			40	12.9			40	11.0
		60	22.7			60	14.4			60	10.0
		80	23.7			80	13.8			80	9.7
	0.6	100	26.7		0.6	100	19.4		0.6	100	13.8
		120	28.8			120	19.3			120	12.6
		40	22.6			40	20.6			40	16.2
		60	24.8			60	21.2			60	17.2
		80	25.6			80	19.9			80	14.8
	0.8	100	27.6		0.8	100	22.2		0.8	100	17.0
		120	29.1			120	20.9			120	15.9
		40	24.2			40	23.1			40	21.2
		60	25.1			60	23.2			60	20.4
		80	25.6			80	23.0			80	17.9
	1	100	26.9		1	100	21.9		1	100	19.5
		120	28.5			120	20.8			120	18.3
		40	25.1			40	25.3			40	24.5
		60	25.9			60	24.8			60	23.9
		80	27.0			80	23.3			80	22.0
	1.2	100	26.9		1.2	100	22.3		1.2	100	19.5
		120	29.0			120	20.6			120	17.7
		40	24.5			40	24.2			40	24.0
		60	25.5			60	24.2			60	22.2
		80	26.6			80	23.9			80	20.8

Table 4-7: Mean values of percentage error for nr=105 and nc=100 for font size 11

From Table: 4-6 and Table 4-7, we can extract that, for eta=0.8, h=0.4 and N2 120, we get the optimum value for number of steps, and the percentage error for these values is very low. However, for eta=1.2, h=0.4 and N2=120, we get the optimum value for percentage error but number of steps for this value are not satisfying. Therefore, we think that eta=0.8, h=0.4, and N2=120 give the best results in term of performance. Figure 4-7, show the graphical representation of Table 4-6 and table 4-7. Now we try to find out what happens when we change nr and nc. Let us test the ANN with second training set i.e. nr=110 and nc=100.

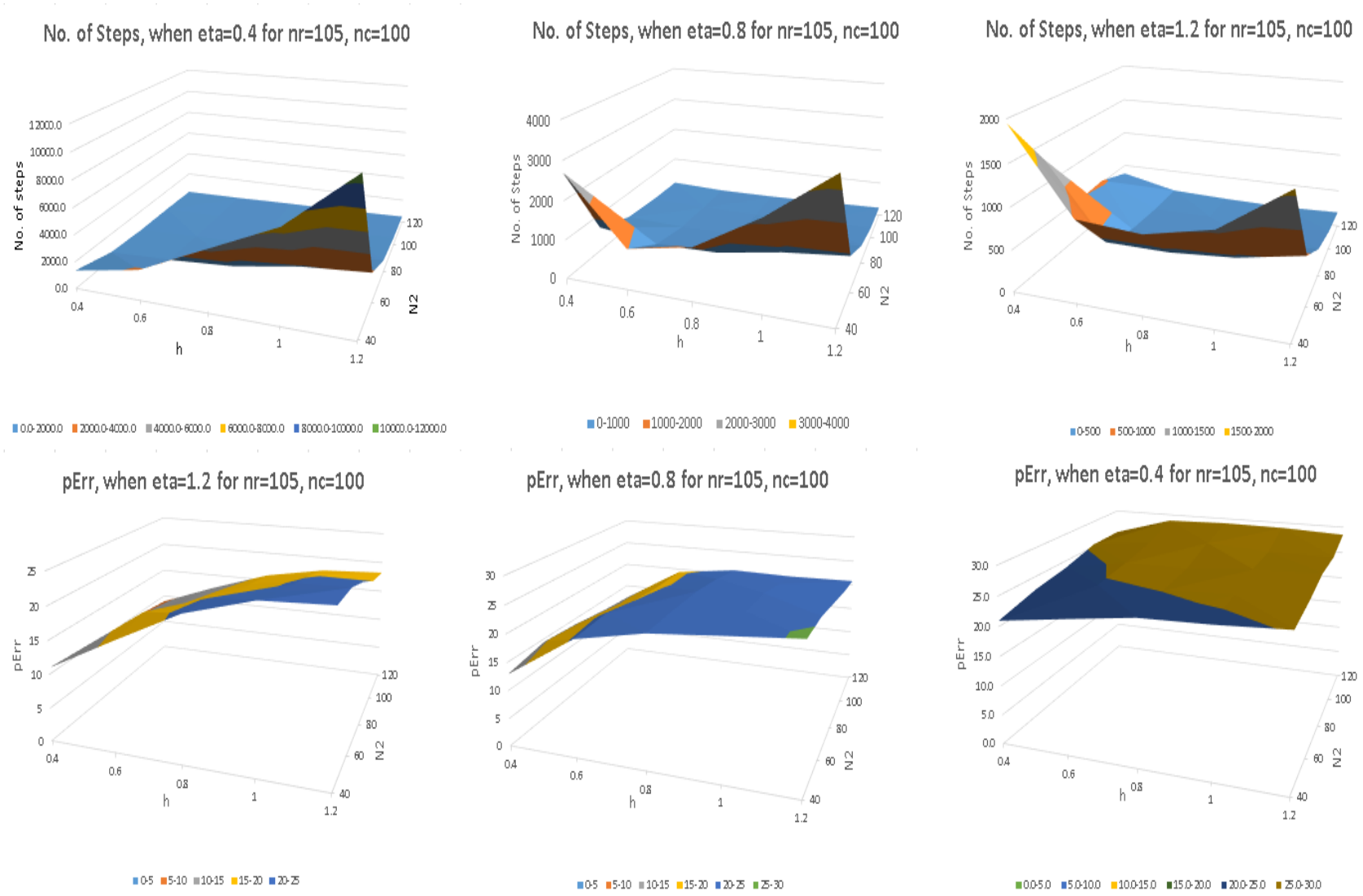

Figure 4-4: Graphical representation of Table 4-6 and Table 4-7

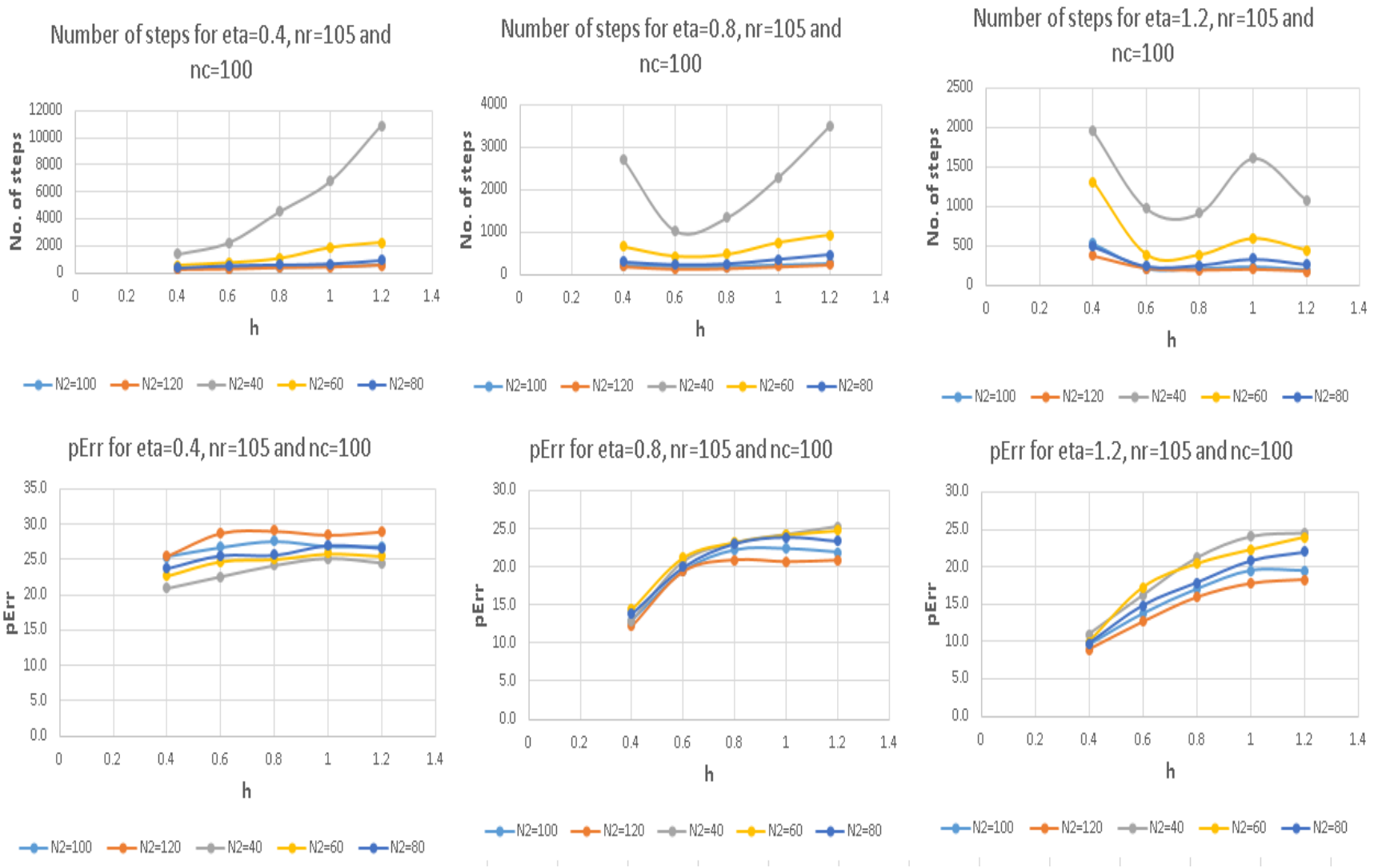

Figure 4-5: scatter and line graphs for Table 4-6 and Table 4-7

We concluded from the first experiment that the values of nr and nc do not affect the results to a large extent. Therefore, Let us analyze the test results for another training set, i.e. nr=110 and nc=100, and if the conclusion of the first experiment holds true, we should get the same optimal values of eta, N2 and h. Following tables show the mean values of percentage error and number of steps after 10 trainings i.e. we trained our network 10 times for each combination of eta, N2 and h and we get following values.

Mean values of number of training steps for nr=110 and nc=100											
Eta	**h**		**Mean**	**Eta**	**h**		**Mean**	**Eta**	**h**		**Mean**
0.4	**0.4**	**100**	271.8	**0.8**	**0.4**	**100**	228.0	**1.2**	**0.4**	**100**	406.1
		120	249.0			**120**	175.7			**120**	400.8
		40	1212.2			**40**	2359.9			**40**	4528.5
		60	554.4			**60**	591.5			**60**	2190.3
		80	382.5			**80**	280.0			**80**	620.1
	0.6	**100**	372.1		**0.6**	**100**	155.4		**0.6**	**100**	227.2
		120	264.6			**120**	133.3			**120**	217.9
		40	2753.2			**40**	1037.4			**40**	1115.2
		60	783.7			**60**	361.6			**60**	391.2
		80	465.3			**80**	221.9			**80**	245.5
	0.8	**100**	412.6		**0.8**	**100**	188.1		**0.8**	**100**	194.7
		120	313.8			**120**	141.7			**120**	180.4
		40	4112.2			**40**	1117.2			**40**	1016.8
		60	1342.3			**60**	450.1			**60**	366.5
		80	685.8			**80**	237.6			**80**	275.8
	1	**100**	520.1		**1**	**100**	265.9		**1**	**100**	217.4
		120	463.9			**120**	248.8			**120**	187.7
		40	11159.9			**40**	3333.5			**40**	1818.0
		60	2539.6			**60**	788.6			**60**	552.2
		80	1017.1			**80**	441.4			**80**	314.1
	1.2	**100**	489.1		**1.2**	**100**	246.7		**1.2**	**100**	204.1
		120	406.6			**120**	185.9			**120**	163.3
		40	6570.9			**40**	2189.7			**40**	1371.0
		60	1532.1			**60**	647.9			**60**	463.4
		80	769.8			**80**	371.1			**80**	254.5

Table 4-8: Mean values of Number of Training Steps for nr=110 and nc=100 for font size 11

In table 4-8, we can see that best mean value for number of steps is 176 that occurs when eta=0.8, N2=120 and h=0.4, and these are the same values, that we got for nr=105 and nc=100.

Mean values of percentage error for nr=110 and nc=100											
Eta	**h**		**Mean**	**Eta**	**h**		**Mean**	**Eta**	**h**		**Mean**
		100	25.0			**100**	13.0			**100**	9.1
		120	25.6			**120**	12.0			**120**	8.7
	0.4	**40**	21.4		**0.4**	**40**	13.8		**0.4**	**40**	18.5
		60	21.9			**60**	14.1			**60**	18.4
		80	23.8			**80**	13.2			**80**	8.6
		100	27.0			**100**	19.7			**100**	13.5
		120	28.3			**120**	18.8			**120**	12.3
	0.6	**40**	22.5		**0.6**	**40**	21.0		**0.6**	**40**	16.3
		60	24.4			**60**	21.2			**60**	15.4
		80	25.6			**80**	18.9			**80**	14.4
		100	27.6			**100**	21.5			**100**	17.0
		120	28.7			**120**	21.0			**120**	15.1
0.4	**0.8**	**40**	23.4	**0.8**	**0.8**	**40**	23.7	**1.2**	**0.8**	**40**	21.2
		60	24.0			**60**	23.6			**60**	19.3
		80	26.7			**80**	23.1			**80**	18.0
		100	26.8			**100**	23.6			**100**	19.7
		120	28.9			**120**	21.3			**120**	18.7
	1	**40**	25.1		**1**	**40**	25.4		**1**	**40**	24.6
		60	25.7			**60**	24.7			**60**	23.6
		80	26.5			**80**	24.1			**80**	22.5
		100	27.2			**100**	22.1			**100**	19.4
		120	28.8			**120**	21.6			**120**	18.0
	1.2	**40**	25.1		**1.2**	**40**	24.7		**1.2**	**40**	24.1
		60	25.5			**60**	24.0			**60**	23.4
		80	25.7			**80**	24.1			**80**	20.8

Table 4-9: Mean values of percentage error for nr=110 and nc=100 for font size 11

In table 4-9, we can analyze that the best mean percentage error value is 8.7, when eta=1.2, N2=120 and h=0.4, but number of steps for these values are excessively high. That is why we can easily say that eta=0.8, N2=120 and h=0.4 are best values because percentage error is not very high, it is just 12% and we have got a very low value for number of steps as well. For the other two training sets, we got the same results. That is eta=0.8, N2=120 and h=0.4 are the optimum values for best performance of our algorithm. We can see the results of table 4-8 and table 4-9 in figure 4-5.

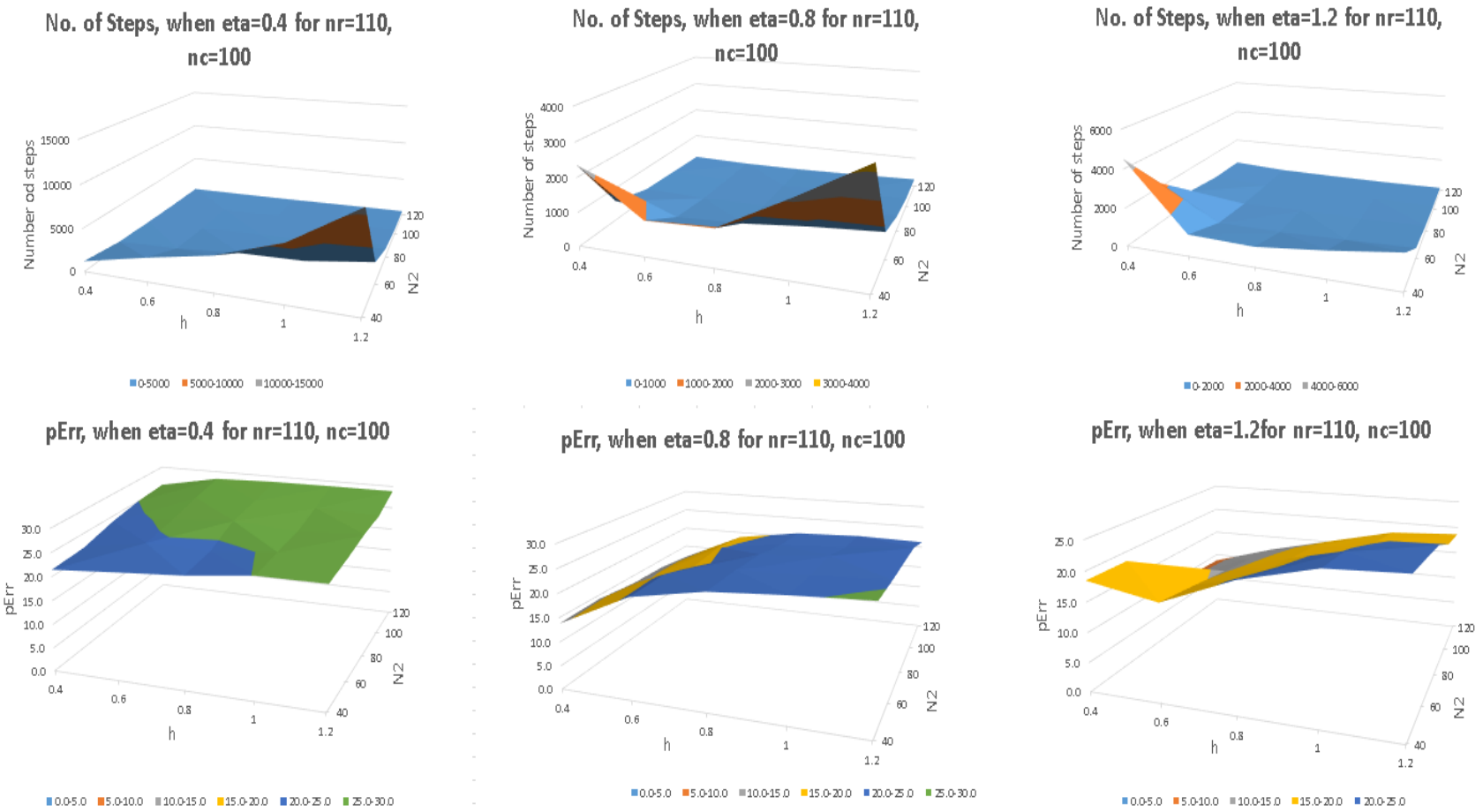

Figure 4-6: Surface plot representation of Table 4-8 and Table 4-9

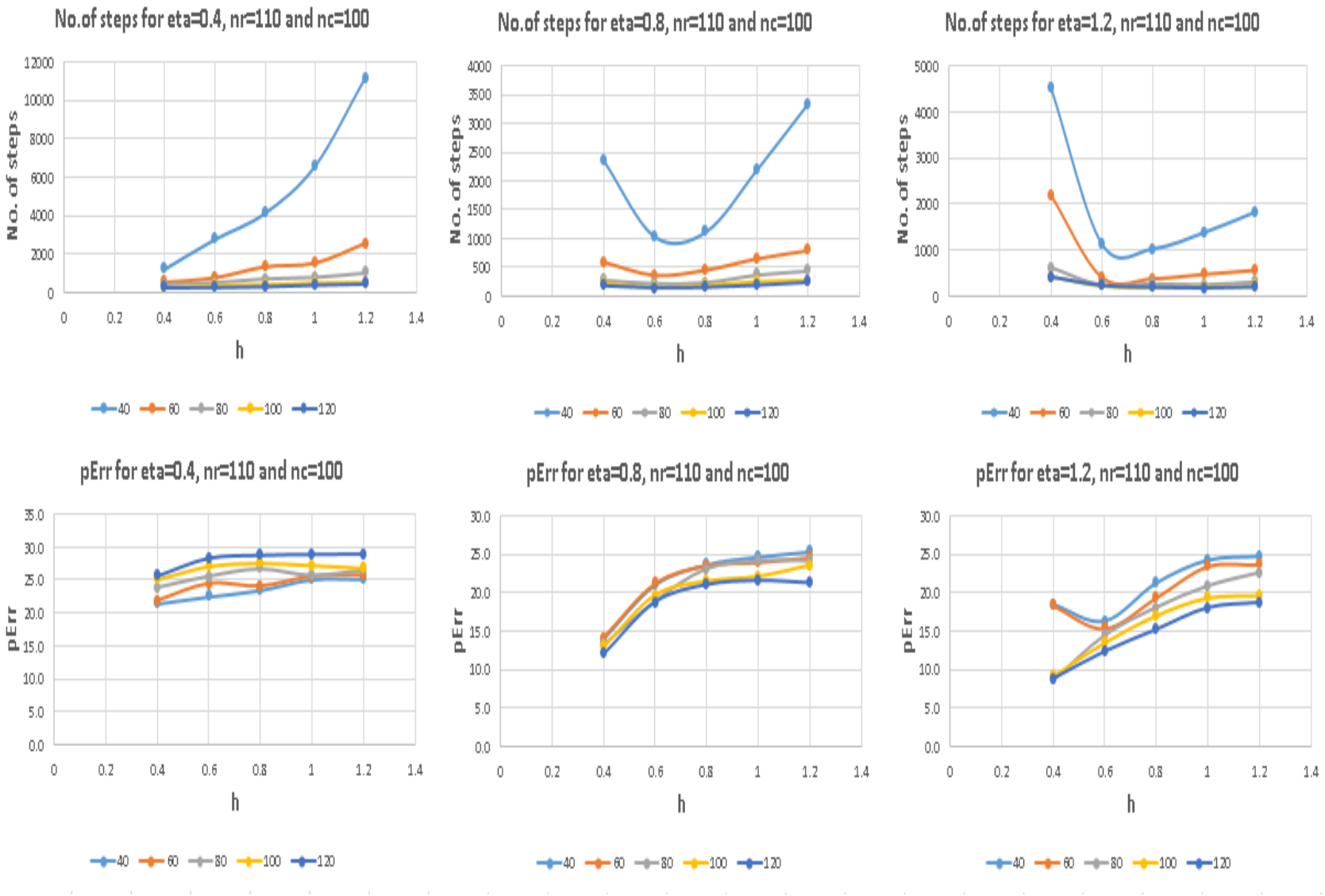

Figure 4-7: Scatter plot representation of Table 4-8 and Table 4-9

4.5.3. Optimization of parameters for font size 10 and 11

In the previous test we found that, eta=0.8, N2=120 and h=0.4 turned out to be the optimum values for the best performance of our algorithm. In this test, we update the training set and introduce fonts of 10 points as well, that means this time our training set contain fonts of 10 and 11 points, that makes our training set double the size of the training set used in the previous test.

The specifications for the third test are as follows:

Number of fonts for training	106
Number of fonts for validation	22
Number of characters	26
Training sets (nr x nc)	105x100, 110x100, 120x105, 125x115, 105x105 and 115x120
Font size	10 and11
eta	0.4, 0.8
h	0.4, 0.8, 1.2
N2	40, 60, 80, 100, 120
maxSteps	15000

Table 4-10: Specifications: optimization of eta, N2 and h for font size 10 and 11

This time we changed some specification for out test, for example for this test we use only two values of eta i.e. 0.4 and 0.8. We also reduced the number of values of h from 5 to 3, that means, we will only use 0.4, 0.8 and 1.2, and we will conduct this test for 6 training sets with different values of nr and nc.

To start with, the first thing to do was to create the new training sets with two font sizes i.e.10 and 11, and we used the same functions and procedure to create these training sets as per table 4-10. All the other procedures were same as used in second test. Let us now analyze the results of this test and then see if these tests again endorse the previous ones.

Mean Values of No. Of steps for nr=120 and nc=105			
Eta	h	N2	Mean
0.4	0.4	100	930
		120	754
		40	5430
		60	2734
		80	1190
	0.8	100	1701
		120	1055
		40	14834
		60	6247
		80	2817
	1.2	100	3495
		120	1792
		40	15000
		60	13525
		80	6027
0.8	0.4	100	489
		120	378
		40	4444
		60	1374
		80	703
	0.8	100	720
		120	523
		40	6169
		60	2238
		80	1183
	1.2	100	1256
		120	906
		40	14999
		60	5075
		80	2295

Table 4-11: Mean values of No. of steps for eta=0.4 and 0.8, nr=120 and nc=105 for font size 10 and 11

Table 4-11 shows that, the algorithm uses minimum number of steps, when eta=0.8, N2=120 and h=0.4, that is the same result as we got in the previous test. Let us now see, what happens with percentage error.

Mean Values of percentage error for nr=120 and nc=105			
Eta	h	N2	Mean
0.4	0.4	100	19.5
		120	19.5
		40	16.8
		60	17.8
		80	19.1
	0.8	100	21.6
		120	21.7
		40	19.5
		60	20.6
		80	20.8
	1.2	100	22.4
		120	22.5
		40	23.1
		60	21.4
		80	22.4
0.8	0.4	100	14.2
		120	12.6
		40	12.0
		60	12.9
		80	13.7
	0.8	100	20.5
		120	19.4
		40	19.5
		60	20.0
		80	20.5
	1.2	100	21.3
		120	21.1
		40	20.3
		60	21.3
		80	21.1

Table 4-12: Mean values of percentage error for eta=0.4 and 0.8, nr=120 and nc=105 for font size 10 and 11

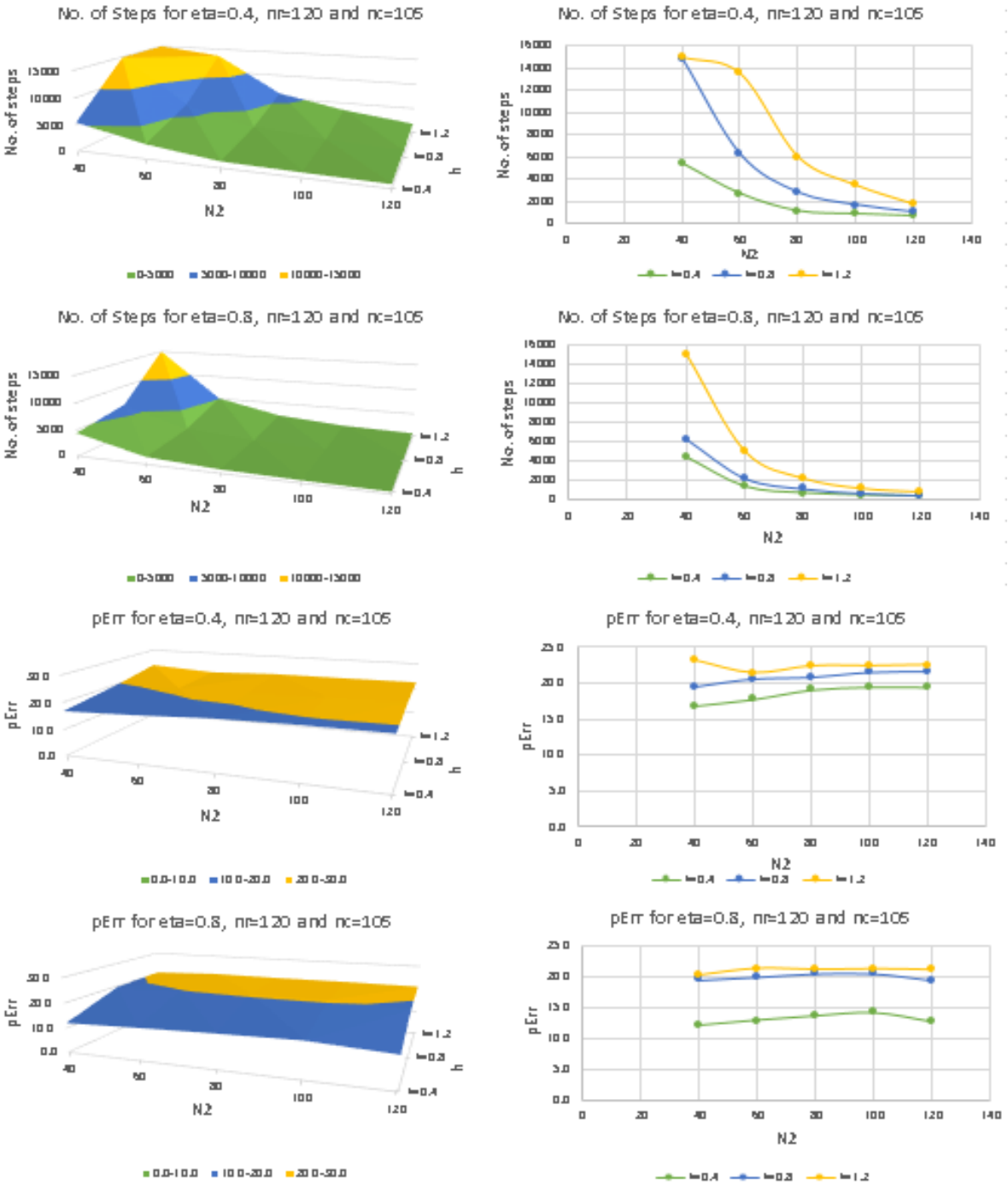

Figure 4-8: Mean Values of No. of steps and pErr for nr=120 and nc=105 represented in form of surface graphs and line graphs

In Table 4-12, we can see that the best value for pErr occurs when eta=0.8, N2=40 and h=0.4, but for these values, number of steps is very high. Therefore, the second best value is 12.6% for eta=0.8, N2=120 and h=0.4. The results of this test for nr=120 and nc=105 are satisfying in terms that the previous test also gave similar result. We can see the graphical representation of the results for table 4-11 and 4-12 in figure 4-8.

Let us now analyze a training set that has same value of nr and nc as in the previous test that we did for font size 11. Let us take nr=105 and nc=100 and train our ANN according to the specifications given in table 4-10 and see what happens. The behavior of the tests conducted until now tells us that the optimum values of the parameters should remain eta=0.8, N2=120 and h=0.4. Following are the tables for number of training steps and percentage error for training set 105x100 using font sizes 10 and 11.

Mean Values of No. of steps for nr=105 and nc=100 for font size 10 and 11			
Eta	**h**	**N2**	**Mean**
0.4	0.4	100	937
		120	681
		40	5386
		60	2125
		80	1368
	0.8	100	1817
		120	1034
		40	14715
		60	5405
		80	3402
	1.2	100	2854
		120	1652
		40	15000
		60	13233
		80	6865
0.8	0.4	100	531
		120	381
		40	4931
		60	1233
		80	668
	0.8	100	775
		120	499
		40	5756
		60	2262
		80	1199
	1.2	100	1242
		120	791
		40	13835
		60	4883
		80	2101

Table 4-13: Mean values of number of training steps for nr=105, nc=100 and font sizes 10 and 11

Mean Values of percentage error for nr=105 and nc=100 for font size 10 and 11			
Eta	**h**	**N2**	**Mean**
0.4	0.4	100	20.0
		120	20.6
		40	17.3
		60	18.3
		80	19.0
	0.8	100	20.8
		120	22.0
		40	18.8
		60	20.3
		80	20.5
	1.2	100	22.0
		120	22.7
		40	22.5
		60	22.2
		80	21.8
0.8	0.4	100	14.0
		120	12.6
		40	12.5
		60	13.7
		80	13.9
	0.8	100	20.1
		120	19.2
		40	18.7
		60	20.5
		80	20.3
	1.2	100	21.4
		120	20.1
		40	20.7
		60	21.6
		80	21.9

Table 4-14: Mean values of percentage error for nr=105, nc=100 and font sizes 10 and 11

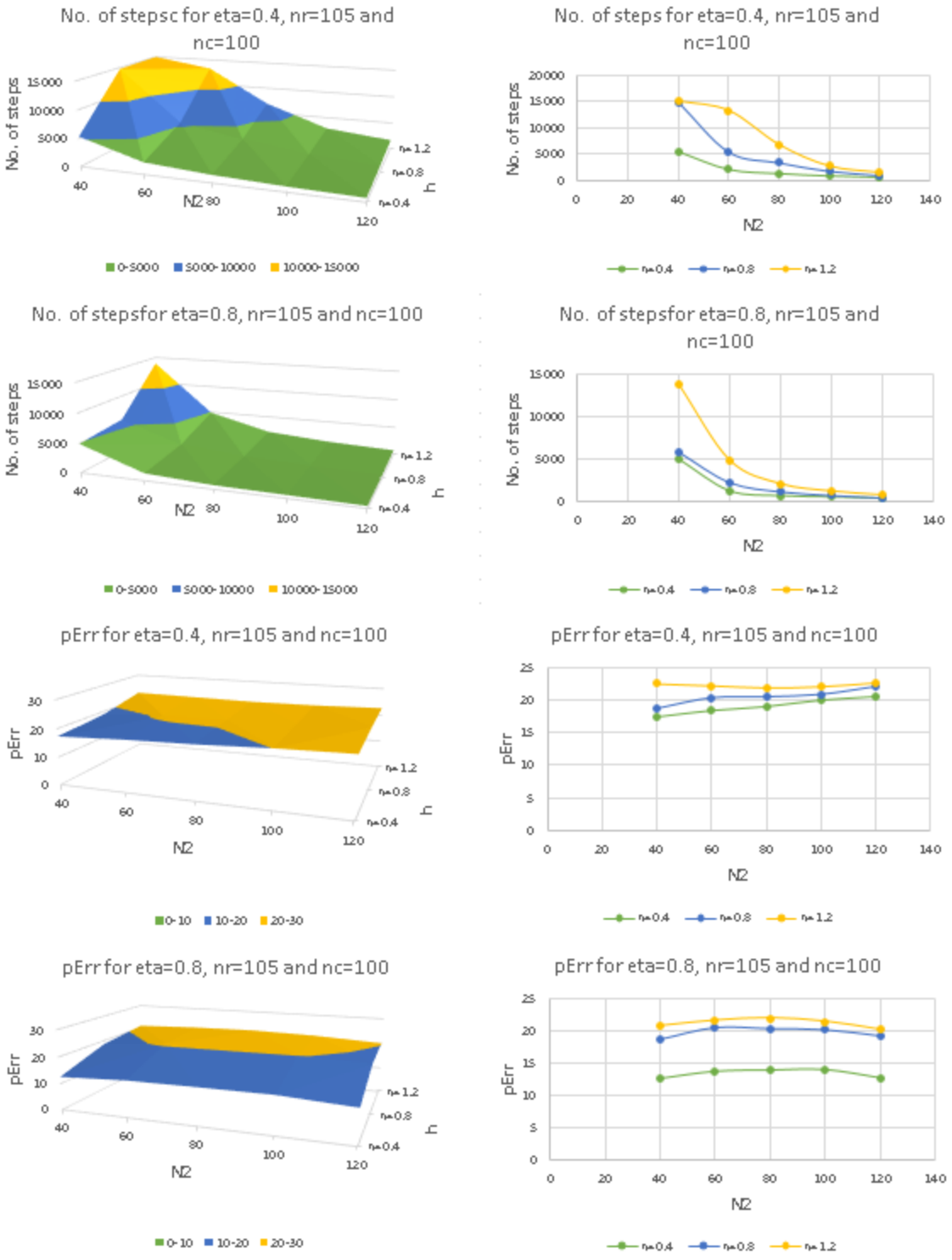

Figure 4-9: Mean Values of No. of steps and pErr for nr=105 and nc=100 represented in form of surface graphs and line graphs

We can see in table 4-13 and 4-14 that the optimum value for number of training steps is for eta=0.8, N2=120 and h=0.4. Whereas we get an optimal value for percentage error for the same values of parameters. This value is not the optimum one, but it isoptimal enough to be considered as the best fit. Figure 4-9 provides a graphical visualization of the results obtained in tables 4-13 and 4-14.

4.6. Conclusion

From the experimental results, we get following important conclusions:

- Values of nr and nc do not affect performance of ANN to large extent.
- Our proposed ANN performs best for eta=0.8, N2=120 and h=04 for all values of nr and nc. This is the optimal choice taking into account the number of training steps and percentage of errors.
- We get an optimal value for percentage error not the optimum when eta=0.8, N2=120 and h=0.4 for all the values of nr and nc, but that is the best trade off.
- When we increase the number of font sizes, we increase the number of training inputs, thus, we see an increase in number of training steps.

Chapter 5: Conclusion and Future work

The main idea of this thesis work was to validate the ANN and its parameters that was developed to work as an OCR. The major activities involved in this research were creating the training and validation sets, training the ANN and testing the ANN. First, a training set of font size 11 and 106 font was created to train the neural networks with six different values of nr and nc (number of rows and number of columns respectively), nr and nc corresponds to the dimensions of the binary matrix that represents a single character. The ANN was trained for 36 different matrices using all possible values of nr and nc and parameters of BP algorithm are fixed (i.e. eta, N and h). Then ANN is validated for all the values of nr and nc. The results of this test proved that the performance of the ANN does not depend on nr and nc.

After that, we trained the ANN with different values of parameters eta, N and h with four selected dimensions of nr and nc and font size 11. The experimental results suggested that the best values of eta, N and h, are 0.8, 120 and 0.4 respectively.

Furthermore, another test was carried out but this time with a training set that contain fonts of size 11 and 10. Therefore, this time we had a larger training set. We trained the ANN and tested it and the results were the same as stated above. Therefore, we came to know that the change in the values of nr and nc does not affect the performance of the ANN to a large extent but the performance majors varry a lot when the different values of parameters of BP algorithm are used. The ANN performance is optimal, when the values of eta, N and h are 0.8, 120 and 0.4 respectively.

Experimental Results of Validation		
Parameters	**Optimal Values**	**Remarks**
nr	N/A	This parameters does not affect the ANN performance
nc	N/A	This parameters does not affect the ANN performance
eta	0.8	Optimal value is 0.8 as it gives the best performnace
N2	120	Performnace is better as N2 increases
h	0.4	Performnace is better as h decreases

Table 5-1: Experimental Results after Validation of data sets with font size 10 and 11

In this thesis, we kept into account only English characters and the fonts, but the next step of the project is to validate the BP algorithm with mathematical formulae and then use it practically. As we discussed earlier in this thesis that the main idea is to develop an OCR that can be helpful in learning and education for visually impaired people, so most probably, the next phases will be actually implementing it as a software and then test it in real time with some volunteers. I hope this project proves to be a big success and we keep doing this kind of work in future as well.

REFERENCES

1. Byrne, J. H. (2016). Introduction to neurons and neuronal networks. In Neuroscience online. Retrieved from http://neuroscience.uth.tmc.edu/s1/introduction.html.
2. Kandel, E. R., Schwartz, J. H., and Jessell, T. M. (1995). Nerve cells and behavior. In Essentials of neuroscience and behavior (pp. 21-31). Norwalk, CT: Appleton & Lange.
3. Nicholls, J. G., Martin, A. R., Wallace, B. G., and Fuchs, P. A. (2001). Principles of signaling and organization. In From neuron to brain (4th ed., pp. 3-9). Sunderland, MA: Sinauer Associates.
4. Purves, D., Augustine, G. J., Fitzpatrick, D., Katz, L. C., LaMantia, A.-S., and McNamara, J. O. (1997). The organization of the nervous system. In Neuroscience (pp. 1-10). Sunderland, MA: Sinauer Associates.
5. Reece, J. B., Urry, L. A., Cain, M. L., Wasserman, S. A., Minorsky, P. V., and Jackson, R. B. (2011). Neuron structure and organization reflect function in information transfer. In Campbell biology (10th ed., p. 1062-1064). San Francisco, CA: Pearson.
6. Sadava, D. E., Hillis, D. M., Heller, H. C., and Berenbaum, M. R. (2009). The spinal cord transmits and processes information. In Life: The science of biology (9th ed., pp. 1033-1034). Sunderland, MA: Sinauer Associates
7. Sadava, D. E., Hillis, D. M., Heller, H. C., and Berenbaum, M. R. (2009). Neurons and nervous systems. In Life: The science of biology (9th ed., pp. 988-993). Sunderland, MA: Sinauer Associates.
8. Swift, A. (2015). Pain management 2: Transmission of pain signals to the brain. Nursing Times, 111(40), 22-26.
9. R. Rojas: Neural Networks, Springer-Verlag, Berlin, 1996

10. Richard Bland, 1998. Learning XOR: exploring the space of a classic problem.

11- John A. Bullinaria, 2015, Neural Computation

12- Sonali B.M., Priyanka W. Research paper on basic of Artificial Neural network, 2014

13- D. Anderson and G. McNeill, Artificial Neural Network Technologies, Data and Analysis Center for Software, Rome Laboratories, 1992.

14- Magali R. G. Meireles, Paulo E. M. Almeida and Marcelo Godoy Simões, A Comprehensive Review for Industrial Applicability of Artificial Neural Networks, 2003

15- Suresh et al.: PARALLEL IMPLEMENTATION OF BACK-PROPAGATION ALGORITHM IN NETWORKS OF WORKSTATIONS.

16- Giuseppe Airo Farulla, Tiziana Armano, Anna Capietto, Nadir Murru, and

Rosaria Rossini, Artificial neural networks and fuzzy logic for recognizing alphabet characters and mathematical symbols.

17- Wilson, D. R., and T. R. Martinez, "The Inefficiency of Batch Training for Large Training Sets", in Proceedings of the International Joint Conference on Neural Networks (IJCNN2000), Vol. II, pp. 113-117, July 2000.

18- Jacobs, R. A., "Increased Rates of Convergence through Learning Rate Adaptation," Neural Networks, Vol. 1, No. 4, pp. 295-307, 1988.

19- D. Randall Wilson, Tony R. Martinez, The Need for Small Learning Rates on Large Problems, 2001

20- Shuxiang Xu and Ling Chen, A Novel Approach for Determining the Optimal Number of Hidden Layer Neurons for FNN's and Its Application in Data Mining, 2008.

21- E. J. Teoh, K. C. Tan, and C. Xiang, Estimating the Number of Hidden Neurons in a Feedforward Network Using the Singular Value Decomposition, 2006

22- Nadir Murru, Rosaria Rossini, A Bayesian approach for initialization of weights in backpropagation neural net with application to character recognition, 2016.

23- Jim Y.F. Yam, Tommy W.S. Chow, A weight initialization method for improving training speed in feedforward neural network, 1999

24- Shital Solanki and H.B.Jethva, A review on back propagation algorithms for Feedforward Networks, January 2013

25- Imran Shafi, Jamil Ahmad, MIEEE, Syed Ismail Shah, Sr. MIEEE, and Faisal M Kashif Impact of Varying Neurons and Hidden Layers in Neural Network Architecture for a Time Frequency Application.

26- Rojas, Neural Networks, A Systematic Introduction, 1996

27- Kauleshwar Prasad and Shubham Agrawal, Character recognition using Artificial neural networks, International Journal of Advanced Engineering Research and Studies, 2015

28- Rita Alkula and Kari Pieska, Optical character recognitiom in microfilmed newspaper library collection, 1994

29- Sameeksha Barve, Optical Character Recognition Using Artificial Neural Network, International Journal of Advanced Research in Computer Engineering & Technology Volume 1, Issue 4, June 2012

30- Rókus Arnold, Póth Miklós, Character Recognition Using Neural Networks.

31- Bekir Karlik and A. Vehbi Olgac, Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks.

32- Shelley B. Wepner & Kathleen A. Bowes, Issues in technology using assistive technology for literacy development.

33- Rosenfeld, Amnon; Dvorachek, Michael; Rotstein, Ilan (2000). "Bronze Single Crown-like Prosthetic Restorations of Teeth from the Late Roman Period". Journal of Archaeological Science

34- Schantz, Herbert F. (1982). The history of OCR, optical character recognition. [Manchester Center, Vt.]: Recognition Technologies Users Association.