PREDICTING STOCK MARKET PRICE BY USING ARTIFICIAL NEURAL NETWORK TRAINED WITH FOUR DIFFERENT ALGORITHM: A PILOT STUDY OF 15 BANGLADESHI COMPANY

MD FURKANUZZAMAN

MASTER OF SCIENCE

NOAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

PREDICTING STOCK MARKET PRICE BY USING ARTIFICIAL NEURAL NETWORK TRAINED WITH FOUR DIFFERENT ALGORITHM: A PILOT STUDY OF 15 BANGLADESHI COMPANY

MD FURKANUZZAMAN

Thesis submitted in partial fulfilment of the requirements for the award of the degree of Master of Science in Information and Communication Engineering

Department of Information and Communication Engineering

NOAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

MARCH 2021

SUPERVISOR'S DECLARATION

I hereby declare that I have checked this thesis and in my opinion this thesis is adequate in terms of scope and quality for the award of the degree of Master of Science in Information and Communication Engineering.

DR. MD. ASHIKUR RAHMAN KHAN
PROFESSOR & CHAIRMAN
DEPARTMENT OF INFORMATION AND COMMUNICATION ENGINEERING
NOAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY (NSTU)

ZAYED- US- SALEHIN
ASSISTANT PROFESSOR
DEPARTMENT OF INFORMATION AND COMMUNICATION ENGINEERING
NOAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY (NSTU)

STUDENT'S DECLARATION

This thesis report is submitted to the Dept. of Information and Communication Engineering, Noakhali Science and Technology University (NSTU) in partial fulfillment of the requirements for having the M.Sc degree in ICE. This is also needed to certify that the thesis work is under the M.Sc course of ICE, NSTU. So, I hereby declare that this thesis report has not been submitted elsewhere for the requirement of any kind of degree, diploma or publication.

MD. FURKANUZZAMAN
ID: ASH1811MSc102M
SESSION: 2017-2018

ACKNOWLEDGEMENTS

I am very grateful and would like to express my gratitude to my supervisor Professor Dr. Md. Ashikur Rahman Khan for his setting up, invaluable guidance, continuous encouragement and constant support in making this research possible. He has always encouraged me with his outstanding and efficient professional conduct. I am grateful for all the support that I have received from him.

My sincere thanks to all my co-supervisor and teachers of the Department of Information and Communication Engineering (ICE) in NSTU for their valuable suggestion and academic help during the course of work.

I acknowledge my sincere indebtedness and gratitude to my parents for their love, dream and sacrifice throughout my life.

ABSTRACT

Stock market price prediction is now a popular and important topic in financial and academic studies because stock market plays a vital rule in economy. Stock market price prediction is the act of trying to determine the future value of company stock. Stock market prices are actually time-series data and Artificial Neural Networks (ANNs) have the ability to find non-linear correlations between time-series data which makes it the best approach to predict stock market prices. Many researchers working on this topic and try to find the best algorithm with using stock market dataset which is suitable for predicting stock price. In this research historical data from Dhaka Stock Exchange is used to train and predict the price by using ANN. The Artificial Neural Network (ANN) is implemented by using multilayer Feedforward backpropagation model. In this paper fifteen companies six years data have been analyzed. To predict the specific result, the model has been trained in four different algorithm with change their parameter. Number of hidden layer, hidden neuron and percentage of training data have been change to get better output. After the training and testing process the predicted values are compared with the real data to find the accuracy. The trained network with the highest accuracy rate will able to predict the best possible price of the stock market.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

ABBREVİATİON

ANN	Artificial Neural Network
SVM	Support Vector Mechine
AI	Artificial Intelligence
DSE	Dhaka Stock Exchange
FFNN	Feed-Forward Neural Network
MLP	Multilayer Perceptron
LM	Levenberg Marquardt
BR	Bayesian Regularization
SCG	Scale Conjugate Gradient
QN	Quasi Newton
DSE	Dhaka Stock Exchange
MSE	Mean Square Error
LSTM	Long Sort-term Memory
ML	Machine Learning
MLP	Multilayer Perceptron

CHAPTER 1

INTRODUCTION

1.1 İNTRODUCTİON

Stock Market prediction and analysis is the act of trying to determine the future value of a company stock or other financial instrument traded on an exchange. Stock market is the important part of economy of the country and plays a vital role in the growth of the industry and commerce of the country that eventually affects the economy of the country. Both investors and industry are involved in stock market and wants to know whether some stock will rise or fall over certain period of time. The stock market is the primary source for any company to raise funds for business expansions. It is based on the concept of demand and supply. If the demand for a company's stock is higher, then the company share price increases and if the demand for company's stock is low then the company share price decrease.

From the beginning of time it has been man's common goal to make his life easier. The prevailing notion in society is that wealth brings comfort and luxury, so it is not surprising that there has been so much work done on ways to predict the markets. Therefore forecasting stock price or financial markets has been one of the biggest challenges to the Artificial Intelligence (AI) community. Various technical, fundamental, and statistical indicators have been proposed and used with varying results. However, none of these techniques or combination of techniques has been successful enough. The objective of forecasting research has been largely beyond the capability of traditional AI research which has mainly focused on developing intelligent systems that are supposed to emulate human intelligence. By its nature the stock market is mostly complex and volatile. With the development of neural networks, researchers and investors are hoping

that the market mysteries can be unraveled. Artificial neural networks inspired by human brain cells' activity can learn the data patterns and generalize their knowledge to recognize the future new patterns.

Researches on neural networks show that neural networks have great capability in pattern recognition and machine learning problems such as classification and regression. These days neural networks are considered as a common data mining method in different fields like economy, business, industry, and science [1].

The application of neural networks in prediction problems is very promising due to some of their special characteristics. First, traditional methods such as linear regression and logistic regression are model based while neural networks are self-adjusting methods based on training data, so they have the ability to solve the problem with a little knowledge about its model and without constraining the prediction model by adding any extra assumptions. Besides, neural networks can find the relationship between the input and output of the system even if this relationship might be very complicated because they are general function approximators. Consequently, neural networks are well applied to the problems in which extracting the relationships among data is really difficult but on the other hand there exists a large enough training data sets. It should be mentioned that, although sometimes the rules or patterns that we are looking for might not be easily found or the data could be corrupted due to the process or measurement noise of the system, it is still believed that the inductive learning or data driven methods are the best way to deal with real world prediction problems. Second, neural networks have generalization ability meaning that after training they can recognize the new patterns even if they haven't been in training set. Since in most of the pattern recognition problems predicting future events (unseen data) is based on previous data (training set), the application of neural networks would be very beneficial. Third, neural networks have been claimed to be general function approximators. It is proved that multilayer feedforward neural network can approximate any complex continuous function that enables us to learn any complicated relationship between the input and the output of the system.

1.2 PROBLEM STATEMENT

Stock market is very vast and difficult to understand. It is considered too uncertain to be predictable due to huge fluidity of the market. Stock market prediction task is

interesting as well as researchers and academics into two groups, those who believe that we can devise mechanisms to predict the market and those who believe that the market is efficient and whenever new information comes up the market absorbs it by correcting itself, thus there is no space for prediction.

Investing in a good stock but at a bad time can have disastrous result, while investing in a stock at the right time can bear profits. Financial investors of today are facing this problem of trading as they do not properly understand as to which stocks to buy or which stocks to sell in order to get optimum result. So, the purposed thesis will reduce the problem with suitable accuracy faced in such real time scenario.

1.3 OBJECTIVES

There are some facts or theme which will be the main objective of this thesis. Such as

i. To predict an approximate value of share price.
ii. To invest easily in stock market for an investor.
iii. To give idea about the stock market.
iv. To predict the financial market performance based on per day evening price of any company.

1.4 SCOPE OF STUDY

In order to make profit from stock market it is important to discovery knowledge from stock market dataset. Stock market is not only important for investor or industry but also important for a country economics. Predicting the stock market price is become high demand. Many researchers working on this fact and try to discover the best algorithm for predicting the stock price. In this thesis data is collected from Dhaka stock exchange. This research uses feedforward multilayer neural network with four different training algorithm.

1.5 THESIS OUTLINE

This thesis organized as follows: Chapter 1 offers the reader a glance into the financial stock market system and machine learning techniques. This chapter aims to get

an interest into this interesting phenomenon. By describing and discussing the surroundings of, a purpose is identified and a research objective is posed. Finally, the overall arrangement of this study is explained. Chapter 2 literature review where discussed some of the existing researches in the sector of financial stock market prediction with machine learning techniques. From this part, readers gain knowledge about the former researches and research types. This chapter also discussed details some existing works of financial stock market prediction and some important papers concepts help to our research work. Chapter 3 provides the requirement and the process of the implementation and simulation of our thesis. The working procedure of finding prediction accuracy is beneficial and it is done by the software. The workflow of model is represented with figures. This forms a base for the research approach and design. Chapter 4 present prediction accuracy of ANN is represented by figures which helps to the investors to understand about the market situation. Chapter 5 discussed about achievements and conclusion of this thesis.

CHAPTER 2

LİTERATURE REVİEW

2.1 İNTRODUCTİON

İt is the literature review where disscuss some of exsisting researches in the sector of financial stock market prediction with machine learning techniques. From this part, readers gain knowledge about the former researches and research types. This chapter also discussed details some existing works of financial stock market prediction and some important papers concepts help to our research work.

2.2 PREVIOUS WORK

In 2017, Sujin Pyo, Jaewook Lee, Mincheol Cha, and Huisu Jang. predicted the KOSPI 200 index with SVMs and ANNs using Google Trends [2]. Results showed a maximum 52% accuracy with shorter periods using Google Trends; however, this is not sufficient for real-world investors.

Pekkaya and Hamzacebi, in 2007 have taken an approach to conduct a comparative study on the results from using a linear regression in comparison to a ANN model [3]. The objective of their research was to forecast macro variables.

In 2016, Ishmat Pasha used artificial neural network to predict stock market price prediction [4]. To predict the specific result the model has been trained in different category of networks based on time period. Three different time period data have been

chosen as one year data, six months data and three months data. This thesis is mainly focusing on short-time perspective analysis.

Md. Tawhid Anwar and Saidur Rahman in 2019, proposed different machine learning technique to predict Dhaka stock market price [5]. They also compare these method for gain higher accuracy. They predict market price by using Deep Neural Network, Recurrent neural network, LSTM and Support Vector Machine method. Deep neural network given best accuracy rate and that is 88%.

In 2015, Tsai and Wang did a research where they tried to predict stock prices by using ensemble learning, composed of decision trees and artificial neural networks [6]. They created dataset from Taiwanese stock market data, taking into account fundamental indexes, technical indexes, and macroeconomic indexes. The performance of Decision Tree and Artificial Neural Network trained on Taiwan stock exchange data showed F-score performance of 77%. Single algorithms showed F-score performance up to 67%.

In 2017, J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, introduced such hybridization by training a 19-qubit gate model processor to solve a clustering problem, a foundational challenge in unsupervised learning [7]. Machine learning techniques have led to broad adoption of a statistical model of computing. The statistical distributions natively available on quantum processors are a superset of those available classically. Harnessing this attribute has the potential to accelerate or otherwise improve machine learning relative to purely classical performance. A key challenge toward that goal is learning to hybridize classical computing resources and traditional learning techniques with the emerging capabilities of general purpose quantum processors. This quantum/classical hybrid algorithm shows robustness to realistic noise, and we find evidence that classical optimization can be used to train around both coherent and incoherent imperfections.

V Kranthi Sai Reddy in 2018, explained the prediction of a stock using Machine Learning [8]. The technical and fundamental or the time series analysis is used by the most of the stockbroker while making the stock predictions. The programming language is used to predict the stock market using machine learning is Python. In this paper we propose a Machine Learning (ML) approach that will be trained from the available stocks data and gain intelligence and then uses the acquired knowledge for an accurate prediction. In this

context this study uses a machine learning technique called Support Vector Machine (SVM) to predict stock prices for the large and small capitalizations and in the three different markets, employing prices with both daily and up-to-the-minute frequencies.

K. Senthamarai Kannan, P. Sailapathi Sekar, M.Mohamed Sathik and P. Arumugam discussed various techniques which are able to predict with future closing stock price will increase or decrease better than level of significance [9]. They used automated computer programs using data mining and predictive technologies do a fare amount of trades in the markets. This technology is designed to help investors discover hidden patterns from the historic data that have probable predictive capability in their investment decisions.

In 2018, Ayodele A. Adebiyi., Aderemi O. Adewumi proposed a Machine Learning (ML) approach that will be trained from the available stocks data and gain intelligence and then uses the acquired knowledge for an accurate prediction [10]. The technical and fundamental or the time series analysis is used by the most of the stockbrokers while making the stock predictions. The programming language is used to predict the stock market using machine learning is Python. In this context this study uses a machine learning called Support Vector Machine (SVM) to predict stock prices for the large and small capitalizations and in the three different markets, employing prices with both daily and up-to-the-minute frequencies.

In 2015, Kai Chen, Yi Zhou, Fangyan Dai showed various machine learning algorithms such as neural networks, genetic algorithms, support vector machine, and others are used to predict stock prices [11]. They used RNNs to predict stock market. Long Short-Term Memory (LSTM) is one of the most successful RNNs architectures. They modeled and predicted China stock returns using LSTM. The historical data of China stock market were transformed into 30-days-long sequences with 10 learning features and 3 day earning rate labeling.

Shunrong Shen, Haomiao Jiang, Tongda Zhang, in 2012, proposed a new prediction algorithm that exploits the temporal correlation among global stock markets and various financial products to predict the next-day stock trend with the aid of SVM [12]. In particular, numerous studies have been conducted to predict the movement of stock market using machine learning algorithms such as support vector machine (SVM) and

reinforcement learning. Numerical results indicate a prediction accuracy of 74.4% in NASDAQ, 76% in S&P500 and 77.6% in DJIA.

In 2016, Mahdi Pakdaman Naeini, Hamidreza Taremian and Homa Baradaran Hashemi completed a research to predict Tehran stock market price by using Multilayer perceptron neural network and Elman recurrent network [13]. They used backpropagation algorithm to train the network. They compared these two method and got batter output from Multilayer perceptron neural network and that is 85%

In 2014, Younuf Yetis, Halid Kaplan and Mo Jamshidi predicted the NASDAQ stock market by using artificial neural network [14]. This paper focus on multilayer perceptron networks. They showed the result of using only leveberg marquardt algorithm and one hidden layer for training the network. They used two years data for analysis the stock market and predicted the price that is short time perspective analysis. In this paper, we used six years data and four algorithms with changing the number of hidden layer, hidden neuron and percentage of training data for trained the networks. After training, the accuracy of prediction for four algorithm is defined and compared them for best accuracy.

Table 2.1: Summary of the related literature review

Sl No	Title	Author	Year	Implemented Method	Result	Remark
01	Predictability of machine learning techniques to forecast the trends of market index prices	Sujin Pyo, Jaewook Lee, Mincheol Cha, and Huisu Jang	2017	Support Vector Mechine	52% accuracy	This is not sufficient for real world investor
02	Stock Market Price Prediction Using Artificial Neural Network	Ismith Pasha	2016	Feed Forward Neural Network	Accuracy rate 91%	Accuracy is good but this research is mainly focusing on short-time perspective analysis

03	External technology sourcing and innovation performance in LMT sectors	Tsai and Wang	2015	Decision Tree and Artificial Neural Network	Performance is 71% accuracy rate	Accuracy in not enough for prediction
04	Forecasting Stock Market Prices Using Advanced Tools of Machine Learning	Md. Tawhid Anwar And Saidur Rahman	2019	Logistic regression, SVM, Deep neural network	Compare these method and Deep neural network given best accuracy that is 88%	Accuracy is good but not best
05	Predicting Stock Markets with Neural Networks	Torkil Aamodt	2015	Support Vector Machine and Feed-forward Neural Network	Compare these two method and Feed-forward Neural Network given good result and that is 87%	Good accuracy but not best
06	Stock Market Value Prediction Using Neural Networks	Mahdi Pakdaman Naeini Hamidreza Taremian And Homa Baradaran Hashemi	2016	Feed forward multi layer Perceptron (MLP) and an Elman recurrent network	Multi layer Perceptron (MLP) neural network given best accuracy and that is 85%	Accuracy in not enough for prediction

2.3 SUMMARY

The analysis of literature review had broadened the scope of stock market price prediction issues. The information and findings collected form this chapter is used as a guidance to develop the proposed systems. In previous most of the researcher focus on only one company stock data that are not sufficient. Some research is mainly focused on short-time perspective analysis but that is not perfect for price prediction. The accuracy

for stock market prediction is not good enough using the methods which were used in these previous work. In this thesis, fifteen companies five years real data have been used to analyzed the stock market price. Also four training algorithm with changing hidden layer, neuron and training data are used to get better accuracy.

CHAPTER 3

METHODOLOGY

3.1 İNTRODUCTİON

The purposed method for developing the system consists of mainly three main steps. Firstly, data is collected and sorted for relevancy from various sources. Secondly, analysis is carried out on the collected data by examining the current market direction, tracking the industry group and specific companies after which the data is represented and scored accordingly. At last, an ANN is designed and a suitable algorithm yielding best accuracy is chosen to predict the stock value.

3.2 DATA COLLECTION AND DATA PROCESSING

3.2.1 Data Collection

This research attempts to predict the stock value with respect to the stock's previous value and trends. It requires historic data of stock market as the project also emphasizes on data mining techniques. So, it is necessary to have a trusted source having relevant and necessary data required for the prediction. We will be using Dhaka Stock Exchange website [15] as the primary source of data. Other source of data is the investing website [16].This website contains all the details such as date, opening value, closing value, highest value, lowest value, number of shares, change in stock values for each financial companies. It also provides the overall performance of Dhaka Stock Exchange

and performance of companies of different categories. The site is updated on daily basis and it is also a repository for years of stock market data for Bangladesh.

Table 3.1: Data Collection

Date	Price	Opening	High	Low	Share(K)	Change (%)
Jan 31, 2019	87.00	89.00	89.50	86.00	191.83	-1.69
Jan30, 2019	88.50	89.40	89.80	88.30	180.75	-1.01
Jan 29, 2019	89.40	90.40	90.40	88.90	225.96	0.34
Jan 28, 2019	89.10	87.10	90.90	87.10	471.23	1.37
Jan 27, 2019	87.90	88.00	88.20	87.20	181.90	-0.23
Jan 24, 2019	88.10	87.60	88.30	87.50	335.99	1.15
Jan 23, 2019	87.10	87.20	88.20	86.70	321.09	0.23
Jan 22, 2019	86.90	86.00	87.50	85.80	408.40	0.58
Jan 21, 2019	86.40	86.00	86.60	85.00	430.36	0.38
Jan 20, 2019	85.80	85.50	86.30	84.50	372.09	1.30
Jan 17, 2019	84.70	84.70	85.40	84.30	237.16	-0.82
Jan 16, 2019	85.40	84.90	86.40	84.70	655.90	0.95
Jan 15, 2019	84.60	85.50	86.00	81.10	535.30	-1.05
Jan 14, 2019	85.50	86.10	86.90	79.70	276.71	-1.16
Jan 13, 2019	86.50	86.10	87.00	85.60	575.16	0.46
Jan 10, 2019	86.10	86.60	86.60	85.50	254.72	0.23
Jan 09, 2019	85.90	86.40	86.60	85.50	521,94	-0.58
Jan 08, 2019	84.90	85.50	86.80	84.10	644.31	1.77
Jan 07, 2019	84.90	86.00	86.00	83.00	517.83	-1.05
Jan 06, 2019	85.80	84.30	86.20	82.80	811.80	1.78
Jan 03, 2019	84.30	81.80	85.50	81.80	1210	3.69

3.2.2 Data Processing

Only a small data pre-processing is required. First of all, after taking the all the collected raw data samples are included in the excel file. Next, Remove Duplicate records operator in order to avoid any duplicate record in our datasets. Then in all the column apply the normalized rules. The rule is given bellow

$$X_{normalize} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (3.1)$$

3.3 STOCK MARKET ANALYSIS

Stock market analysis is divided into two parts – fundamental analysis and technical analysis.

3.3.1 Fundamental Analysis

Fundamental Analysts are concerned with the company that underlies the stock itself. They evaluate a company's past performance as well as the credibility of its accounts. Many performance ratios are created that aid the fundamental analyst with assessing the validity of a stock.

Fundamental analysis in stock market is finding out the true value of a stock, which then can be compared with the value it is being traded with on stock markets and therefore finding out whether the stock on the market is undervalued or not [17]. Finding out the true value can be done by various methods with basically the same principle. The principle being that a company is worth all of its future profits added together. These future profits also have to be discounted to their present value. This principle goes along well with the theory that a business is all about profits and nothing else [18].

Contrary to technical analysis, fundamental analysis is thought of more as a long-term strategy. Fundamental analysis is built on the belief that human society needs capital to make progress and if a company operates well, it should be rewarded with additional capital and result in a surge in stock price. Fundamental analysis is widely used by fund managers as it is the most reasonable, objective and made from publicly available information like financial statement analysis [19]. Another meaning of fundamental analysis is beyond bottom-up company analysis, it refers to top-down analysis from first analyzing the global economy, followed by country analysis and then sector analysis, and finally the company level analysis.

3.3.2 Technical Analysis

Technical analysts or chartists are not concerned with any of the company's fundamentals. They seek to determine the future price of a stock based solely on the trends of the past price. Alongside the patterns, techniques are used such as the exponential moving average (EMA), oscillators, support and resistance levels or momentum and volume indicators. Candle stick patterns, believed to have been first developed by Japanese rice merchants, are nowadays widely used by technical analysts [20].

Technical analysis is rather used for short-term strategies, than the long-term ones. And therefore, it is far more prevalent in commodities and forex markets where traders focus on short-term price movements. There are some basic assumptions used in this analysis, first being that everything significant about a company is already priced into the stock, other being that the price moves in trends and lastly that history (of prices) tends to repeat itself which is mainly because of the market psychology [21].

3.4 ARTIFICIAL NEURAL NETWORK

Artificial neural networks were developed with inspiration from the human brain's structure. The human brain contains neurons, synapses, and electrical signals communicated over those synapses. An artificial neural network has similar components with nodes, connections, and weights representing the human brain's neurons, synapses, and strengths of electrical signals, respectively. Humans are able to logically deduce, artistically express themselves, and possess consciousness because of the brain’s structure. An artificial neural network implements that structure, and there are three different layers within a neural network: the input layer, the hidden layer, and the output layer. An artificial neural network can take on many forms where connections, loops, and multiple directions of connections are permitted, but only one type of neural network is presented within this paper. Do not be mistaken, while they are similar, artificial neural networks do not resemble human’s brains in size or abstract reasoning. Google is working on creating a human brain sized artificial neural network. Although empirical observation has proven that neural networks do not need extensive computational power for successful domain specific regression.

In an Artificial Neural Network, each input from the input layer is feed up to each node in the hidden layer, and from there to each node on the output layer. There can be any number of nodes per layer and there are usually multiple hidden layers to pass through before ultimately reaching the output layer. It is important to choose the right number of nodes and layers when optimizing the neural network. As per the diagram, it's called a feed-forward network because of how the signals are passed through the layers of the neural network in a single direction. There are many others neural network present. There are also feedback networks where its architecture allows signals to travel in both directions.

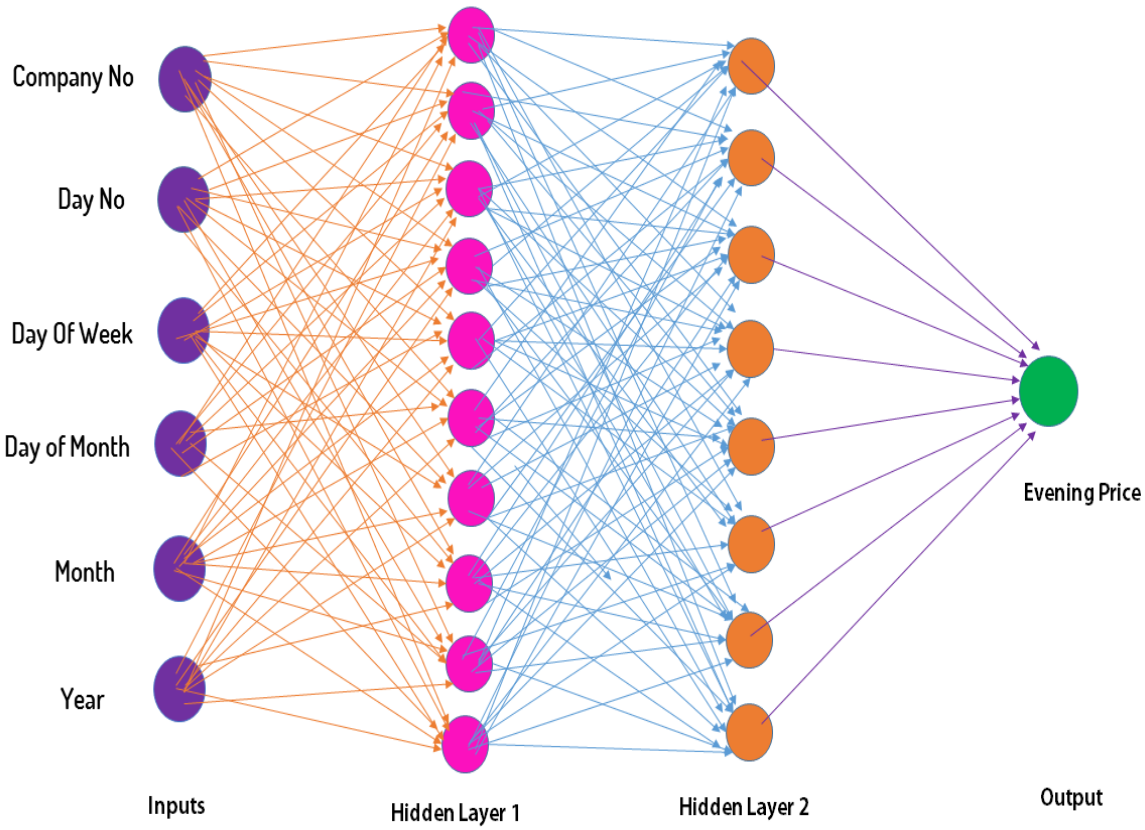

Figure 3.1: Artificial neural network

The input layer can be thought of as the “sensor organ” of the ANN. It is the place where user set the parameters of the environment (i.e. the information ANN to make a decision about). The neurons in this layer have no incoming connections, since their values are set from an external source. The outgoing connections send these values to the neurons of the next layer in the hierarchy. Normally "hidden" layers are present in between input and output layers. It is called hidden because they are invisible to any external processes that interact with the ANN. The neurons in these layers have both incoming connections from the preceding layer and outgoing connections to the succeeding layer, and work just as described earlier in this section. The hidden layers can be thought of as the "cognitive brain" of the network. The output layer is the final layer where the end result of the computations of the ANN. If the input layer holds the parameters of a problem, the information can be interpreted. The neurons in this layer have no outgoing connections, because their φ-values are read directly by whatever external process is using the network.

Neural networks are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the connections between elements largely determine the network function. You can train a neural network to perform a particular function by adjusting the values of the connections (weights) between elements.

Typically, neural networks are adjusted, or trained, so that a particular input leads to a specific target output. The next figure illustrates such a situation. There, the network is adjusted, based on a comparison of the output and the target, until the network output matches the target. Typically, many such input/target pairs are needed to train a network.

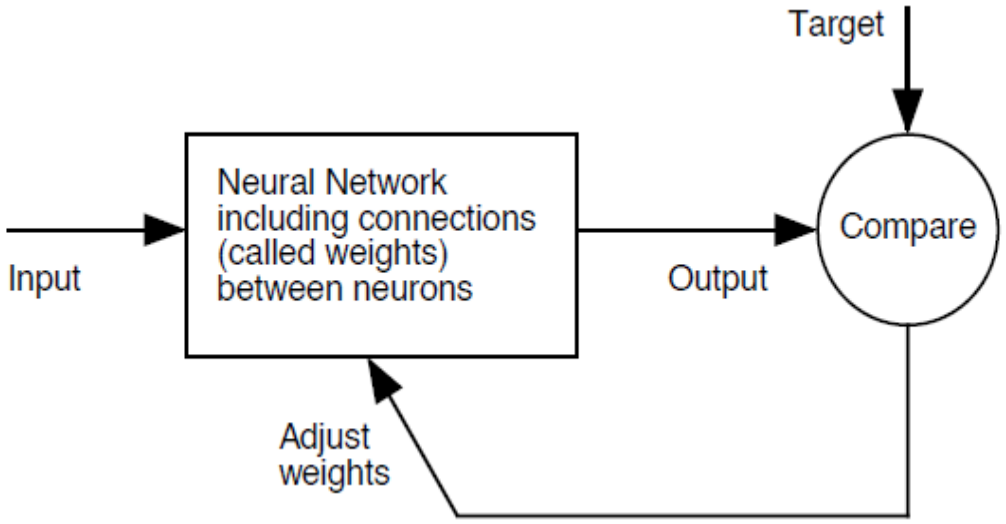

Figure 3.2: Neural network function

3.5 MULTILAYER NEURAL NETWORK

An artificial neural network with feedforward topology is called a feedforward neural network (FFNN) and is the most popular and widely-used ANN topology. The only condition for such a network is that the information must flow in only one direction, from input to output, with no connections such as cycles or loops. A FFNN consists of layers of interconnected neurons where each neuron in the input layer is connected to all neurons in the next layer, and each neuron in this layer is connected to all neurons in the next layer and so on until finally they are connected to the final output layer.

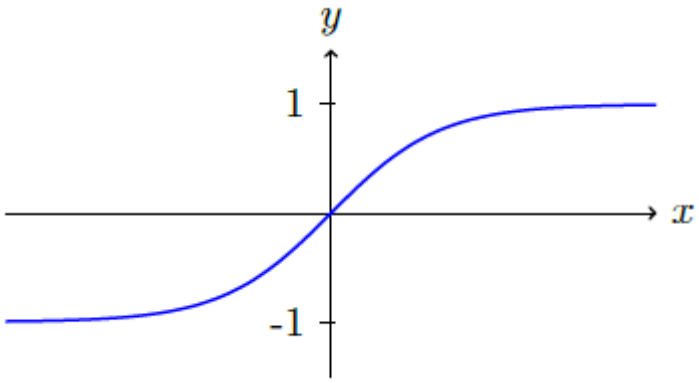

Figure 3.3: Sigmoid function

There are no limitations on the number of layers, increasing the number of layers does however increase the complexity of the network. All layers between the input and output layers are called the hidden layers, simply meaning that they are neither input nor output neurons. The figure below illustrates a multilayer FFNN.

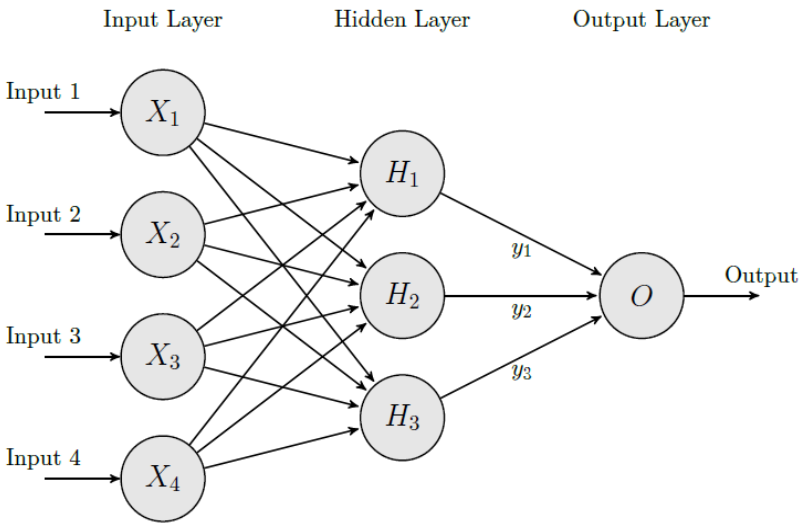

Figure 3.4: Architecture of Feedforward neural network

The input layer is made up of neurons X_1 to X_4, the inputs to the network. At the neurons in the hidden layer, denoted H_1 to H_3 in Figure 3.4, each neuron processes its input using the principles of Equation 3.2 and 3.3 to calculate this layer's outputs, as seen in Equation 3.2.

$$y_j = f(\sum_{i=1}^{4} w_{ij} x_i + b_j) \tag{3.2}$$

where, *f* is a continuous activation function, b_j correspond to the bias at H_j and w_{ij} denotes the weight assigned to input *xi* at neuron H_j . This layer's outputs, y_j, are then fed forward to the output neuron *O*, where the network output *o* is computed as in Equation 3.3.

$$O = g(\sum_{i=1}^{3} w_j y_j + b_O \tag{3.3}$$

Here w_j denotes the weight in O corresponding to the hidden neuron output y_j, b_O denotes the bias at the output neuron, and *g* denotes a linear activation function.

The popularity of FFNNs derives from the fact that they have been successfully applied for the purpose of function approximation. Meaning that they are able to approximate any continuous function arbitrarily well given the right number of neurons in the hidden layer. For this reason FFNNs are applicable in situations with unknown nonlinear relationships, making it suitable for the stock market [22].

Feed-Forward Backpropagation Model:

In order to use an ANN it has to train and learn how to handle the input data. This learning process corresponds to setting the values of weights and biases so that it minimizes the error between the outputs and expected values. There are different training algorithms that can be used for the ANN depending on, for instance, the topology and activation function. The backpropagation algorithm is mainly used in FFNNs together with a sigmoid function because of its easily computed and continuous derivative [23]. Also, since data is only fed forward in an FFNN, errors are propagated backwards while adjusting the weights and biases according to the error. Finally, for practical reasons, FFNNs implementing the backpropagation algorithm do not have many layers because of the time for training the network grows exponentially.

The backpropagation algorithm uses the concept of supervised learning with error correction to train the network. This means that given a sample of inputs and a target (desired output) to the network, the error is calculated between the target and actual output [24]. The idea of the backpropagation algorithm is to obtain the target as output when the respective inputs are given. Since the error is the difference between the target and output

the error depends on the weights and biases in the neurons. Therefore the weights and biases are adjusted in order to minimize the error. The change to the weights and biases is decided based on their impact on the error, how this is achieved is out of scope for this report. More specifically, the error is calculated using the mean square error function as seen in equation (3.4).

$$MSE = (t_i - t_o)^2 \tag{3.4}$$

The mean square error between the network output *oi* and the target *ti* will always be positive. The error will also be greater if the difference is big, and lesser if the difference is small. The total error of the network will simply be the sum of the errors of all the neurons in the output layer as seen in equation (3.5).

$$MSE_N = \sum_{i=1}(t_i - t_o)^2 \tag{3.5}$$

3.6 DISTINCT ALGORITHM OF MULTILAYER NEURAL NETWORK

In multilayer neural network there are many algorithm have been used to train the network such as Levenberg Marquardt, Bayesian Regularization, Gradient Descent, Quasi Newton, Scaled Conjugate Gradient, Resilient Backpropagation Fletcher-Powell Conjugate Gradient etc. But all of these are not given good result. In our experiment Levenberg Marquardt, Bayesian Regularization, Scaled Conjugate Gradient, Quasi Newton are given good result.

So, for train the network we can use four algorithm, these are

i. Levenberg Marquardt
ii. Bayesian Regularization
iii. Scaled Conjugate Gradient
iv. Quasi Newton

3.6.1 Levenberg Marquardt Algorithm

This algorithm typically require more memory but less time. Training automatically stop when generalization stops improving as indicated by an increase in the mean square error of the validation sample.

The Levenberg-Marquardt algorithm, also known as the damped least-squares method, has been designed to work specifically with loss functions, which take the form of a sum of squared errors. It works without computing the exact Hessian matrix. Instead, it works with the gradient vector and the Jacobian matrix.

Consider a loss function which can be expressed as a sum of squared errors of the form

$$f = \sum_{i=1}^{m} e_i^2 \tag{3.6}$$

here m is the number of instances in the data set.

We can define the Jacobian matrix of the loss function as that containing the derivatives of the errors concerning the parameters,

$$J_{i,j} = \frac{\partial_{e_i}}{\partial_{e_j}} \tag{3.7}$$

for i=1,…,m and j=1,…,n.

where m is the number of instances in the data set, and n is the number of parameters in the neural network. Note that the size of the Jacobian matrix is m·n.

The gradient vector of the loss function can be computed as:

$$\nabla f = 2J^T.e \tag{3.8}$$

here e is the vector of all error terms.

Finally, we can approximate the Hessian matrix with the following expression.

$$\mathrm{H}f \approx 2J^T.J + \lambda \mathrm{I} \tag{3.9}$$

Where λ is a damping factor that ensures the positiveness of the Hessian and I is the identity matrix.

The next expression defines the parameters improvement process with the Levenberg-Marquardt algorithm

$$w^{(i+1)} = w^i \tag{3.10}$$

$$-(J^{(i)T}.J^{(i)} + \lambda^{i} I)^{-1}.2J^{(i)T}.e^{(i)} \quad (3.11)$$

for i =0, 1…….

When the damping parameter λ is zero, this is just Newton's method, using the approximate hessian matrix. On the other hand, when λ is large, this becomes gradient descent with a small training rate.

The parameter λ is initialized to be large so that the first updates are small steps in the gradient descent direction. If any iteration happens to result in a fail, then λ is increased by some factor. Otherwise, as the loss decreases, λ is decreased so that the Levenberg Marquardt algorithm approaches the Newton method. This process typically accelerates the convergence to the minimum. .

As we have seen, the Levenberg Marquardt algorithm is a method tailored for functions of the type sum-of-squared-error. That makes it to be very fast when training neural networks measured on that kind of errors.

Code:

```
%% This script assumes these variables are defined:
x=inputs;
x=sample_inputs;
t=target;
t=sample_target;
trainFcn='trainlm'; % for Levenberg-Marquardt algorithm
% Create a Fitting Network
hiddenLayerSize = [10,8];
% feedforward network, with a sigmoid transfer function in the
% transfer function in the hidden layer,
net = fitnet(hiddenLayerSize,trainFcn);
% Configuration to examining the input and target data
net1 = configure(net,x,t);
% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
```

```
net.divideParam.testRatio = 15/100;
% Training occurs according to trainlm training parameters, shown
% here with their default values;you can change these values
net.trainparam.epochs=1000;
net.trainparam.goal=0;
net.trainParam.max_fail=6;
net.trainParam.min_grad=1e-7;
net.trainParam.mu_max=1e10;
net.trainParam.showWindow=true;
% If you wanted to change the transfer function to logsig, for example,
% To view the layerWeights subobject for the weight between layer 1 and layer 2,
net.layerWeights{2,1}
%% Train the Network
[net,tr] = train(net,x,t);
% View the Network
view(net);
% Test the Network or to calculate the network response to any input.
outputs=net(x);
errors = gsubtract(t,outputs);
e=t-outputs;
performance = perform(net,t,outputs)
%% Plots
% Uncomment these lines to enable various plots.
figure, plotperform(tr); % or figure, plotperf(tr)
figure, plottrainstate(tr);
figure, ploterrhist(errors);
figure, plotregression(t,outputs,'Regression');
% or to generates multiple plots.
Outputs = net(inputs);
% Train and Apply Multilayer Neural Networks to calculate the network response to
any input
y= net(Sample_inputs);
T=Sample_target;
```

```
e=T-y;
%% If the network is not sufficiently accurate, you can try
% initializing the network and the training again.
%net = init(net);
%net = train(net,inputs,targets);
```

3.6.2 Bayesian Regularization Algorithm

BR is a training algorithm that updates the weights and bias values according to LM optimization. It minimizes a combination of squared errors and weights, and then determines the correct combination so as to produce a network that generalizes well. BR introduces network weights into the training objective function which is denoted as F(ω) in (3.12).

$$F(\omega) = \alpha E_{\omega} + \beta E_{D} \tag{3.12}$$

Where, E_{ω} is the sum of the squared network weights and E_D is the sum of network errors. Both α and β are the objective function parameters. In the BR framework, the weights of the network are viewed as random variables, and then the distribution of the network weights and training set are considered as Gaussian distribution.

The α and β factors are defined using the Bayes' theorem. The Bayes' theorem relates two variables (or events), A and B, based on their prior (or marginal) probabilities and posterior (or conditional) probabilities as in (3.13)

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \tag{3.13}$$

where P(A|B) is the posterior probability of A conditional on B, P(B|A) the prior of B conditional on A, and P(B) the non-zero prior probability of event B, which functions as a normalizing constant. In order to find the optimal weight space, objective function (2.16) needs to be minimized, which is the equivalent of maximizing the posterior probability function given as in (3.14)

$$P(\alpha,\beta|D,M)=\frac{P(D|\alpha,\beta,M)P(\alpha,\beta|M)}{P(D|M)} \quad (3.14)$$

where α and β are the factors needed be to optimized, D is the weight distribution, M is the particular neural network architecture, $P(D|M)$ is the normalization factor, $P(\alpha,\beta|M)$ is the uniform prior density for the regularization parameters and $P(D|\alpha,\beta,M)$ is the likelihood function of D for given α,β,M. Maximizing the posterior function $(\alpha|D, M)$ is equivalent of maximizing the likelihood function $(D|\alpha,,)$. As a result of this process, optimum values for α and β for a given weight space are found. Afterwards, algorithm moves into LM phase where Hessian matrix calculations take place and updates the weight space in order to minimize the objective function.

Then, if the convergence is not met, algorithm estimates new values for α and β and the whole procedure repeats itself until convergence is reached.

Code:

```
%% This script assumes these variables are defined:
x=inputs;
x=sample_inputs;
t=target;
t=sample_target;
trainFcn='trainbr'; % for Bayesian Regularization algorithm
% Create a Fitting Network
hiddenLayerSize = [10];
% feedforward network, with a sigmoid transfer function in the
% transfer function in the hidden layer,
net = fitnet(hiddenLayerSize,trainFcn);
% Configuration to examining the input and target data
net1 = configure(net,x,t);
% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Training occurs according to trainlm training parameters, shown
```

```
% here with their default values;you can change these values
net.trainparam.epochs=1000;
net.trainparam.goal=0;
net.trainParam.max_fail=6;
net.trainParam.min_grad=1e-7;
net.trainParam.mu_max=1e10;
net.trainParam.showWindow=true;
% If you wanted to change the transfer function to logsig, for example,
% To view the layerWeights subobject for the weight between layer 1 and layer 2,
net.layerWeights{2,1}
%% Train the Network
[net,tr] = train(net,x,t);
% View the Network
view(net);
% Test the Network or to calculate the network response to any input.
outputs=net(x);
errors = gsubtract(t,outputs);
e=t-outputs;
performance = perform(net,t,outputs)
%% Plots
% Uncomment these lines to enable various plots.
figure, plotperform(tr); % or figure, plotperf(tr)
figure, plottrainstate(tr);
figure, ploterrhist(errors);
figure, plotregression(t,outputs,'Regression');
% or to generates multiple plots.
Outputs = net(inputs);
% Train and Apply Multilayer Neural Networks to calculate the network response to
any input
y= net(Sample_inputs);
T=Sample_target;
e=T-y;
%% If the network is not sufficiently accurate, you can try
```

% initializing the network and the training again.
%net = init(net);
%net = train(net,inputs,targets);

3.6.3 Scaled Conjugate Gradient Algorithm

The basic backpropagation algorithm adjusts the weights in the steepest descent direction, i.e., the most negative of the gradient. This is the direction in which the performance function is decreasing most rapidly. It turns out that, although the function decreases most rapidly along the negative of the gradient, this does not necessarily produce the fastest convergence. In the conjugate gradient algorithms a search is performed along such a direction which produces generally faster convergence than the steepest descent direction, while preserving the error minimization achieved in all previous steps. This direction is called the conjugate direction. In most of the CG algorithms the step-size is adjusted at each iteration. A search is made along the conjugate gradient direction to determine the step size, which will minimize the performance function along that line.

In the conjugate gradient training algorithm, the search is performed along with conjugate directions, which produce generally faster convergence than gradient descent directions. These training directions are conjugated concerning the Hessian matrix.
Let denote d the training direction vector. Then, starting with an initial parameter vector w(0) and an initial training direction vector $d^0 = g^0$, the conjugate gradient method constructs a sequence of training directions as:

$$d^{(i+1)} = g^{(i+1)} + d^i . \gamma^i \tag{3.15}$$

for i=0,1,…
Here γ is called the conjugate parameter, and there are different ways to calculate it. Two of the most used are due to Fletcher and Reeves and Polak and Ribiere. For all conjugate gradient algorithms, the training direction is periodically reset to the negative of the gradient.

$$w^{(i+1)} = w^{(i+1)} + d^i . \sigma^i \quad (3.16)$$

for i =0,1,…

This method has proved to be more effective than gradient descent in training neural networks. Since it does not require the Hessian matrix, the conjugate gradient is also recommended when we have vast neural networks.

Code:

%% This script assumes these variables are defined:

x=inputs;

x=sample_inputs;

t=target;

t=sample_target;

trainFcn='trainscg'; for Scaled Conjugate Gradient algorithm

% Create a Fitting Network

hiddenLayerSize = [10];

% feedforward network, with a sigmoid transfer function in the

% transfer function in the hidden layer,

net = fitnet(hiddenLayerSize,trainFcn);

% Configuration to examining the input and target data

net1 = configure(net,x,t);

% Set up Division of Data for Training, Validation, Testing

net.divideParam.trainRatio = 70/100;

net.divideParam.valRatio = 15/100;

net.divideParam.testRatio = 15/100;

% Training occurs according to trainlm training parameters, shown

% here with their default values;you can change these values

net.trainparam.epochs=1000;

net.trainparam.goal=0;

net.trainParam.max_fail=6;

net.trainParam.min_grad=1e-7;

net.trainParam.mu_max=1e10;

```
net.trainParam.showWindow=true;
% If you wanted to change the transfer function to logsig, for example,
% To view the layerWeights subobject for the weight between layer 1 and layer 2,
%% Train the Network
[net,tr] = train(net,x,t);
% View the Network
view(net);
% Test the Network or to calculate the network response to any input.
outputs=net(x);
errors = gsubtract(t,outputs);
e=t-outputs;
performance = perform(net,t,outputs)
%% Plots
% Uncomment these lines to enable various plots.
figure, plotperform(tr); % or figure, plotperf(tr)
figure, plottrainstate(tr);
figure, ploterrhist(errors);
figure, plotregression(t,outputs,'Regression');
Outputs = net(inputs);
% Train and Apply Multilayer Neural Networks to calculate the network response to
any input
y= net(Sample_inputs);
T=Sample_target;
e=T-y;
%% If the network is not sufficiently accurate, you can try
% initializing the network and the training again.
%net = init(net);
%net = train(net,inputs,targets);
```

3.6.4 Quasi Newton Algorithm

The application of Newton's method is computationally expensive since it requires many operations to evaluate the Hessian matrix and compute its inverse.

Alternative approaches, known as quasi-Newton or variable metric methods, are developed to solve that drawback. These methods, instead of calculating the Hessian directly, and then evaluating its inverse, build up an approximation to the inverse Hessian at each iteration of the algorithm. This approximation is computed using only information on the first derivatives of the loss function.

The Hessian matrix is composed of the second partial derivatives of the loss function. The main idea behind the quasi-Newton method is approximating the inverse Hessian by another matrix G, using only the first partial derivatives of the loss function. Then, the quasi-Newton formula can be expressed as:

$$W^{i+1} = W^{i} - (G^{i}.g^{i}.\eta^{i}) \tag{3.17}$$

for i=0,1,…

The training rate η can either be set to a fixed value or found by line minimization. The inverse Hessian approximation G has different flavors.

Code:

```
%% This script assumes these variables are defined:
x=inputs;
x=sample_inputs;
t=target;
t=sample_target;
trainFcn='trainbfg';for bfgs quasi newton algorithm
% Create a Fitting Network
hiddenLayerSize = [10];
% feedforward network, with a sigmoid transfer function in the
% transfer function in the hidden layer,
net = fitnet(hiddenLayerSize,trainFcn);
% Configuration to examining the input and target data
net1 = configure(net,x,t);
% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
```

```
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Training occurs according to trainlm training parameters, shown
% here with their default values;you can change these values
% To view the layerWeights subobject for the weight between layer 1 and layer 2,
%% Train the Network
[net,tr] = train(net,x,t);
% View the Network
view(net);
% Test the Network or to calculate the network response to any input.
outputs=net(x);
errors = gsubtract(t,outputs);
e=t-outputs;
performance = perform(net,t,outputs)
%% Plots
% Uncomment these lines to enable various plots.
figure, plotperform(tr); % or figure, plotperf(tr)
figure, plottrainstate(tr);
figure, ploterrhist(errors);
figure, plotregression(t,outputs,'Regression');
% or to generates multiple plots.
Outputs = net(inputs);
% Train and Apply Multilayer Neural Networks to calculate the network response to
any input
y= net(Sample_inputs);
T=Sample_target;
e=T-y;
%% If the network is not sufficiently accurate, you can try
% initializing the network and the training again.
%net = init(net);
%net = train(net,inputs,targets);
```

3.7 TRANING AND TESTING

The main problem in predicting share market is that the share market is a chaos system. There are many variables that could affect the share market directly or indirectly. There are no significant relations between the variables and the price. We cannot draw any mathematical relation among the variables. There are no laws of predicting the share price using these variables. For this kind of chaotic system the neural network approach is suitable because we do not have to understand the solution. This is a major advantage of neural network approaches. On the other hand in the traditional techniques we must understand the inputs, the algorithms and the outputs in great detail. With the neural network we just need to simply show the correct output for the given inputs. With sufficient amount of training, the network will mimic the function. Another advantage of neural network is that during the tanning process, the network will learn to ignore any inputs that don't contribute to the output. In our purposed system, there is a training phase where some parameters named weights are found from this section and Backpropagation Algorithm is used for this training phase. These weights are used in prediction phase using same equations which are used in training phase.

The ANN was design and implemented by MATLAB which is a programming language used for mathematical computations. The implementation included construction, training, testing and validation of the ANN.

1. At first data is inserting in the network considered input and target layer then load these data. After inserting the data of input layer and target layer we can define the variable for these input and output data. Six data has been considered as input these are company no., day no, day of week, day of month, month no. and year. Different number of output layer has been considered but using one output layer got best accuracy. Six input and one output layer is selected for this network.

2. After defining input and output variable then we had to select the training algorithm. For training this network, four training algorithms have been used these are Levenberge Marquardt, Bayesian Regularization, Scaled Conjugate Gradient and Quasi Newton algorithm.

3. This step define the hidden layer of the network. One hidden layer with 10 neuron, two hidden layer with 10 and 8 neuron, two hidden layer with 5 and 5 neuron and two hidden layer with 10 and 5 neuron have been used. From two hidden layer with 10 and 8 neuron we got batter output. Anyone can use different number of hidden layer in this network design then configure the network. In this research different number of hidden layer have been used these are 1 layer with 10 neuron, 2 layer with 10 and 8 neuron but 2 layer with 10 and 8 neuron given best result.

4. In this step, the input and target data are randomly divided into three sets for training, testing and validation. In this designed network various percentage of training, testing and validation dataset have been used, such as 70% or 80% or 90% for training, 15% or 10% or 5% will be used to validate that the network is generalizing and to stop training before overfitting. Then last 15% or 10% or 5% will be used as a completely independent test of network generalization.

5. At last train the network and view the network and save the output value for check the accuracy.

Table 3.2: Change hidden layer with hidden neuron and training data

S.N.	Hidden Layer	Neuron	Training Data	Testing Data	Validation Data
01	1	10	70%	15%	15%
02	1	10	80%	10%	10%
03	1	10	90%	5%	5%
04	2	5 and 5	70%	15%	15%
05	2	10 and 5	70%	15%	15%
06	2	10 and 5	80%	10%	10%
07	2	10 and 8	70%	15%	15%
08	2	10 and 8	80%	10%	10%

In table 3.2 show the different combination of changing the hidden layer with hidden neuron and training, testing and validation data to train the network.

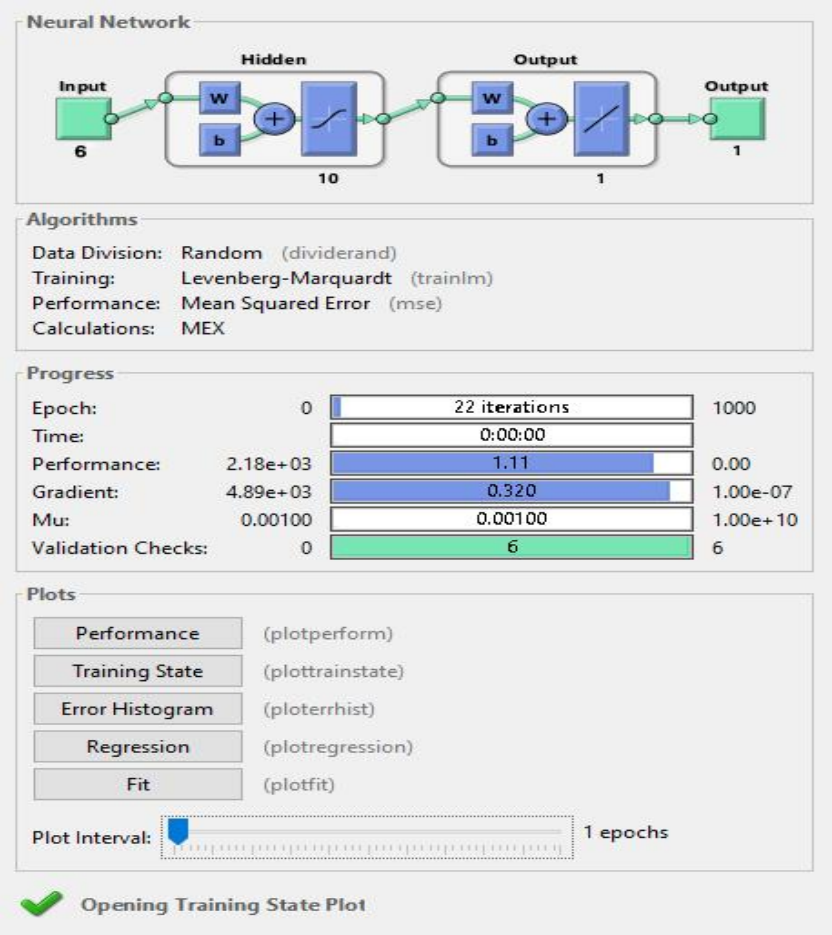

Figure 3.5: After train the network

3.8 PERFORMANCE ANALYSIS

To evaluate the performance of the ANN, different sets of metrics can be used and compared such as regression analysis, error histogram, mean square error and accuracy testing

Regression Analysis:

Regression analysis is a set of statistical methods used for the estimation of relationships between a dependent variable and one or more independent variables. An independent variable is an input, assumption, or driver that is changed in order to assess its impact on a dependent variable

$$Y_i = f(X_i, \beta) + e_i \tag{3.18}$$

where,

Y_i = dependent variable

f = function

X_i = independent variable

β = unknown parameters

e_i = error terms

Error Histogram:

Error histogram is the histogram of the errors between target values and predicted values after training a feedforward neural network. As these error values indicates how predicted values are differing from the target values, hence these can be negative. Bins are the number of vertical bars you are observing on the graph.

Mean Square Error:

In statistics, the mean squared error *(*MSE*)* of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors that is, the average squared difference between the estimated values and the actual value.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{3.19}$$

where

MSE = mean square error

n = number of data point

Y_i = observed value

$\hat{Y}_i$ = predicted value

Accuracy Testing:

The main goal of this thesis is to predict stock price so we can test the accuracy. To find the accuracy from the predicted value this formula have been used

$$Accuracy = \frac{\text{Target value} - \text{Predicted value}}{\text{Target Value}} \times 100 \tag{3.20}$$

3.9 ACCURACY TESTING

In accuracy testing, target value and predicted value are compared. For accuracy testing some sample data are trained those are not trained previous.

Table 3.3: Sample data for accuracy testing

Company Name	Day No	Day of Week	Day of Month	Month	Year	Evening Price
ICBS	54	5	4	8	16	6.3
DBHF	12	1	16	1	20	109.8
GP	19	1	19	1	20	263.3
LANKBNG	142	4	5	8	15	14.78
SQUARE	99	5	28	5	15	163.49
NAVANA	14	4	20	1	16	49.6
IDLC	29	4	11	2	15	47.66
PRAN	29	2	29	1	19	250.4
LANKBNG	119	2	3	7	19	19.62
CITY BANK	55	1	24	2	19	15.81
SQUARE	45	3	14	2	17	218.24
RUPALI LIFE	229	2	12	12	16	34.13
DBHF	177	3	26	19	16	105
ICBS	27	1	29	11	15	5.7
CITY BANK	200	5	19	7	18	27.62
CITY BANK	277	5	4	10	18	31.05
DBHF	85	3	9	5	19	123
IFIC	147	2	2	8	17	17.77
IPDC	62	4	30	3	16	19
SQUARE	84	2	25	3	19	251.78
SQUARE	100	1	31	5	15	161.06
IFIC	45	5	5	3	15	20
RUPALI LIFE	162	2	27	8	17	37.4
DBHF	165	3	16	9	19	120.5
NAVANA	135	4	29	7	19	45.2

3.10 SUMMARY

In this chapter, the methods and process of Artificial Neural Network for the prediction of financial stock market is discussed. Using this methodology techniques, this thesis work will be experimented. The results and discussion chapter are based on those above methods.

CHAPTER 4

RESULT AND DISCUSSION

4.1 INTRODUCTION

The general research associated with predicting how the stock market will perform, is one of the most difficult things to do. The common trend towards the stock market among the society that it is highly risky for investment or not suitable for trade so most of the people are not even interested. To solve these types of problems the financial market stock price prediction will be the best for predicting the future values. This research mainly focus on to predict evening price of stock market. For training the network we can use three algorithm and estimation the evening price.

4.2 LEVENBERGE MARQUARDT ALGORITHM

After train the neural network using Levenberg Marquardt algorithm, we can get regression, error histogram and mean square error that measure the performance.

4.2.1 With two hidden layer (6-5-5-1) and 70% training data

Regression plot which shows the relationship between outputs of the network and the targets. The plots representation training, validation, testing data. The dashes line in each plot represent the perfect result. The solid line represent the best fit linear regression line between outputs and targets. R=1 indicates that there is an exact linear relationship between outputs and targets. If R is close to zero, then there is no linear relationship between output and targets.

Figure 4.1 shows the training data indicates a good fit. Training regression value is 0.97677, testing regression value is 0.9781 and validation regression value is 0.9761.

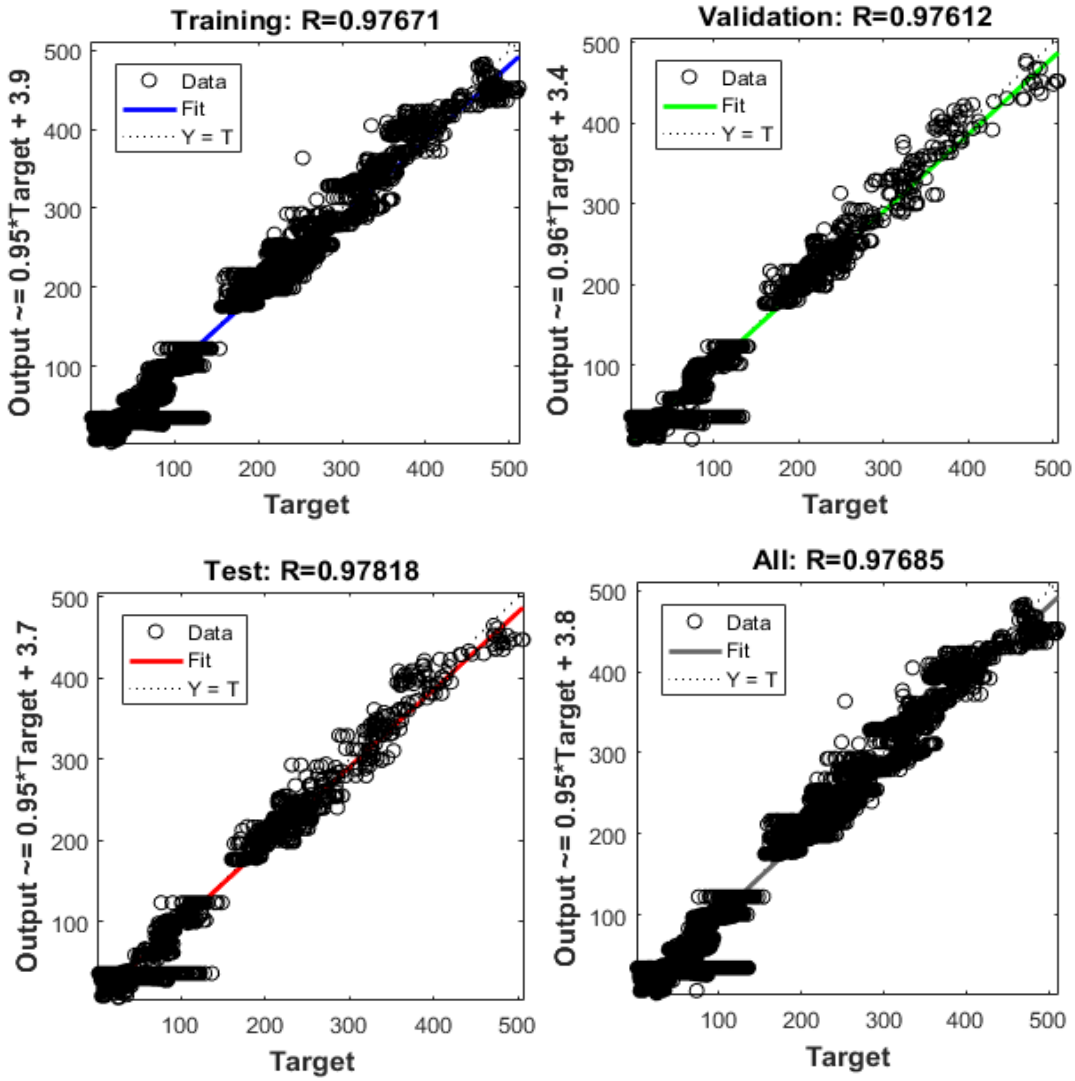

Figure 4.1: Regression value of two hidden layer (6-5-5-1) and 70% training data

Error histogram is the histogram of the errors between target values and predicted values after training a feedforward neural network. As these error values indicates how predicted values are differing from the target values, hence these can be negative.

The blue bar represent training data, the green bars represent validation data and red bars represent testing data. The histogram can give an indication of outlier which are data point where the fit is significantly worse than the majority of data.

Bins are the number of vertical bars are observing on the graph. The total error range is divided into 20 smaller bins here. Y-axis represents the number of samples from dataset, which lies in a particular bin.

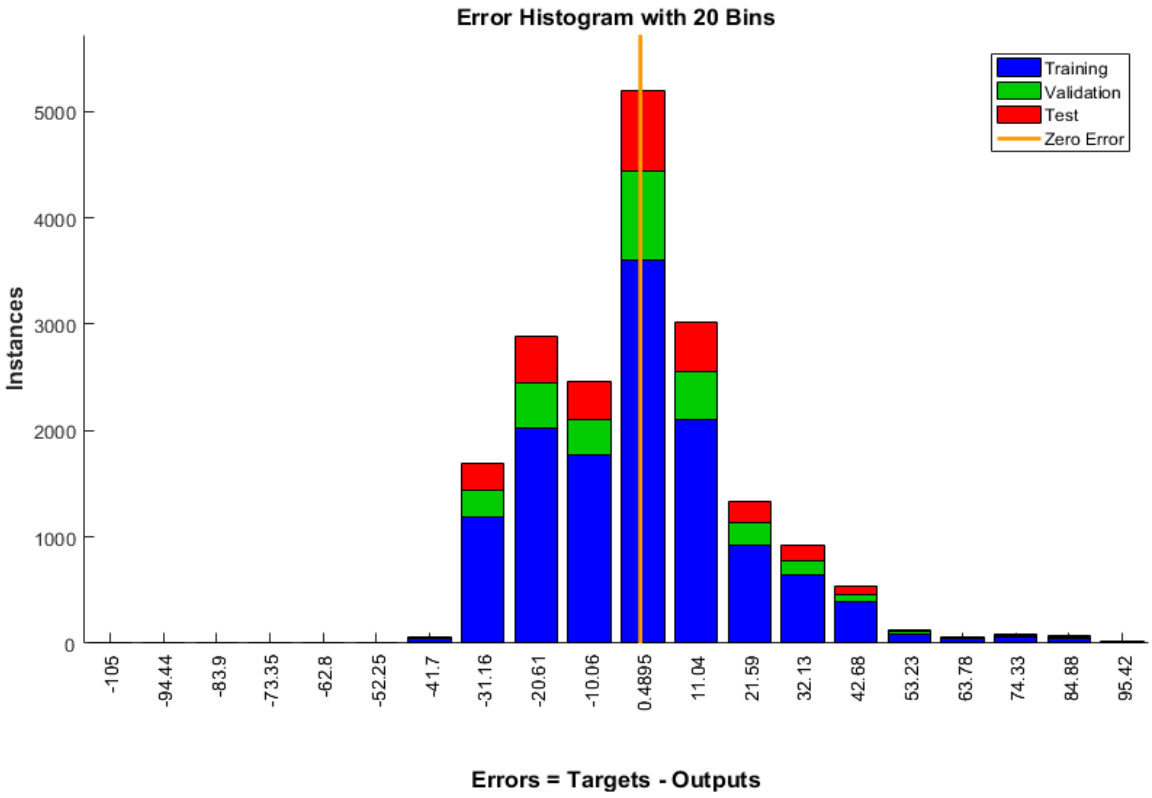

Figure 4.2: Error histogram of two hidden layer (6-5-5-1) and 70% training data

In figure 4.2, at the mid of this plot, have a bin corresponding to the error of 0.4895 and the height of that bin for training dataset lies but near to 4000 and validation and test dataset lies between 5000 and 6000. It means that many samples from different datasets have an error lies in that following range.

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the bin with center 0.4895. In figure 4.2, most errors fall between -31.16 and 32.13.

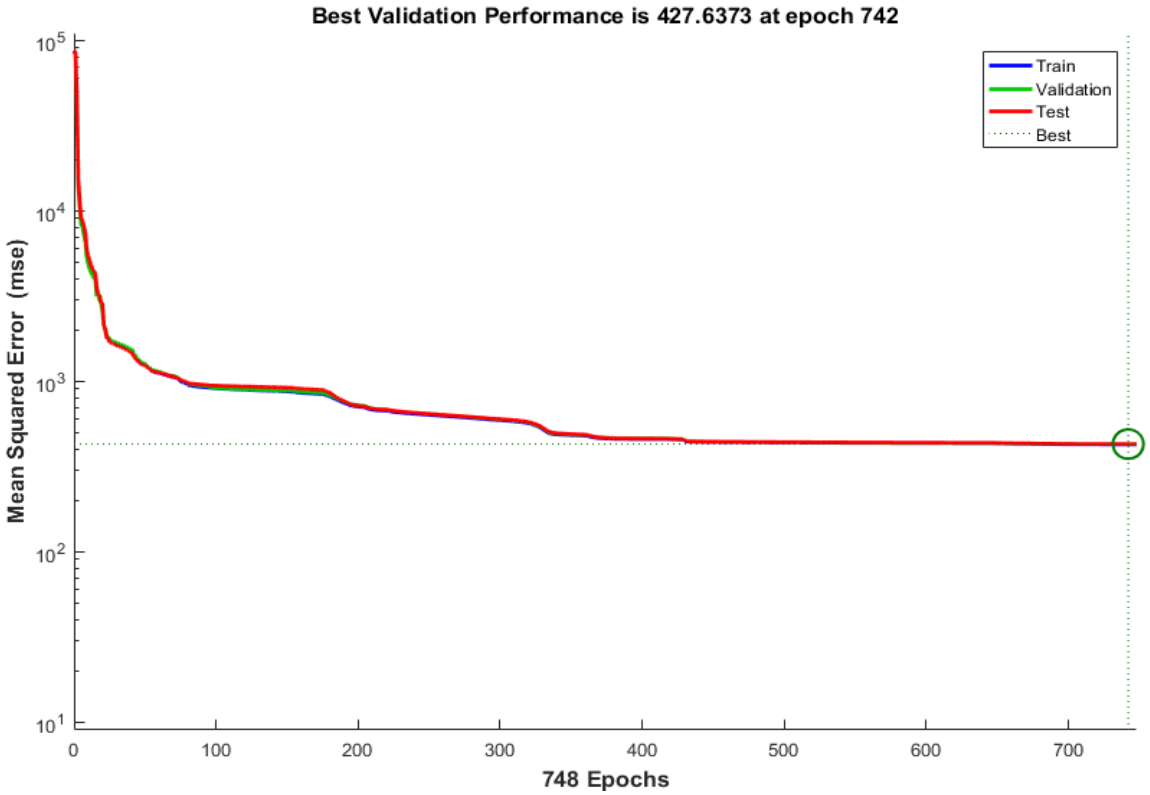

Figure 4.3: Epochs vs MSE of two hidden layer (6-5-5-1) and 70% training data

In figure 4.3, the training, testing and validation curves are very similar and no significant overfitting has occurred. 748 epochs indicates that the training stopped at iteration 748 and at iteration 742 where the best validation performance occurs and error is minimum that means mean square error is minimum.

4.2.2 With one hidden layer (6-10-1) and 70% training Data

Figure 4.4 shows the training data indicates a good fit. Training regression value is 0.9771, testing regression value is 0.9775 and validation regression value is 0.9768.

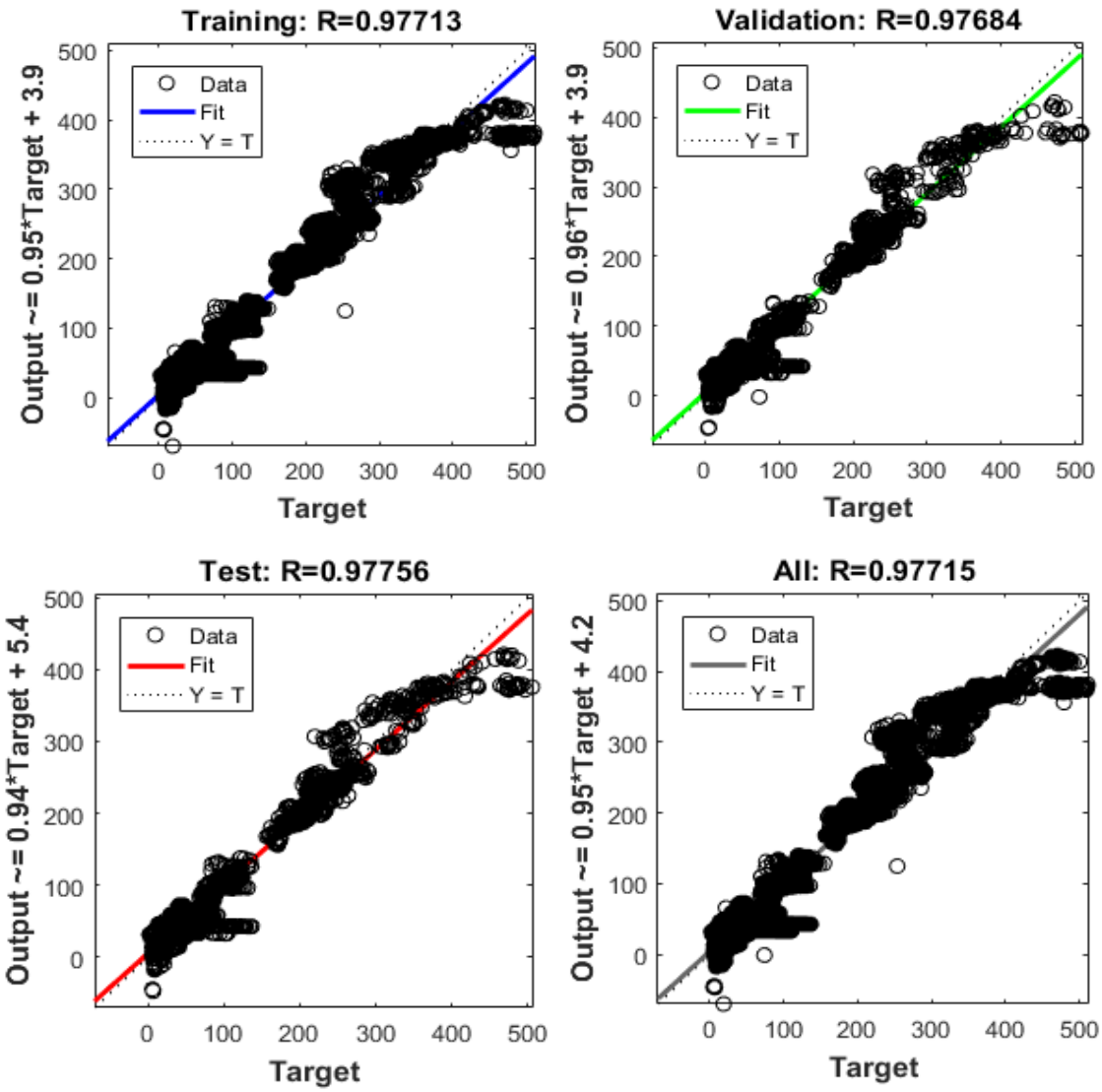

Figure 4.4: Regression value of one hidden layer (6-10-1) and 70% training data

In figure 4.5, at the mid of this plot, have a bin corresponding to the error of 5.07 and the height of that bin for training dataset lies but near to 4000 and validation and test dataset lies between 5000 and 6000. It means that many samples from different datasets have an error lies in that following range.

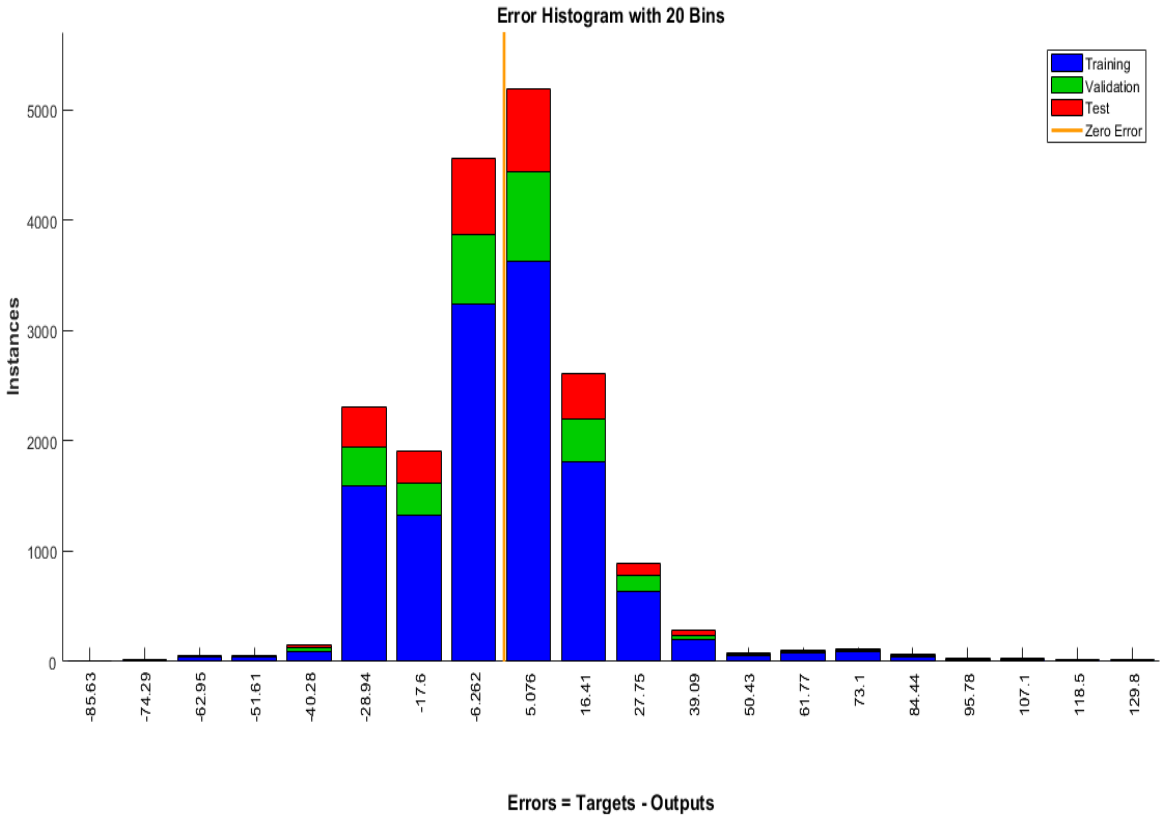

Figure 4.5: Error histogram of one hidden layer (6-10-1) and 70% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the bin with center between -6.26 and 5.07. In figure 4.5, most error fall between -28.94 and 27.75.

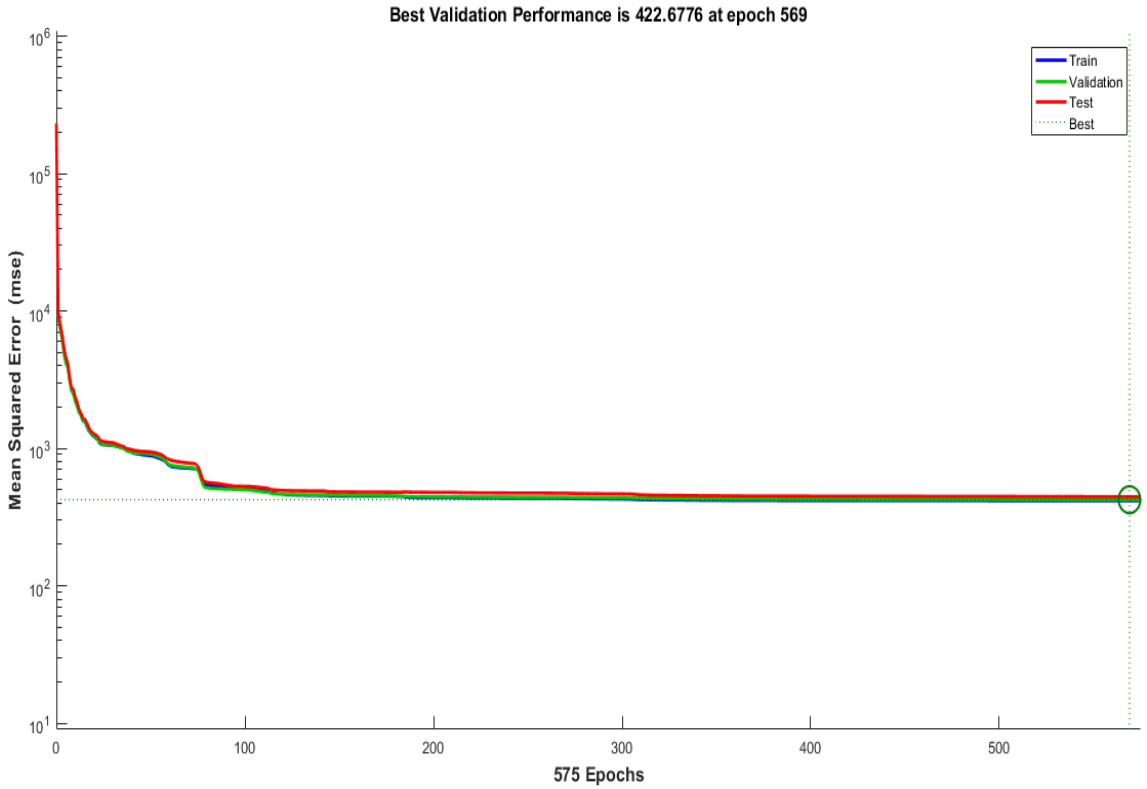

Figure 4.6: Epochs vs MSE of one hidden layer (6-10-1) and 70% training data

In figure 4.6, the training, testing and validation curves are very similar and no significant overfitting has occurred. 575 epochs indicates that the training stopped at iteration 575 and at iteration 569 where the best validation performance occurs and error is minimum.

4.2.3 With two hidden layer (6-10-5-1) and 70% training Data

Figure 4.7 shows the training data indicates a good fit. Training regression value is 0.9954, testing regression value is 0.9797 and validation regression value is 0.9947.

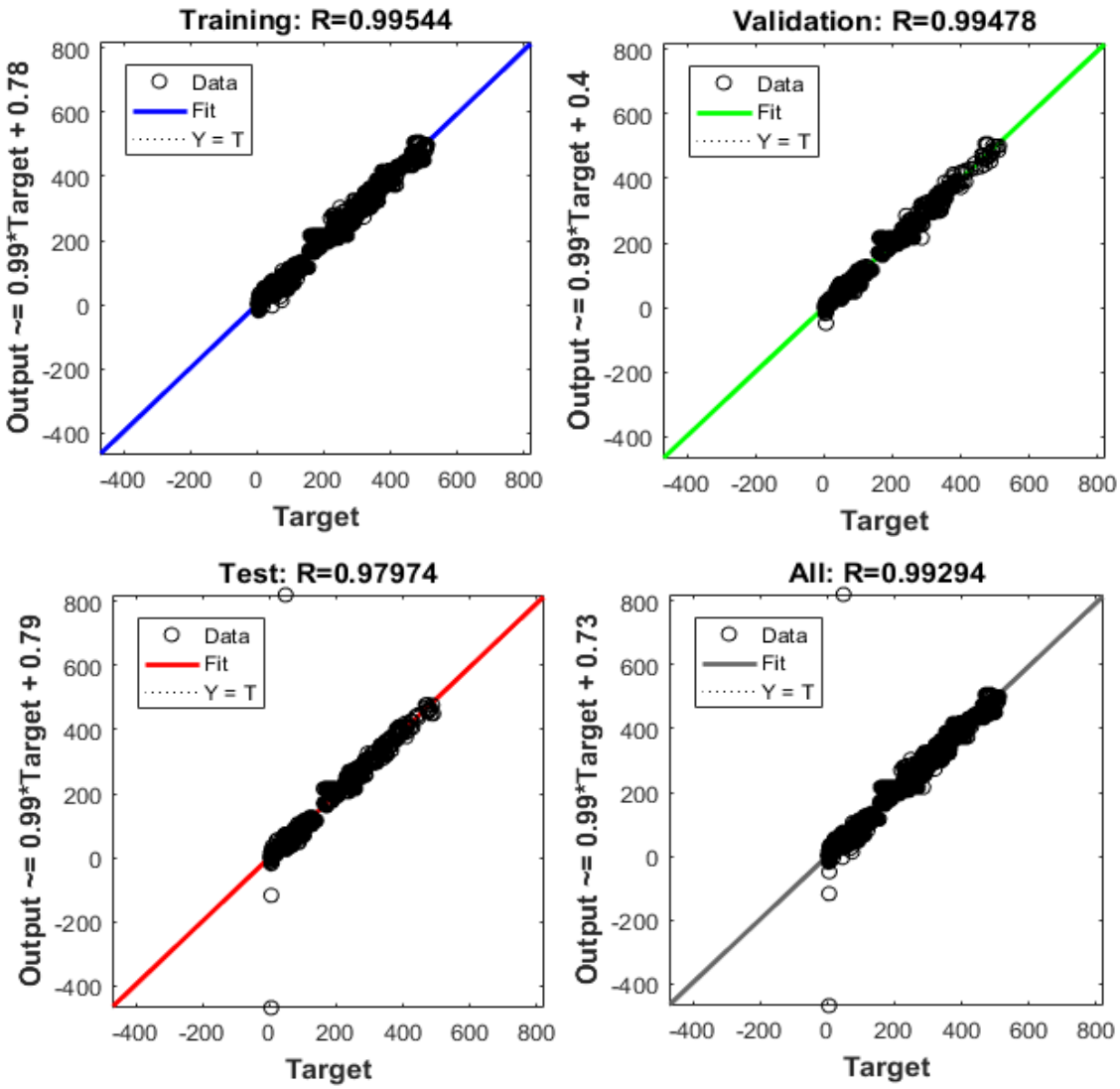

Figure 4.7: Regression value of two hidden layer (6-10-5-1) and 70% training data

In figure 4.8, at the mid of this plot, have a bin corresponding to the error of -4.96 and the height of that bin for training dataset lies but near to 8000 and validation and test dataset lies between 8000 and 10000.

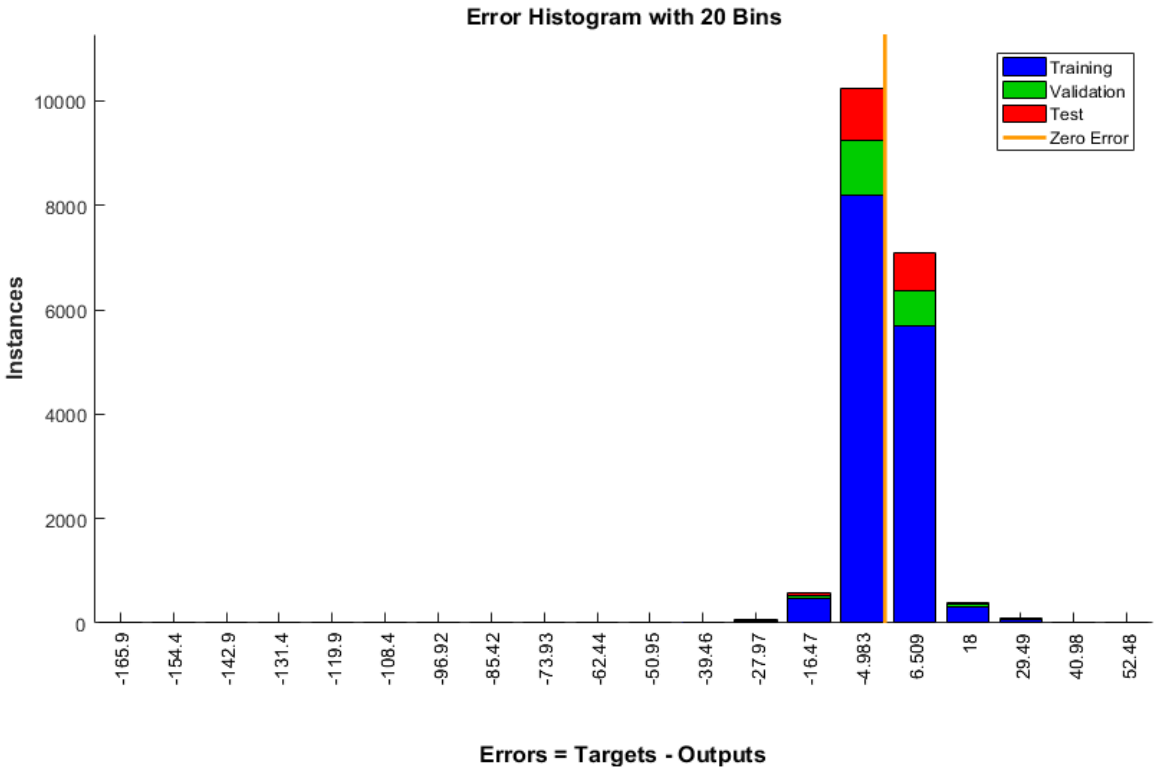

Figure 4.8: Error histogram of two hidden layer (6-10-5-1) and 70% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the bin -4.98. Figure 4.8 shows most errors fall between -4.978 and 6.50.

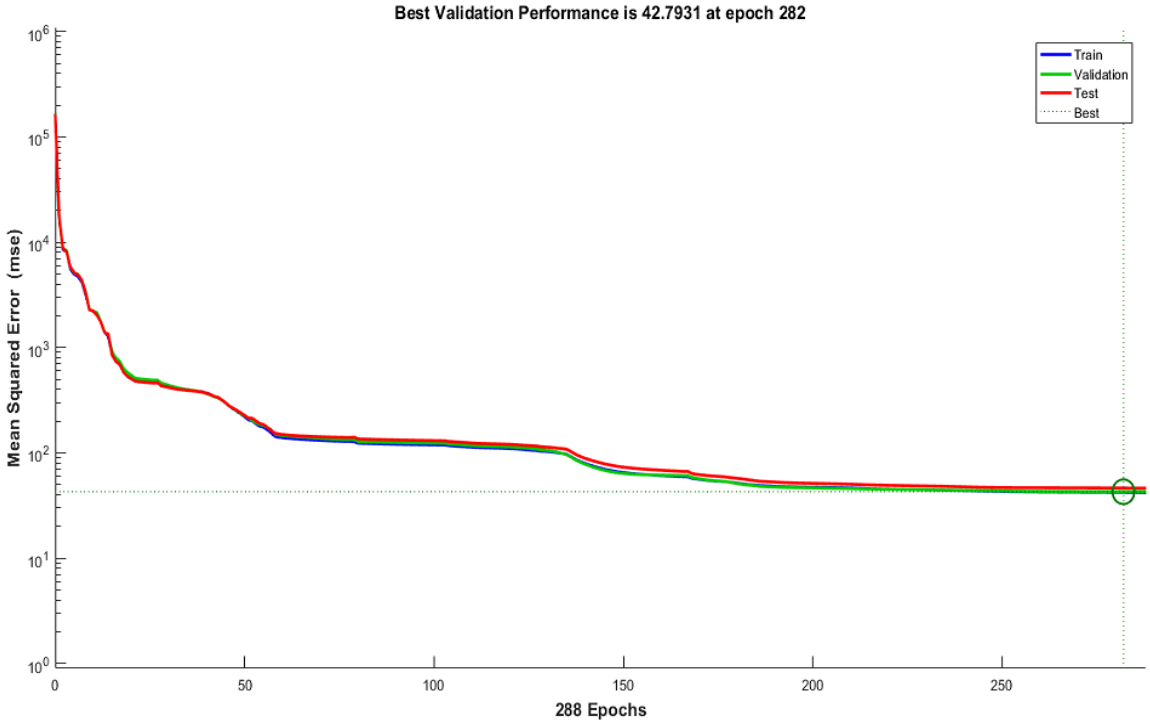

Figure 4.9: Epochs vs MSE of two hidden layer (6-10-5-1) and 70% training data

In figure 4.9, the training, testing and validation curves are very similar and no significant overfitting has occurred. 288 epochs indicates that the training stopped at iteration 288 and at iteration 282 where the best validation performance occurs and error is minimum.

4.2.4 With one hidden layer (6-10-1) and 90% training Data

Figure 4.10 shows the training data indicates a good fit. Training regression value is 0.9531, testing regression value is 0.9499 and validation regression value is 0.9539.

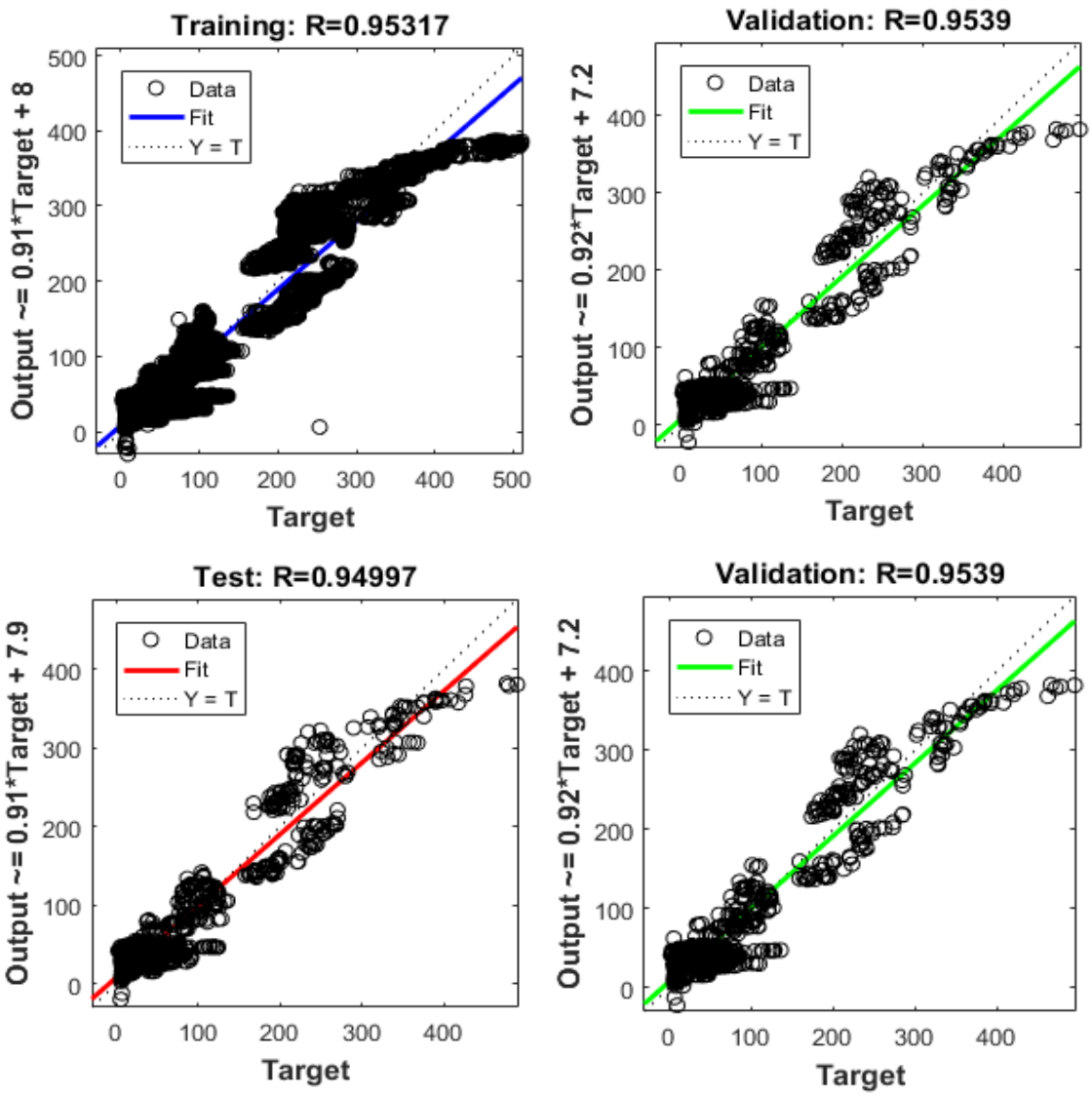

Figure 4.10: Regression value of two hidden layer (6-10-1) and 90% training data

In figure 4.11, at the mid of this plot, have a bin corresponding to the error of 1.031 and the height of that bin for training dataset lies but near to 4000 and validation and test dataset lies between 4500 and 5000.

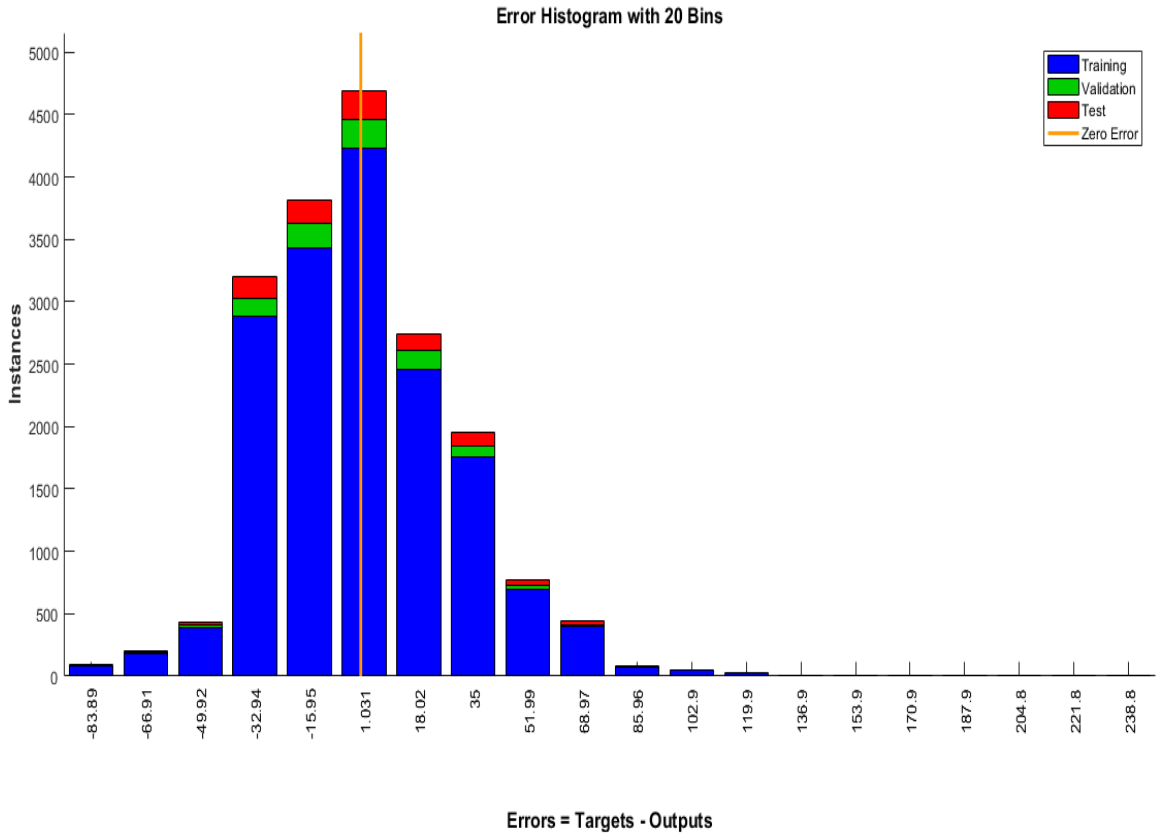

Figure 4.11: Error histogram of one hidden layer (6-10-1) and 90% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the bin with center 1.031. In this figure, most errors fall between -32.79 and 51.90.

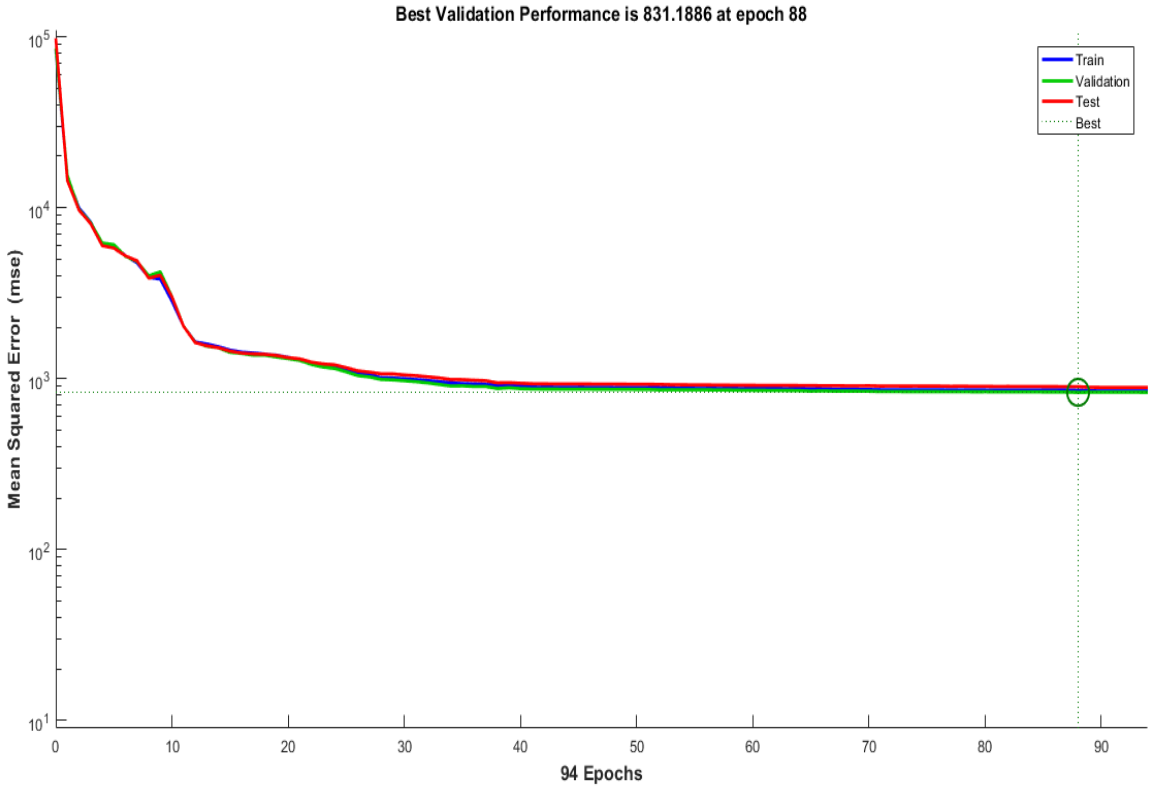

Figure 4.12: Epochs vs MSE of one hidden layer (6-10-1) and 90% training data

In figure 4.12, the training, testing and validation curves are very similar and no significant overfitting has occurred. 94 epochs indicates that the training stopped at iteration 96 and at iteration 88 where the best validation performance occurs and error is minimum.

4.2.5 With two hidden layer (6-10-8-1) and 70% training data

Figure 4.13 shows the training data indicates a good fit. Training regression value is 0.9977, testing regression value is 0.9975 and validation regression value is 0.9977.

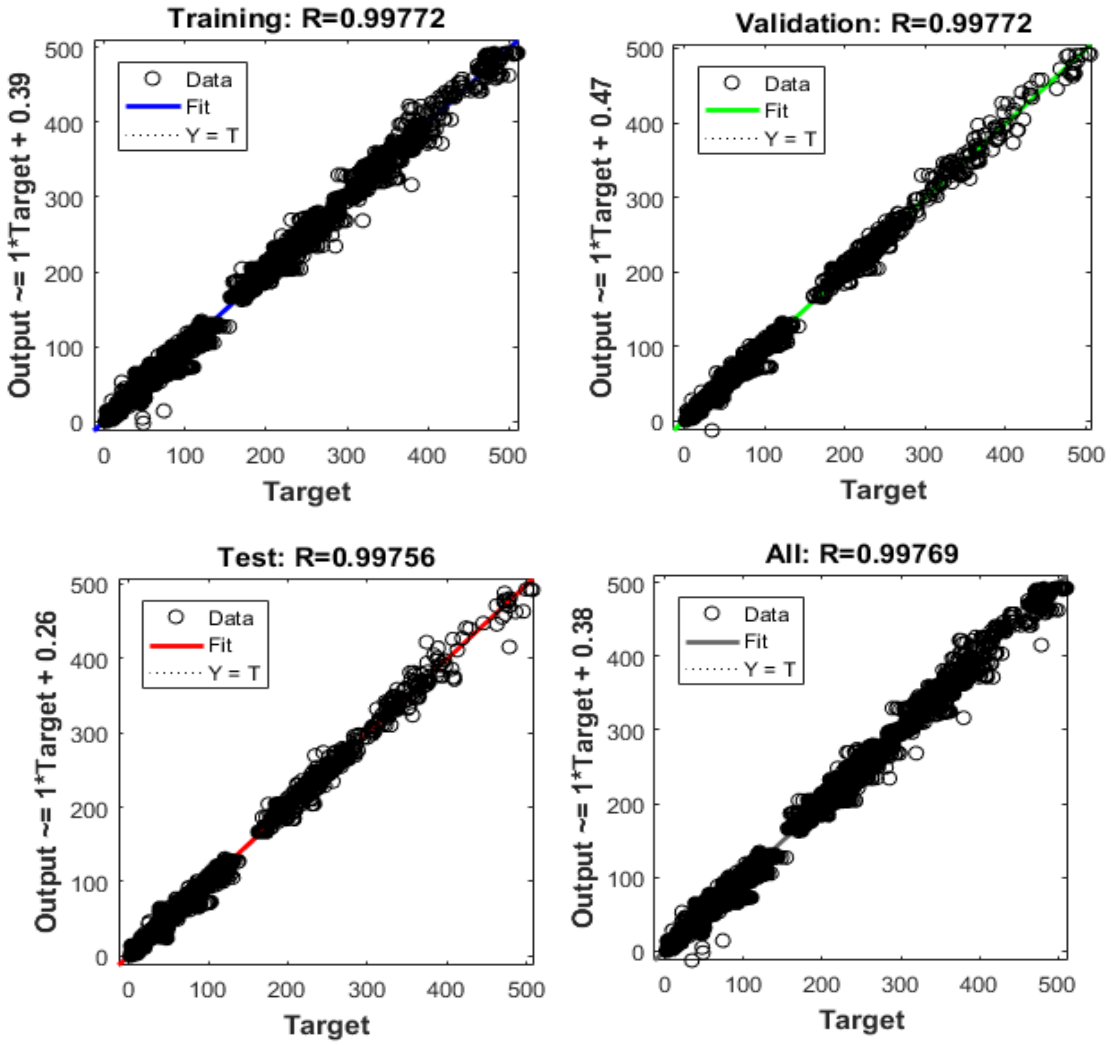

Figure 4.13: Regression value of two hidden layer (6-10-8-1) and 70% training data

In figure 4.14, at the mid of this plot, have a bin corresponding to the error of -0.263 and the height of that bin for training dataset lies but near to 6000 and validation and test dataset lies between 8000 and 9000.

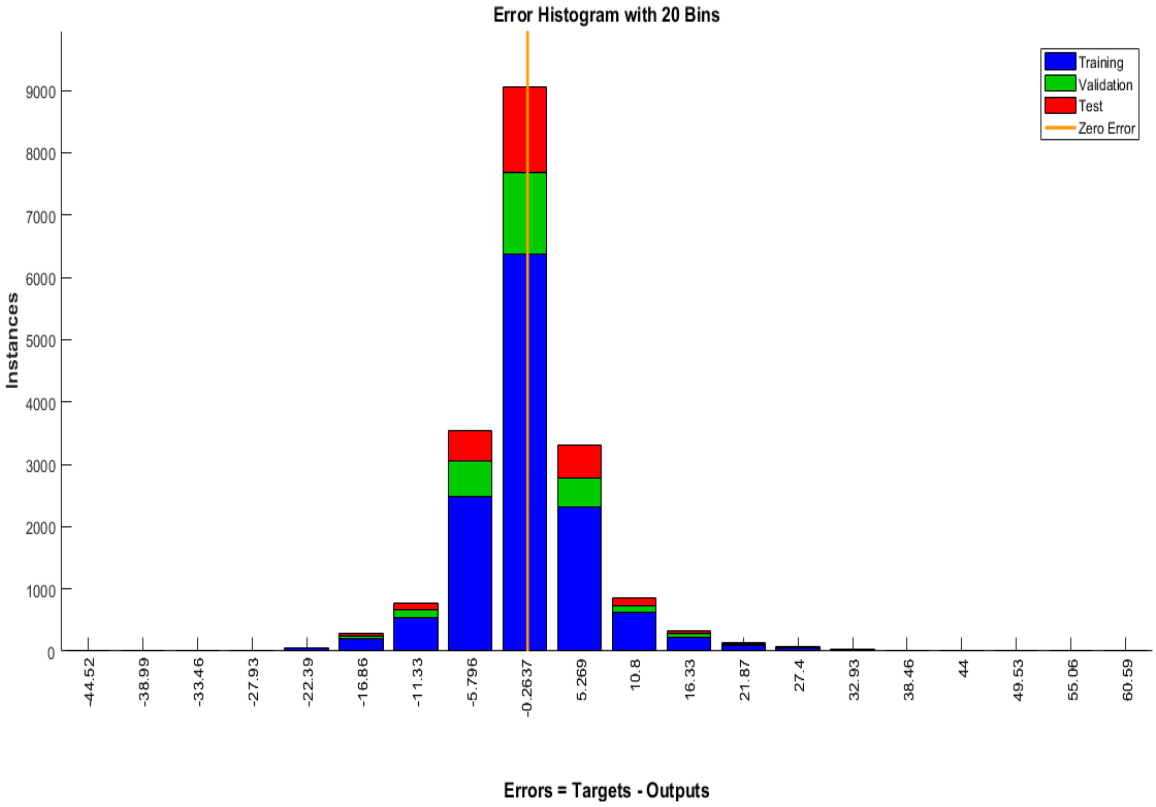

Figure 4.14: Error histogram of two hidden layer (6-10-8-1) and 70% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the bin with center -0.263. In figure 4.2, most errors fall between -5.79 and 5.26.

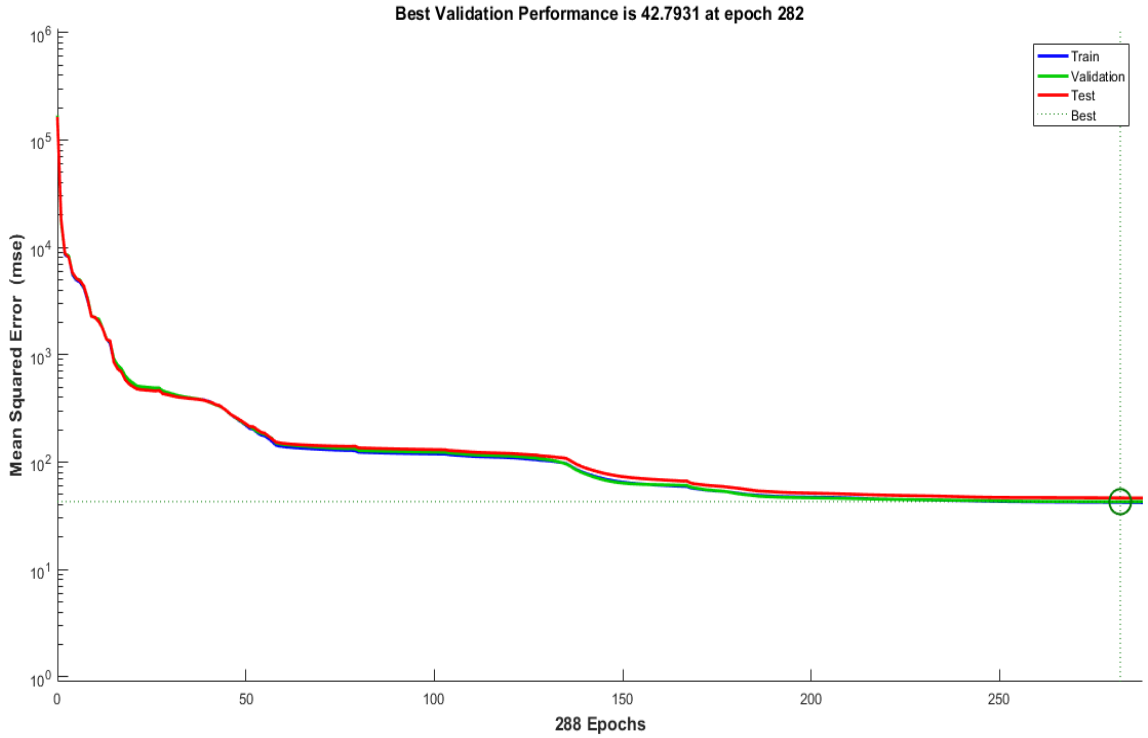

Figure 4.15: Epochs vs MSE of two hidden layer (6-10-8-1) and 70% training data

In figure 4.15, the training, testing and validation curves are very similar and no significant overfitting has occurred. 288 epochs indicates that the training stopped at iteration 288 and at iteration 282 where the best validation performance occurs and error is minimum.

Table 4.1: Summary of Levenberg Marquardt algorithm

Algorithm	Hidden Layer	Hidden Neuron	Training Data	Regression Value	Error Range
Levenberg Marquardt	2	5-5	70%	0.9768	-31.16 to 32.13
	1	10	70%	0.9771	-28.94 to 27.75
	1	10	90%	0.9510	-32.79 to 51.90
	2	10-8	70%	0.9976	-5.79 to 5.26
	2	10-5	70%	0.9929	-4.98 to 6.50

Table 4.1 shows that in Leveberg Marquardt algorithm, two hidden layer (6-10-8-1) and 70% training, 15% testing and 15% validation data given the higher regression value and lower error range.

Table 4.2: Error testing of Levenberg Marquardt algorithm

Sample No.	**Targets**	**Predicted**	**Error (%)**	**Predicted**	**Error (%)**	**Predicted**	**Error (%)**
		For 6-10-1 and 70% training data		6-10-5-1 and 70% training data		For 6-10-8-1 and 70% training data	
1	6.3	5.37	14.76	5.7	9.52	5.75	8.69
2	109.8	119.7	9.01	116.36	5.97	115.36	5.06
3	263.3	275.92	4.79	272.92	3.65	270.92	2.89
4	14.78	11.21	24.15	14.21	3.80	14.21	3.80
5	163.49	170.67	4.39	167.8	2.63	167.89	2.63
6	49.6	43.9	11.49	51.3	3.42	47.80	3.62
7	47.66	42.8	10.19	46.20	3.05	46.20	3.05
8	250.4	257.78	2.94	252.78	0.95	251.78	0.55
9	19.62	17.61	10.23	17.4	11.31	17.61	10.23
10	15.81	13.7	13.34	14.5	8.28	15.81	6.13
11	218.24	234.49	7.44	222.49	3.78	229.49	0.79
12	34.13	32.82	3.83	33.1	3.01	33.82	0.89
13	105	101.84	3.01	103.95	1	101.84	1.09
14	5.7	4.9	14.03	4.9	14.03	4.9	14.03
15	27.62	30.01	8.65	29.87	8.14	29.87	4.52
16	31.05	27.1	12.72	29	6.60	27.1	6.26
17	123	128.4	4.39	125.8	2.27	128.4	1.13
18	17.77	14.9	16.15	16.7	6.02	15.54	6.87
19	19	18.1	4.73	20.3	6.84	17.09	4.75
20	251.78	245.6	2.45	246.76	1.99	245.76	1.99
21	161.06	168.32	4.50	165.5	2.75	153.22	3.82
22	20	17.8	11	18.23	8.85	17.8	5.96
23	37.4	36.1	3.47	35.2	5.88	35.43	2.57
24	120.5	126.3	4.81	125.84	4.43	125.3	4.43
25	45.2	38.9	13.93	47.7	5.53	40.3	4.63
		Avg error	8.81	Avg error	5.27	Avg error	4.36

In table 4.4 shows the error and average error of Levenberg Marquardt training algorithm. Here targets price is actual value and predicted price is network predicted value. Error is calculated from targets and predicted value then average them. Table 4.1 shows the average error of Levenberg Marquardt with two hidden layer (6-10-8-1) and 70% training is 4.36, two hidden layer (6-10-5-1) and 70% training is 5.27 and one hidden layer (6-10-

1) and 70% training data is 8.81. This thesis experiments more with changing hidden layer and training data but this table shows only best three.

Table 4.3: Accuracy testing of Levenberg Marquardt algorithm

Hidden Layer with Neuron and Training data	Average Error (%)	Accuracy (%)
For 6-10-1 and 70% training data	8.81	91.19
For 6-10-5-1 and 70% training data	5.27	94.73
For 6-10-8-1 and 70% training data	4.36	95.64

From Table 4.2 we got the average error Table 4.3 shows the accuracy of Levenberg Marquardt algorithm, the accuracy of two hidden layer (6-10-5-1) with 70% training data is 94.73, one hidden layer (6-10-1) with 70% training data is 91.19, two hidden layer (6-10-8-1) with 70% training data is 95.64 and this is the best accuracy of Levenberg Marquardt training algorithm.

4.3 BAYESIAN REGULARIZATION

4.3.1 With one hidden layer (6-10-1) and 70% training Data

Figure 4.16 shows the training data indicates a good fit. Training regression value is 0.942 and testing regression value is 0.9409.

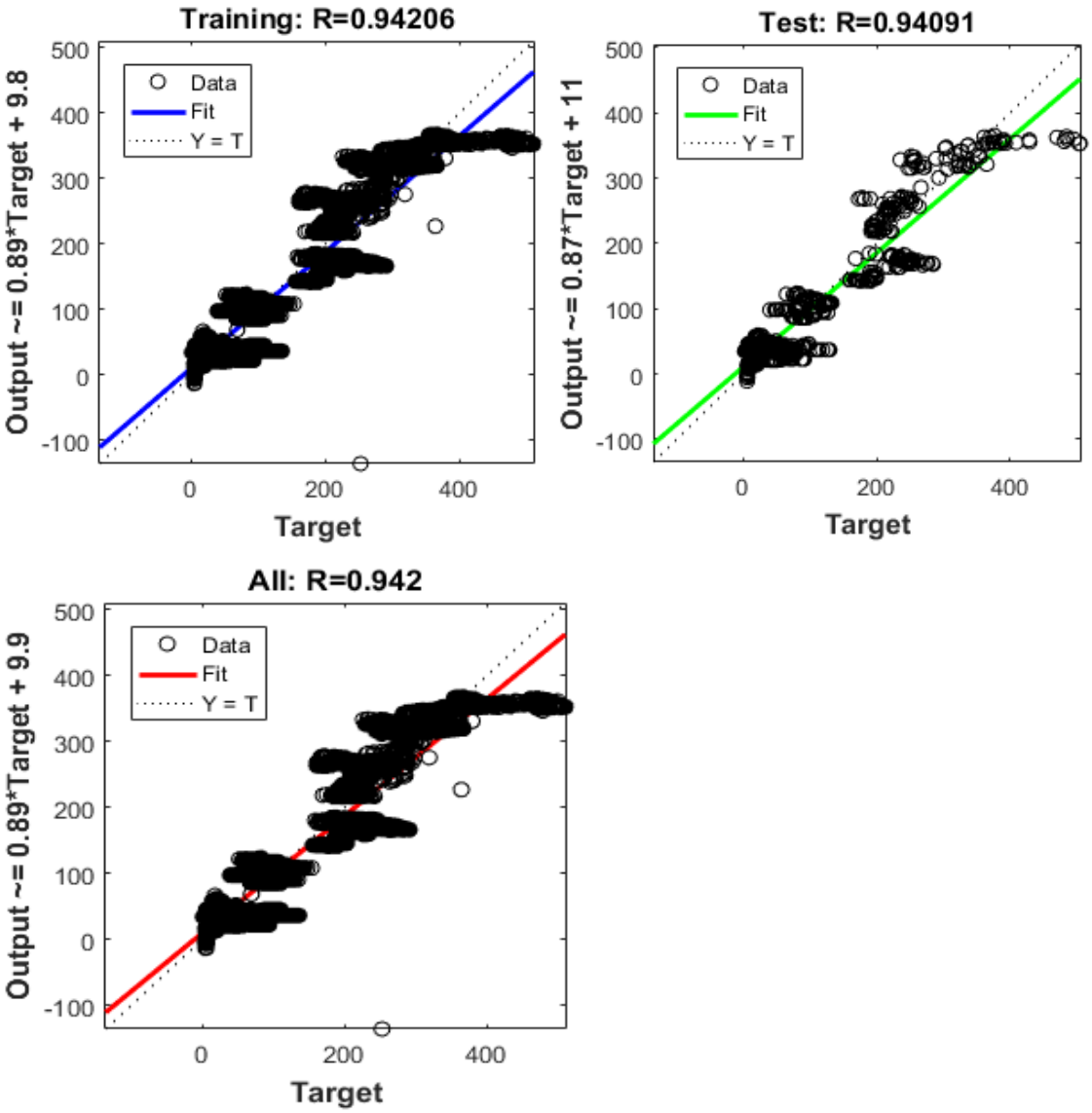

Figure 4.16: Regression value of one hidden layer (6-10-1) and 70% training data

In figure 4.17, at the mid of this plot, have a bin corresponding to the error of 2.18 and the height of that bin for training dataset lies above but near to 6000 and test dataset also lies above 6000. It means that many samples from different datasets have an error lies in that following range.

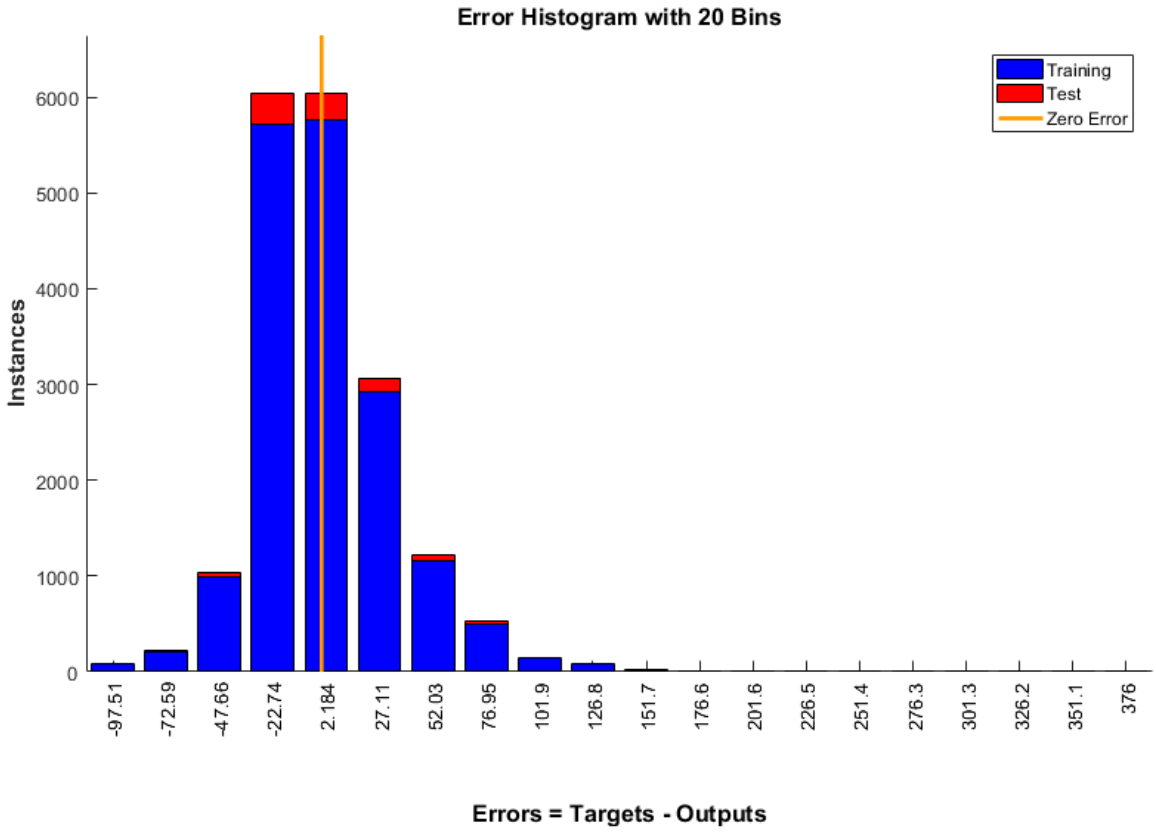

Figure 4.17: Error histogram of one hidden layer (6-10-1) and 70% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the bin with center 2.18. In figure 4.17, maximum errors fall between -47.66 and 52.03.

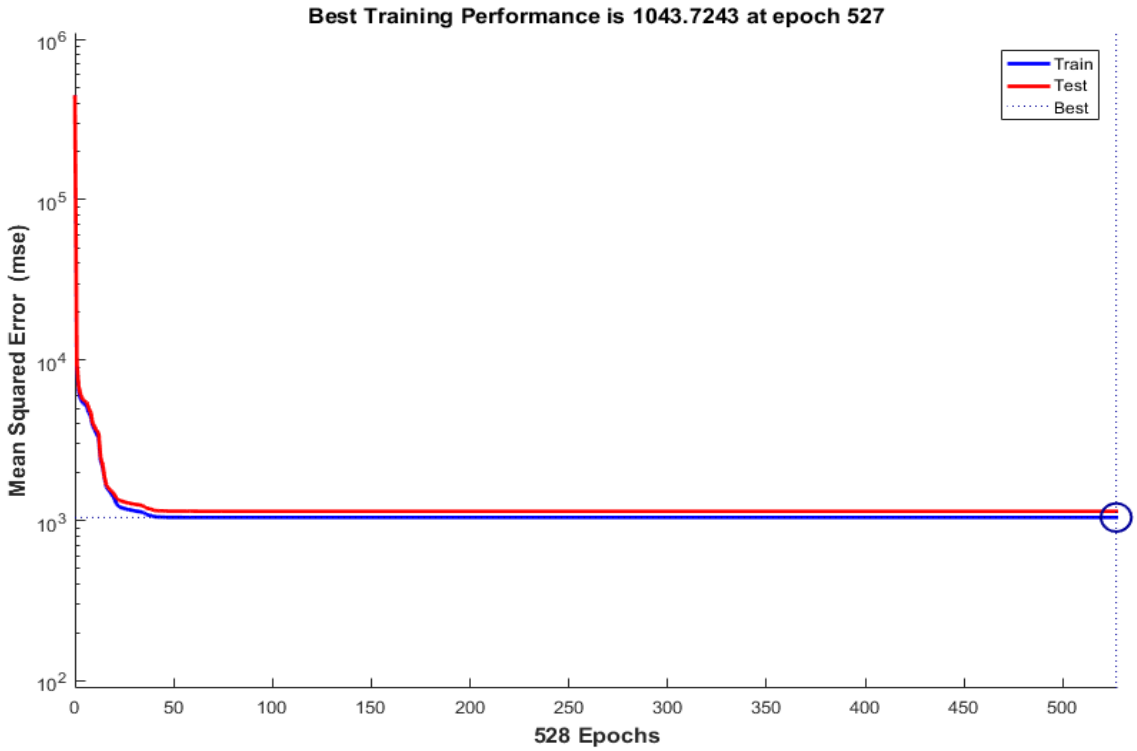

Figure 4.18: Epochs vs MSE of one hidden layer (6-10-1) and 70% training data

In figure 4.18, the training, testing and validation curves are very similar and no significant overfitting has occurred. 528 epochs indicates that the training stopped at iteration 528 and at iteration 527 where the best validation performance occurs and error is minimum.

4.3.2 With two hidden layer (6-10-5-1) and 70% training data

Figure 4.19 shows the training data indicates a good fit. Training regression value is 0.9629 and testing regression value is 0.957.

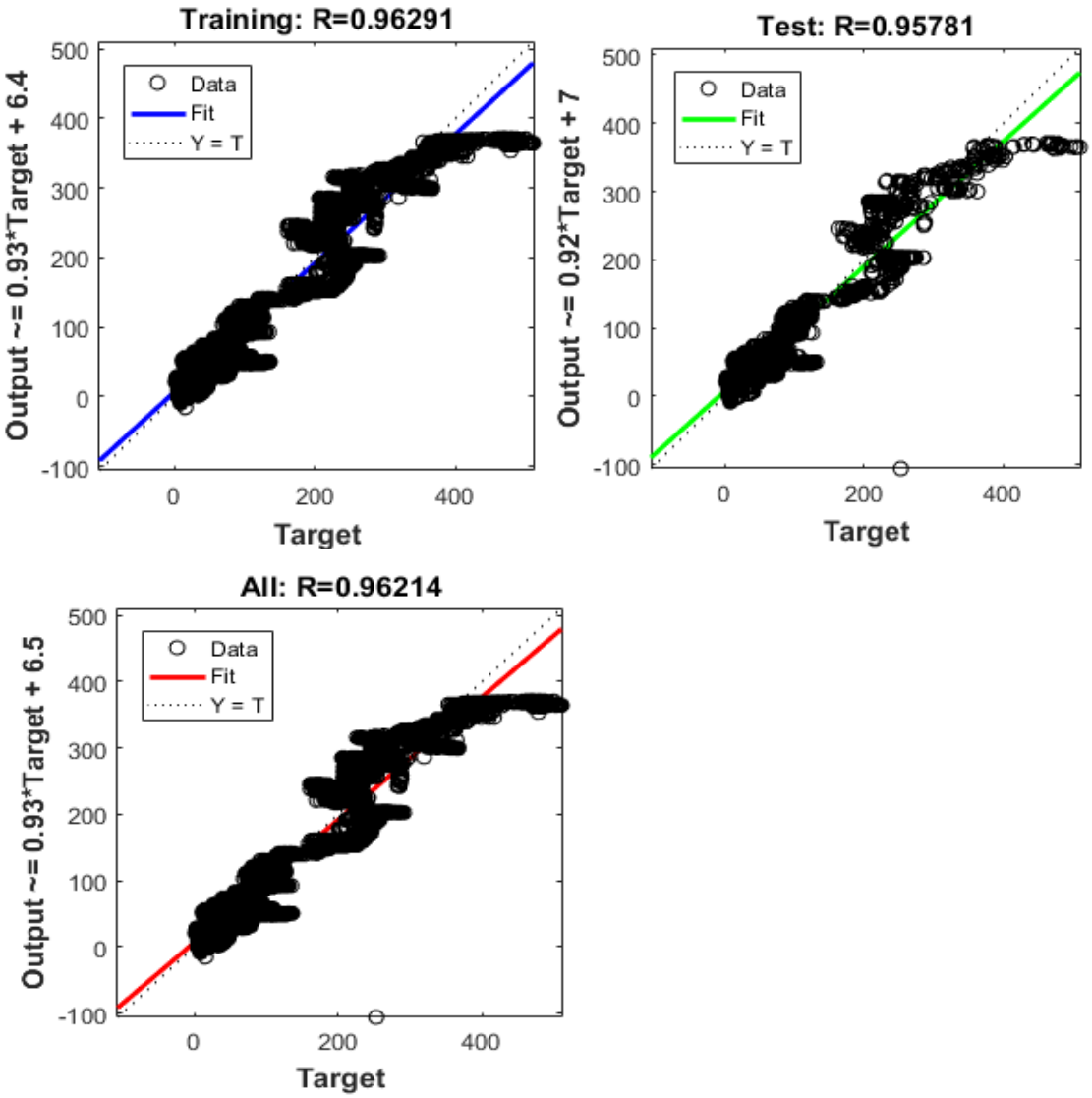

Figure 4.19: Regression value of two hidden layer (6-10-5-1) and 70% training data

In figure 4.20, at the mid of this plot, have a bin corresponding to the error of -0.263 and the height of that bin for training dataset lies above but near to 5000 and test dataset lies between 5000 and 6000.

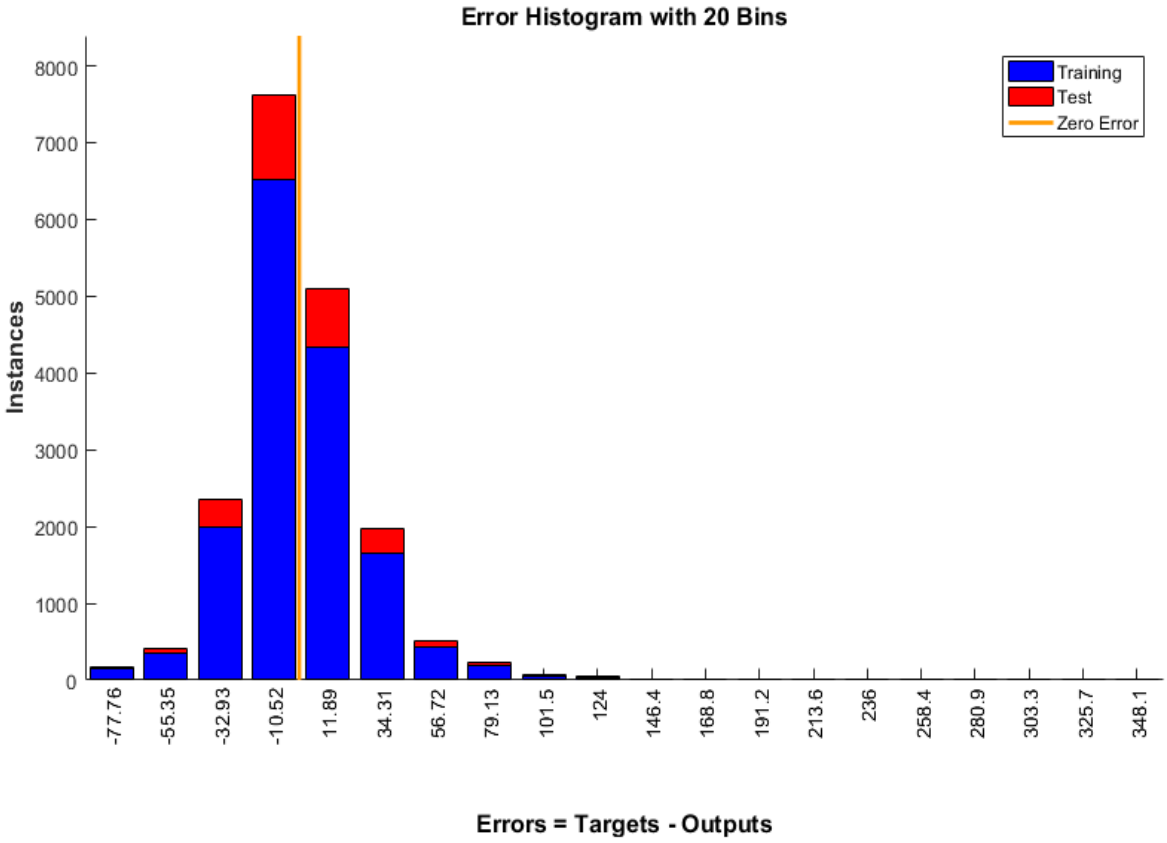

Figure 4.20: Error Histogram of two hidden layer (6-10-5-1) and 70% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under center of bin between 10.52 and11.89 . In figure 4.20, maximum errors fall between -32.93 and 34.31.

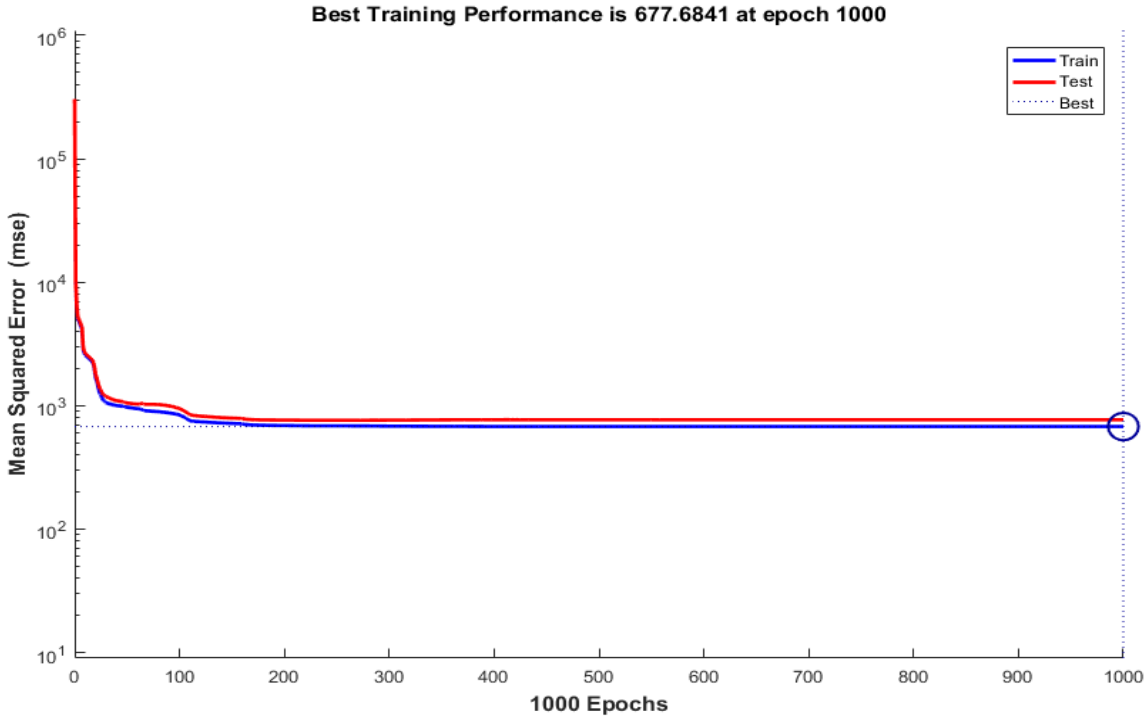

Figure 4.21: Epochs vs MSE of two hidden layer (6-10-5-1) and 70% training data

In figure 4.21, the training, testing and validation curves are very similar and no significant overfitting has occurred. 1000 epochs indicates that the training stopped at iteration 1000 and at iteration 1000 where the best validation performance occurs and error is minimum.

4.3.3 With two hidden layer (6-10-8-1) and 70% training data

Figure 4.22 shows the training data indicates a good fit. Training regression value is 0.979 and testing regression value is 0.978.

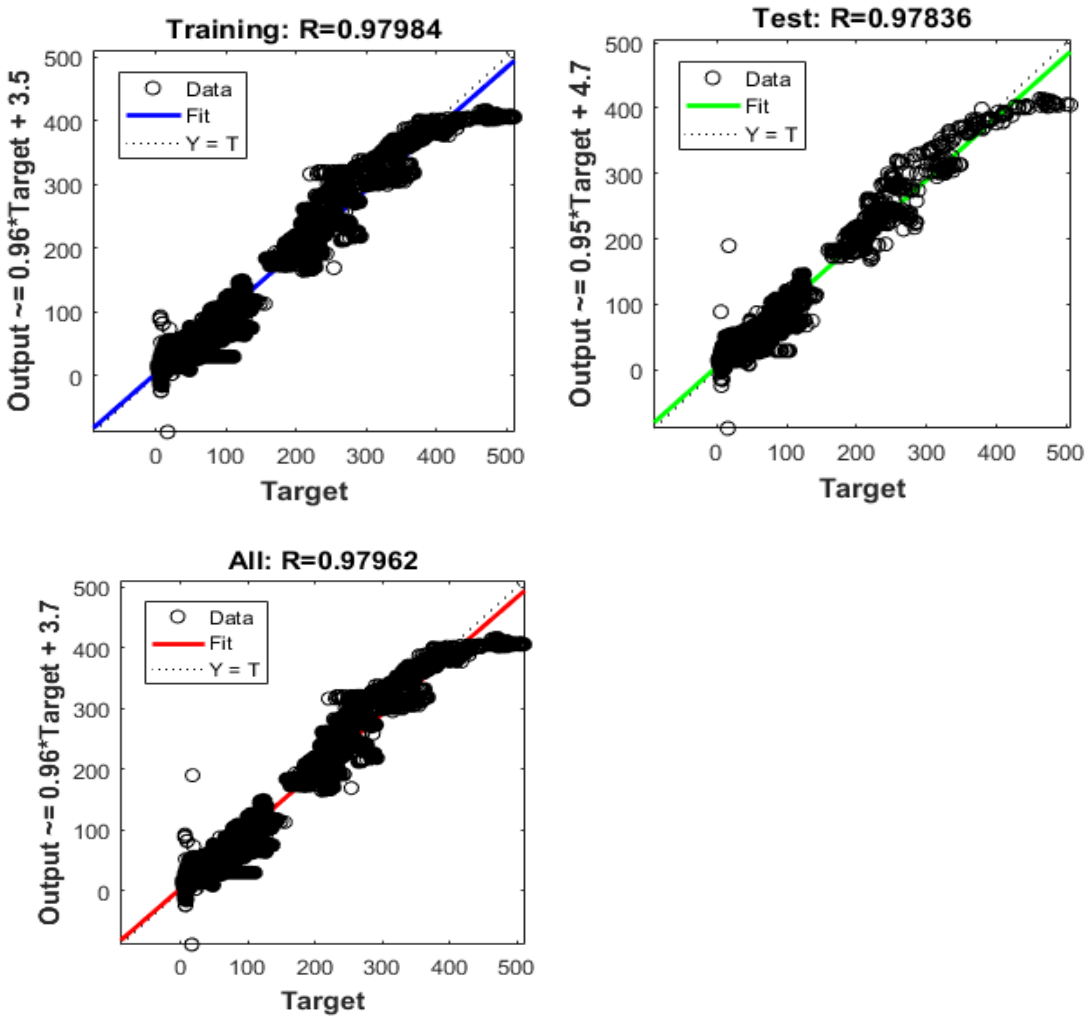

Figure 4.22: Regression of two hidden layer (6-10-8-1) and 70% training data

In figure 4.23, at the mid of this plot, have a bin corresponding to the error of 1.24 and the height of that bin for training dataset lies above but near to 5000 and test dataset lies between 5000 and 6000. It means that many samples from different datasets have an error lies in that following range.

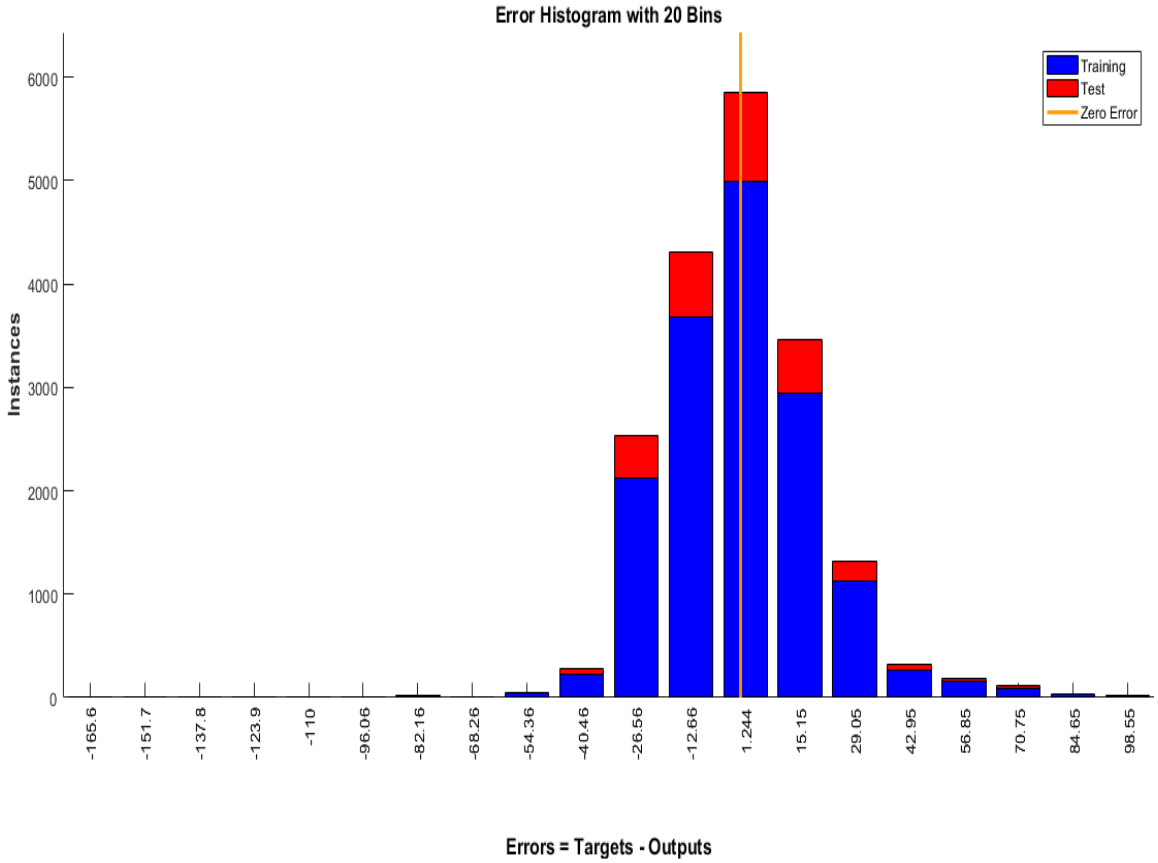

Figure 4.23: Error histogram of two hidden layer (6-10-8-1) and 70% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the bin with center 1.24. Figure 4.23 shows most errors fall between -26.56 to 29.05.

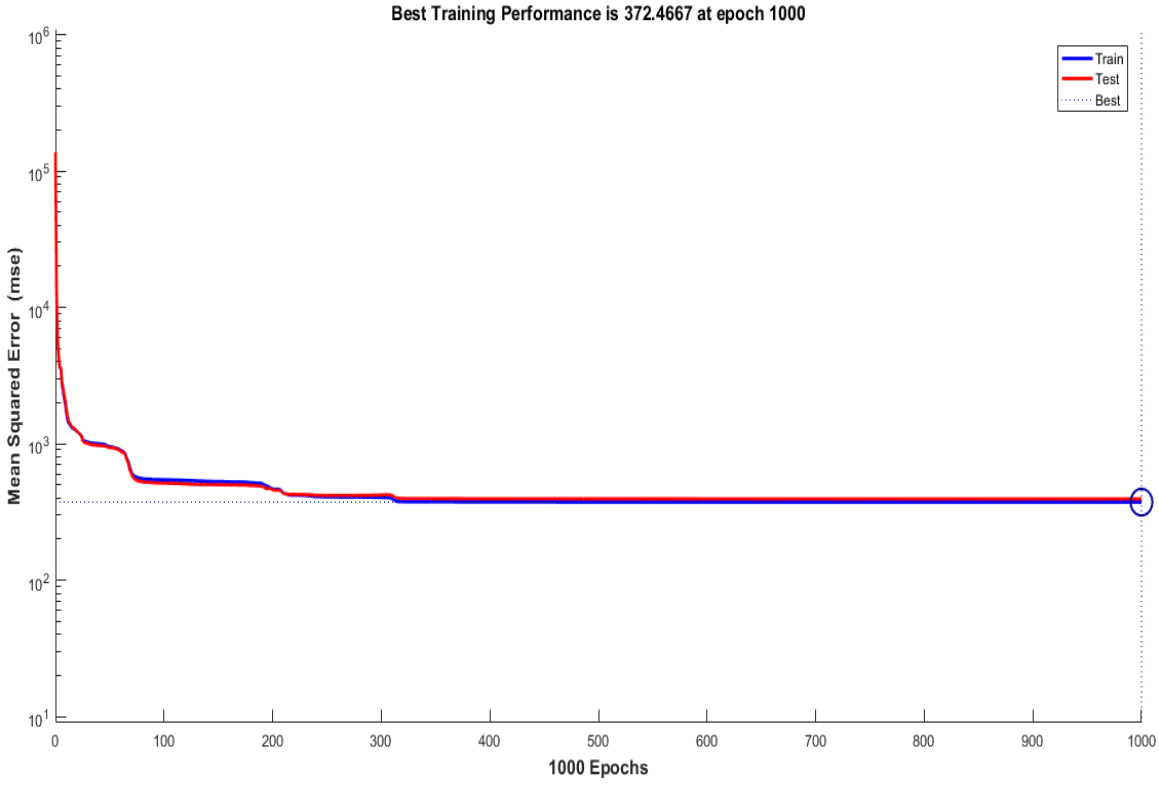

Figure 4.24: Epochs vs MSE of two hidden layer (6-10-8-1) and 70% training data

In figure 4.24, the training, testing and validation curves are very similar and no significant overfitting has occurred. 1000 epochs indicates that the training stopped at iteration 1000 and at iteration 1000 where the best validation performance occurs and error is minimum.

Table 4.4: Summary of Bayesian Regularization algorithm

Algorithm	Hidden Layer	Hidden Neuron	Training Data	Regression Value	Error Range
Bayesian Regularization	1	10	70%	0.942	-47.66 to 52.03
	2	10-5	70%	0.977	-28.94 to 27.75
	2	10-8	70%	0.979	-26.56 to 29.05

Table 4.4 shows that in Bayesian regularization algorithm, two layer (6-10-8-1) and 70% training, 15% testing and 15% validation data given the higher regression value and lower error range.

Table 4.5: Error testing of Bayesian Regularization algorithm

Sample No	**Targets**	**Predicted**	**Error (%)**	**Predicted**	**Error (%)**	**Predicted**	**Error (%)**
		For 6-10-8-1 and 70% training data		For 6-10-5-1 and 70% training data		For 6-10-1 and 70% training data	
1	6.3	5.3	15.87	5.1	19.04	5	20.63
2	109.8	119.7	9.01	117.7	7.19	101.7	7.37
3	263.3	277.92	5.55	149.91	5.08	247.5	6
4	14.78	11.21	24.15	11.21	24.15	11.21	24.15
5	163.49	170.67	4.39	173.67	6.22	178.67	9.28
6	49.6	45.12	9.03	44.12	11.04	44.23	10.82
7	47.66	43.11	9.54	43.11	9.54	43	9.77
8	250.4	251.78	0.55	252.9	0.99	240.3	3.79
9	19.62	17.61	10.23	16.5	15.90	16.9	13.86
10	15.81	13.83	12.48	12.7	19.67	12.6	20.30
11	218.24	234.49	7.44	205.49	5.84	233.48	6.98
12	34.13	32.82	3.83	29.9	12.39	29.89	12.39
13	105	101.84	3.0	98.84	5.86	112.84	7.4
14	5.7	4.9	14.03	6.5	14.03	4.9	14.03
15	27.62	29.87	8.14	30.85	11.69	30.69	11.11
16	31.05	27.1	12.72	27.5	11.43	27.2	12.39
17	123	128.4	4.39	135.4	10.08	135.4	10.08
18	17.77	15.01	15.53	13.9	21.77	12.9	27.40
19	19	17.09	10.05	17.3	8.94	21.4	12.63
10	251.78	245.76	2.39	278.96	10.08	220.96	12.24
21	161.06	153.22	4.86	149.7	7.05	147.7	8.29
22	20	17.8	11	16	20	16.3	18.5
23	37.4	35.43	5.26	35.1	6.14	34.1	8.82
24	120.5	125.3	3.98	129.7	7.63	111.6	7.30
25	45.2	40.3	10.84	51.3	13.49	51.2	13.27
		Avg error	8.73	Avg error	11.44	Avg error	12.35

Table 4.5 shows the error and average error of Bayesian Regularization training algorithm. This table shows the average error of Levenberg Marquardt with two hidden layer (6-10-8-1) and 70% training is 8.73, two hidden layer (6-10-5-1) and 70% training is 11.44 and one hidden layer (6-10-1) and 70% training data is 12.35. This thesis

experiments more with changing hidden layer and training data but this table shows only best three.

Table 4.6: Accuracy testing of Bayesian Regularization algorithm

Hidden Layer with Neuron and Training data	Average Error (%)	Accuracy (%)
For 6-10-8-1 and 70% training data	8.74	91.26
For 6-10-5-1 and 70% training data	11.44	88.56
For 6-10-1 and 80% training data	12.35	87.65

From Table 4.5 we got the average error Table 4.6 shows the accuracy of Bayesian Regularization algorithm, the accuracy of one hidden layer (6-10-1) with 70% training data is 87.65, two hidden layer (6-10-5-1) with 70% training data is 88.56 and two hidden layer (6-10-8-1) with 70% training data is 91.26 and this is the best accuracy of Bayesian Regularization training algorithm.

4.4 SCALED CONJUGATE GRADIENT

4.4.1 With two hidden layer (6-5-5-1) and 70% training data

Figure 4.25 shows the training data indicates a good fit. Training regression value is 0.910, testing regression value is 0.9146 and validation regression value is 0.9055.

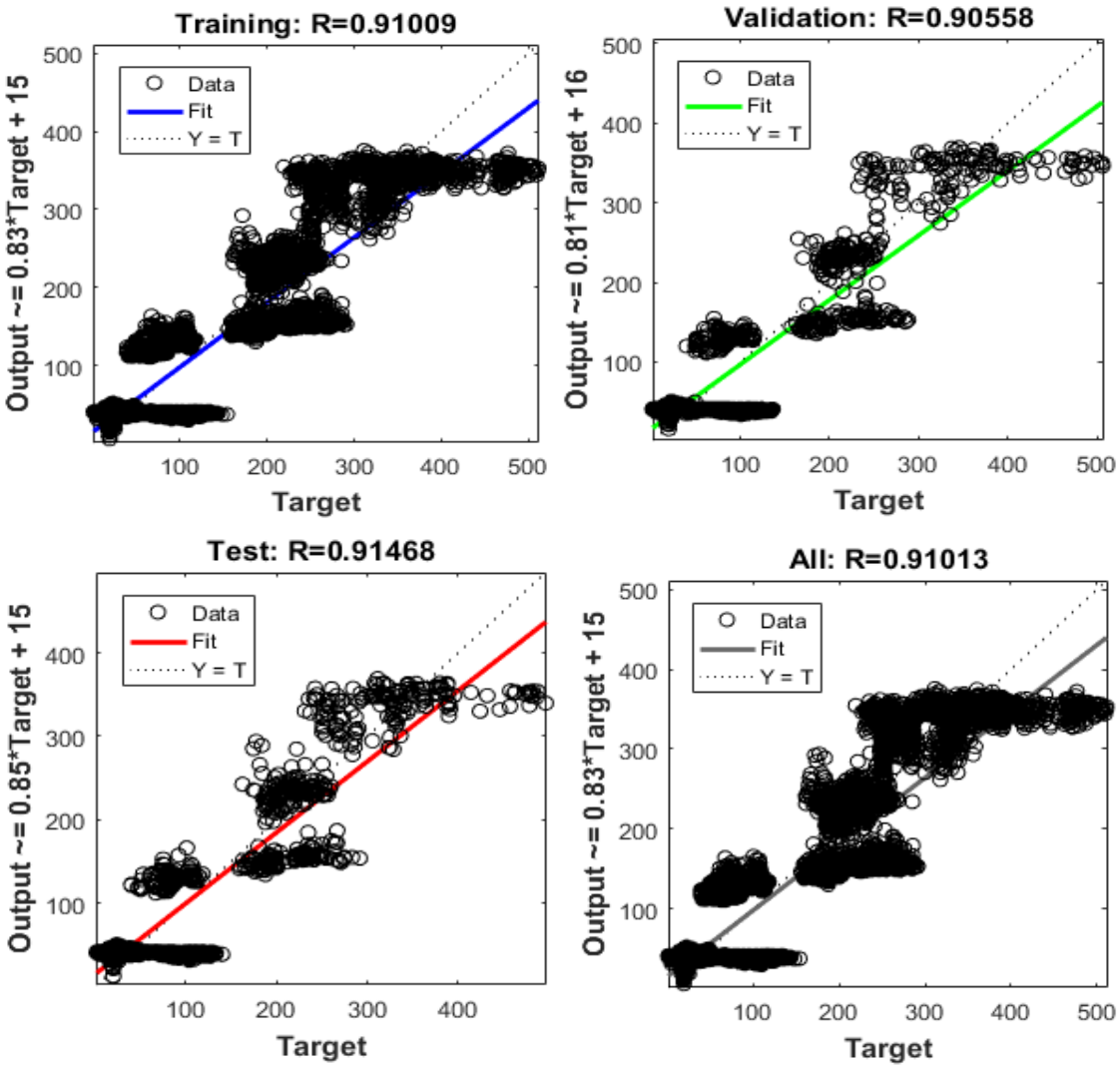

Figure 4.25: Regression value of two hidden layer (6-5-5-1) and 70% training data

In figure 4.26, at the mid of this plot have a bin corresponding to the error of -10.7 and the height of that bin for training dataset lies above but near to 3000 and test dataset lies between 4000 and 4500.

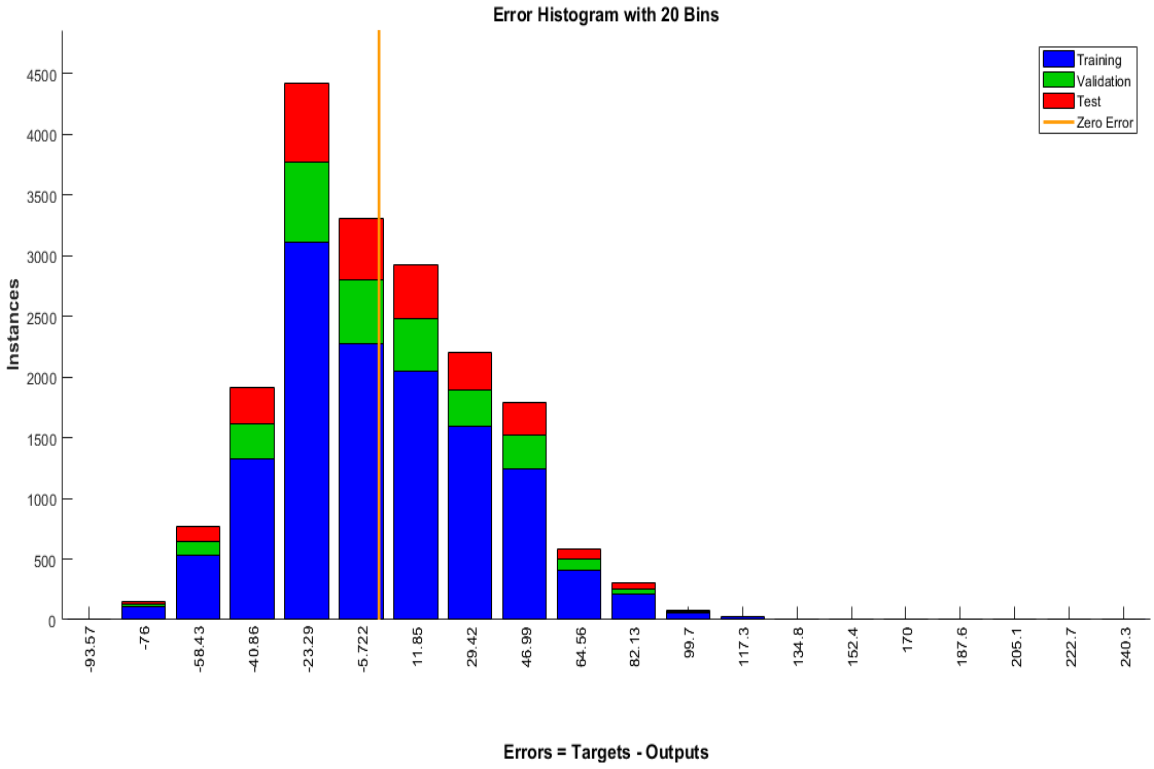

Figure 4.26: Error histogram of two hidden layer (6-5-5-1) and 70% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the bin 5.72. Figure 4.26 shows maximum error fall between -40.86.78 to 46.99.

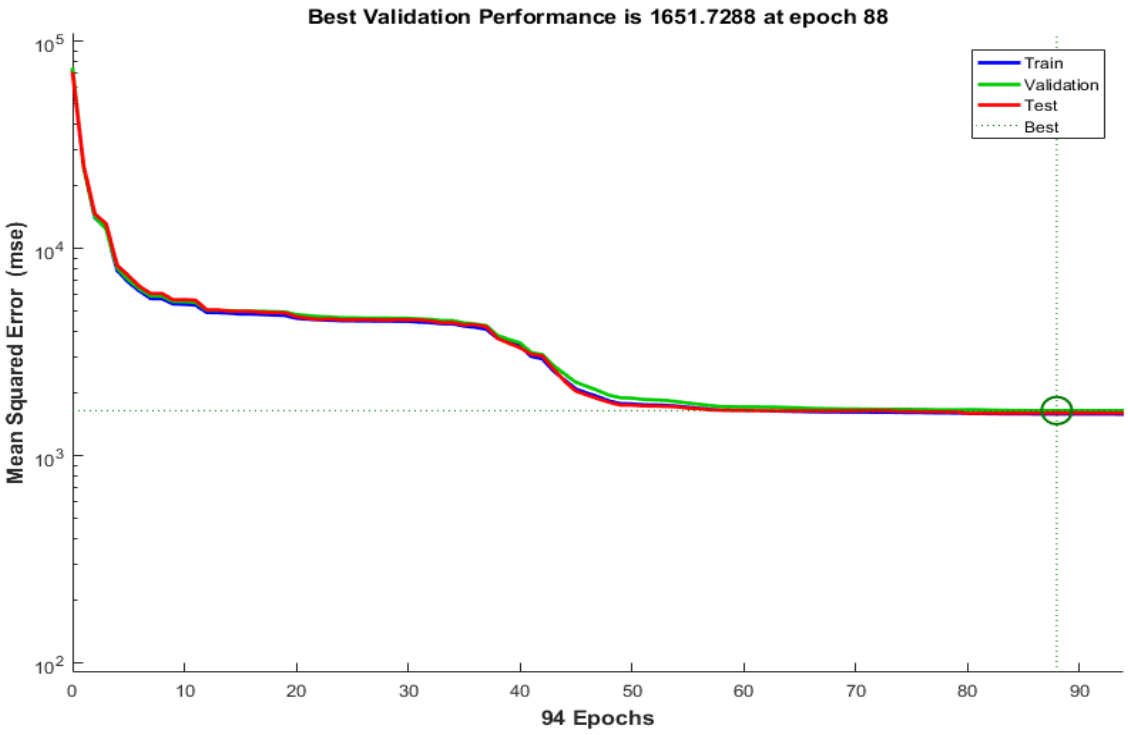

Figure 4.27: Epochs vs MSE of two hidden layer (6-5-5-1) and 70% training data

In figure 4.27, the training, testing and validation curves are very similar and no significant overfitting has occurred. 94 epochs indicates that the training stopped at iteration 94 and at iteration 88 where the best validation performance occurs and error is minimum.

4.4.2 With one hidden layer (6-10-1) and 80% training data

Figure 4.28 shows the training data indicates a good fit. Training regression value is 0.913, testing regression value is 0.9133 and validation regression value is 0.912.

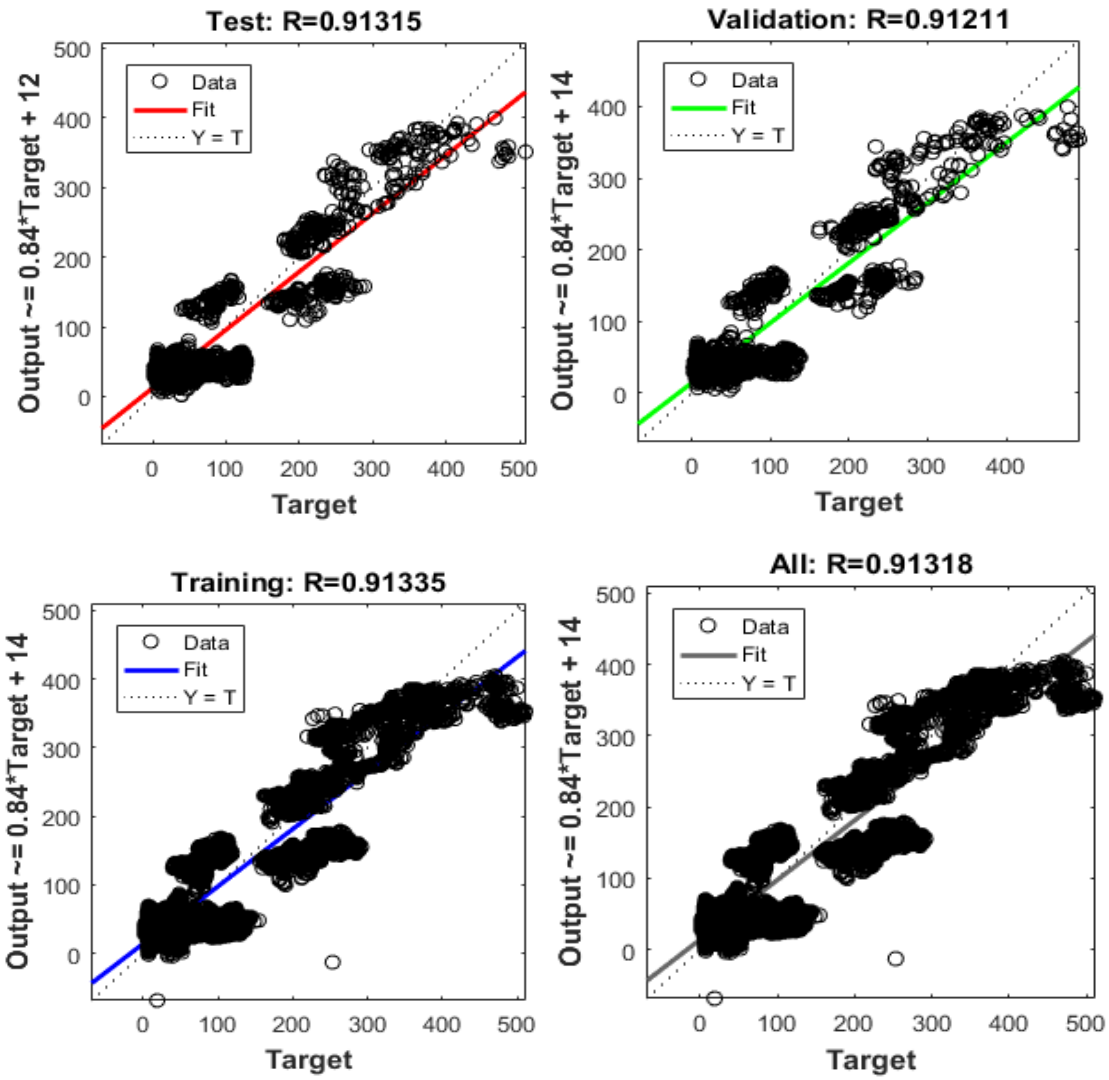

Figure 4.28: Regression value of two hidden layer (6-10-1) and 80% training data

In figure 4.29, at the mid of this plot have a bin corresponding to the error of -10.36 and the height of that bin for training dataset lies above but near to 3000 and test dataset lies between 3500 and 4000.

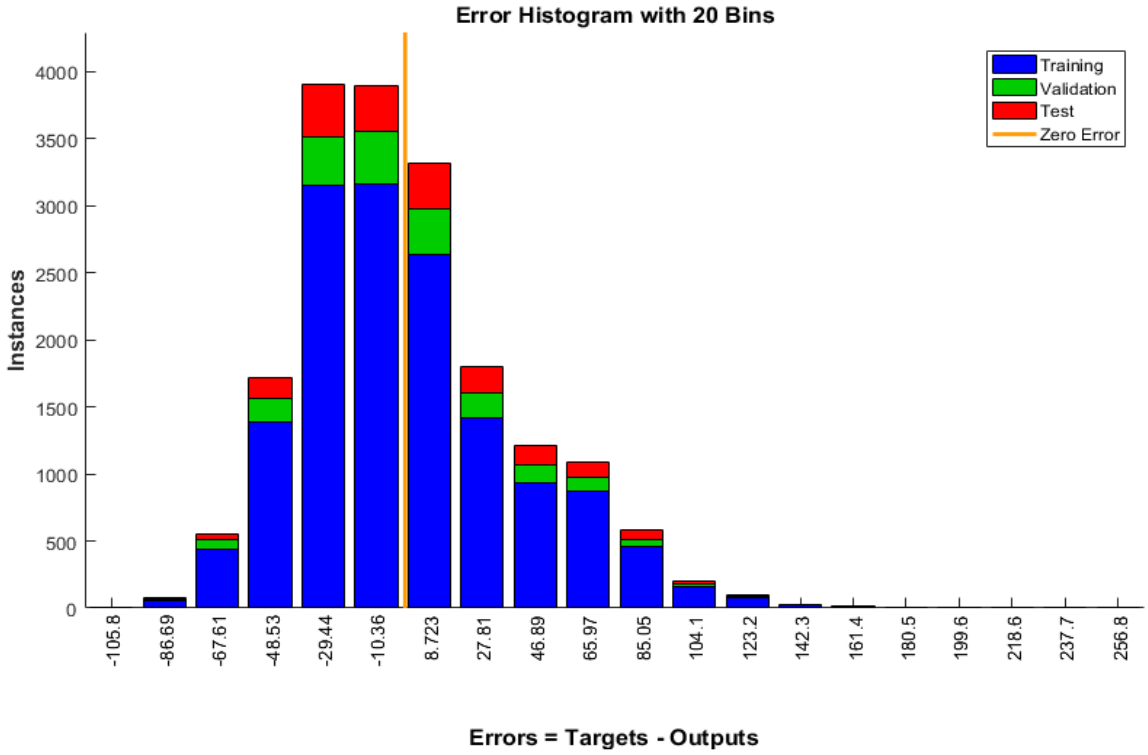

Figure 4.29: Error histogram of two hidden layer (6-10-1) and 80% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the center bin between -10.3 and 8.7. Figure 4.29 shows most errors fall between -48.53 and 65.97.

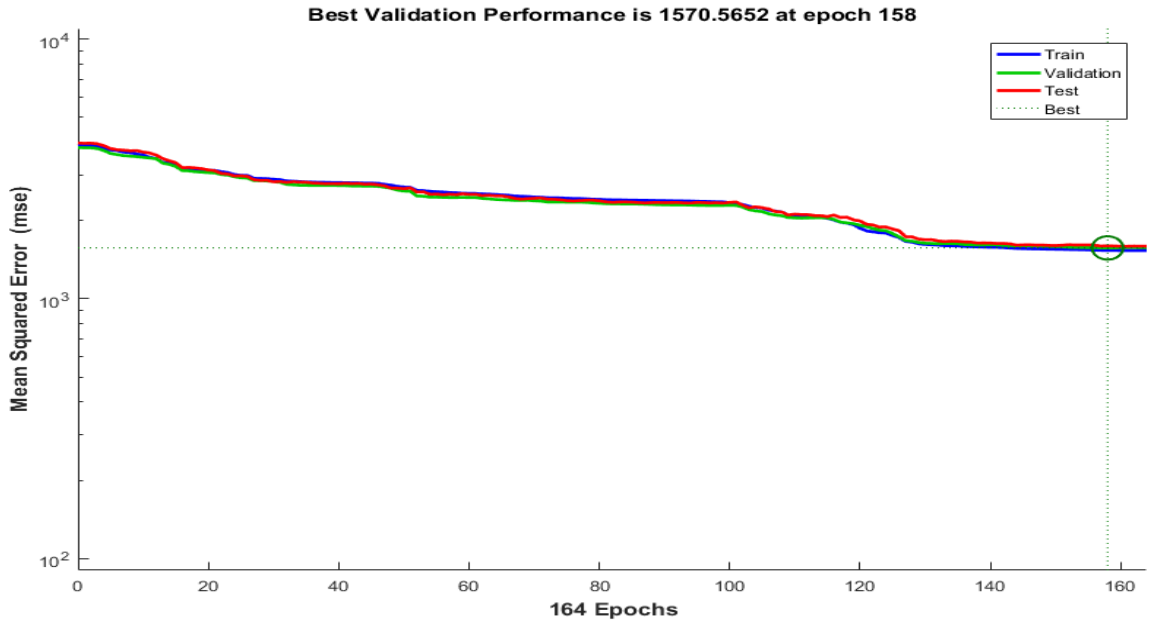

Figure 4.30: Epochs vs MSE of two hidden layer (6-10-1) and 80% training data

In figure 4.30, the training, testing and validation curves are very similar and no significant overfitting has occurred. 164 epochs indicates that the training stopped at iteration 164 and at iteration 158 where the best validation performance occurs and error is minimum.

4.4.3 With two hidden layer (6-10-8-1) and 70% training data

Figure 4.31 shows the training data indicates a good fit. Training regression value is 0.947, testing regression value is 0.943 and validation regression value is 0.944.

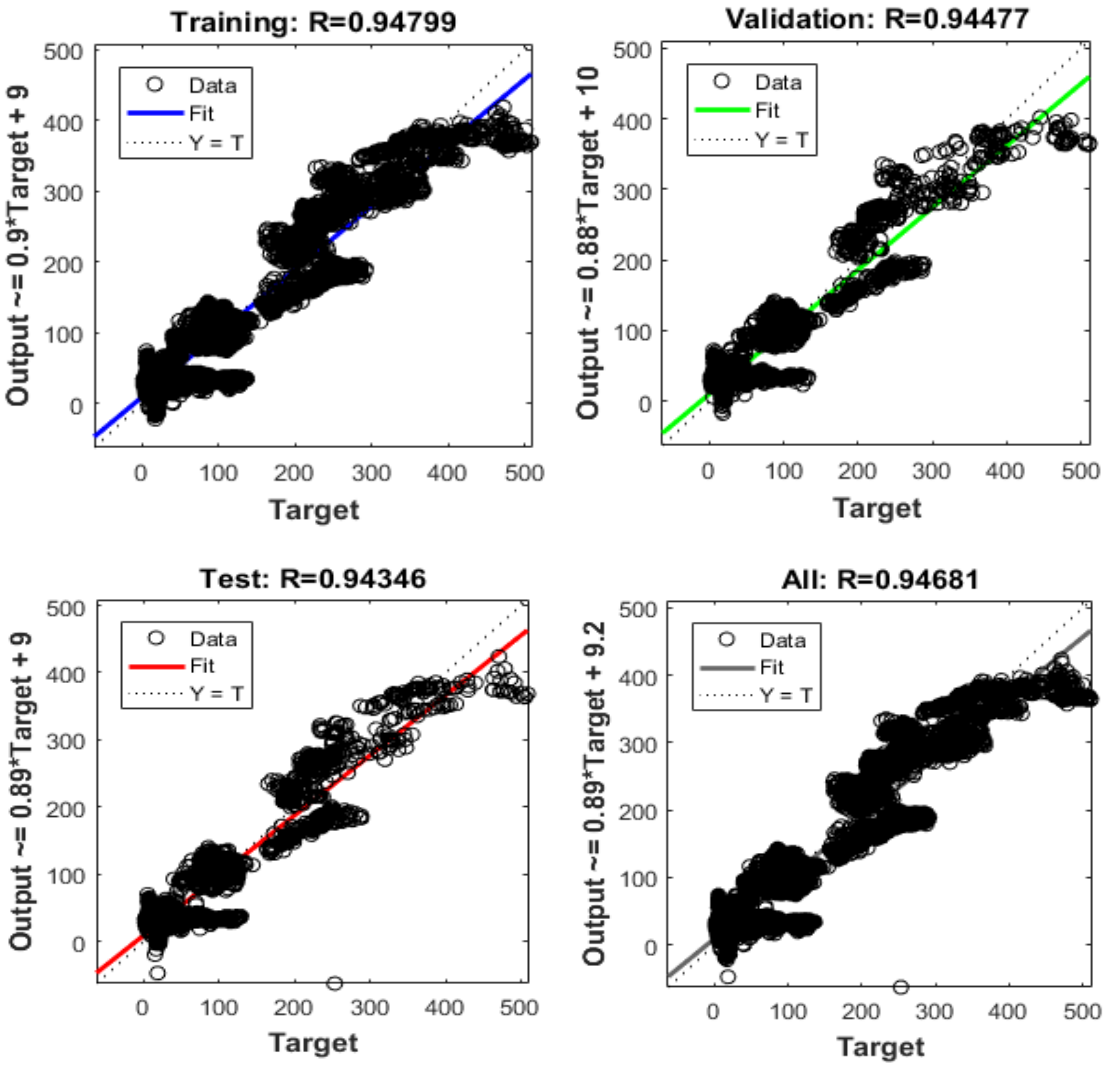

Figure 4.31: Regression value of two hidden layer (6-10-8-1) and 70% training data

In figure 4.32, at the mid of this plot have a bin corresponding to the error of -10.7 and the height of that bin for training dataset lies above but near to 3000 and test dataset lies between 4500 and 5000.

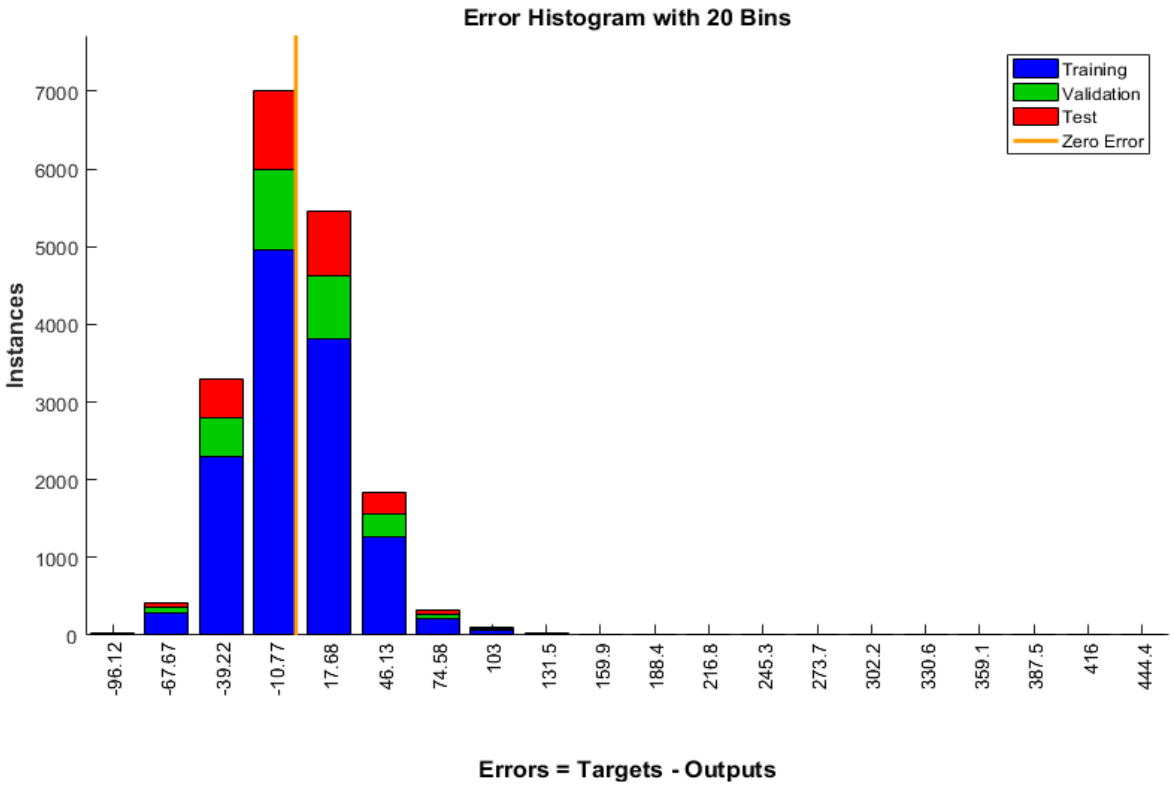

Figure 4.32: Error histogram of two hidden layer (6-10-8-1) and 70% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the center of bin between -10.7 and 10.37. In figure 4.32, maximum errors fall between in -31.78 and 31.45.

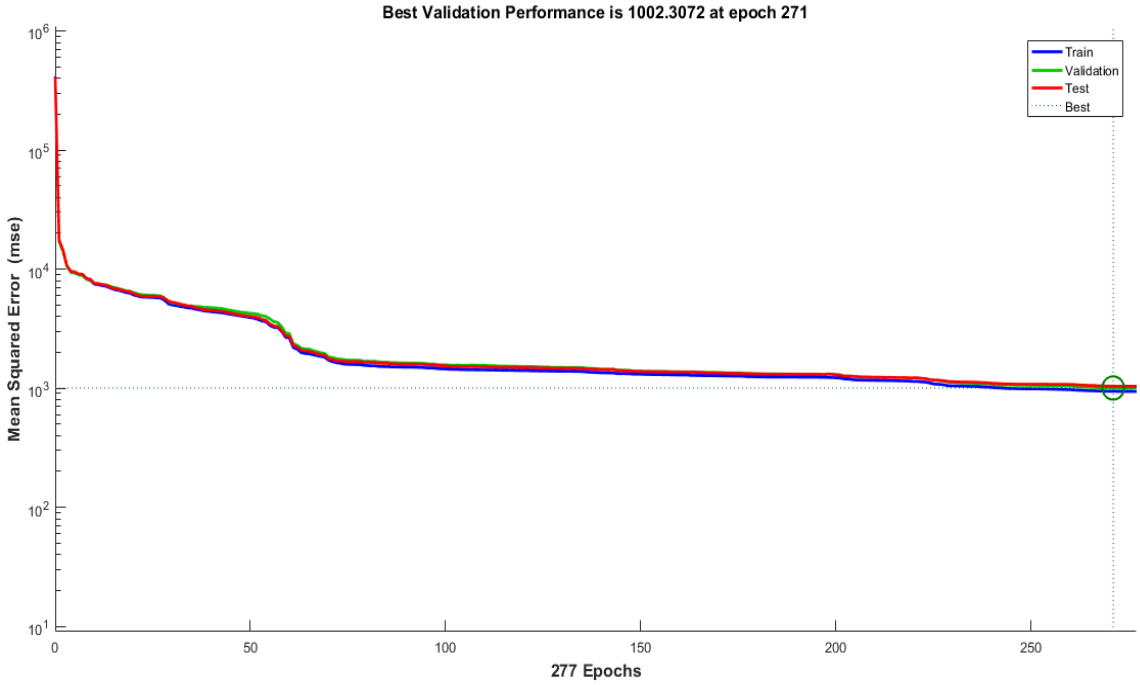

Figure 4.33: Epochs vs MSE of two hidden layer (6-10-8-1) and 70% training data

In figure 4.33, the training, testing and validation curves are very similar and no significant overfitting has occurred. 277 epochs indicates that the training stopped at iteration 277 and at iteration 271 where the best validation performance occurs and error is minimum.

Table 4.7: Summary of Scale Conjugate Gradient Algorithm

Algorithm	Hidden Layer	Hidden Neuron	Training Data	Regression Value	Error Range
	2	10-5	70%	0.9101	-40.86.78 to 46.99
Scale Conjugate Gradient	1	10	80%	0.9131	-28.94 to 27.75
	2	10-8	70%	0.9468	-31.78 to 31.45.

Table 4.7 shows that in Scaled Conjugate Gradient algorithm, two hidden layer (6-10-8-1) and 70% training, 15% testing and 15% validation data set given the higher regression value and lower error range.

Table 4.8: Error testing of Scaled Conjugate Gradient algorithm

Sample No.	Targets	Predicted	Error (%)	Predicted	Error (%)	Predicted	Error (%)
		For 6-10—5-1 and 70% training data		6-10-8-1 and 70% training data		For 6-10-5-1 and 80% training data	
1	6.3	4.9	22.22	5.1	19.04	7.89	25.23
2	109.8	120.7	9.92	117.7	7.19	100.7	8.28
3	263.3	291.92	10.86	277.92	5.55	230.92	12.29
4	14.78	10.9	26.25	11.21	24.15	10.9	26.25
5	163.49	193.67	18.45	173.67	6.22	195.67	19.68
6	49.6	44.21	10.86	44.12	11.04	44.5	10.28
7	47.66	52.11	9.33	43.11	9.54	44.11	7.44
8	250.4	231.9	7.38	252.9	0.99	230.1	8.10
9	19.62	16.2	17.43	16.5	15.90	16.2	17.43
10	15.81	18.7	18.27	12.7	19.67	12.9	18.40
11	218.24	240.49	10.19	238.49	9.27	244.49	12.02
12	34.13	30.7	10.04	31.1	8.87	30.9	9.46
13	105	110.7	5.42	100.84	3.96	110	4.76
14	5.7	4.8	15.78	4.9	14.03	4.8	15.78
15	27.62	34.85	26.17	31.85	15.31	34.85	26.17
16	31.05	27.1	12.72	27.1	12.72	27.1	12.72
17	123	111.4	9.43	131.4	6.82	111.4	9.43
18	17.77	23.55	32.52	13.55	23.74	22.55	26.89
19	19	16.7	12.10	17.10	10	16.69	12.10
20	251.78	241.96	3.90	241.96	3.90	231.96	7.87
21	161.06	152.5	5.31	153.7	4.56	173.5	7.72
22	20	15.8	21	15.8	21	24.2	21
23	37.4	34.2	8.55	35.2	5.88	34.15	8.68
24	120.5	112.7	6.47	128.7	6.80	129	7.05
25	45.2	50.3	11.28	40.3	10.84	40.6	10.17
		Avg error	13.67	Avg error	11.09	Avg error	13.81

Table 4.8 shows the error and average error of Bayesian Regularization training algorithm. This table shows the average error of Scale Conjugate Gradient with two hidden layer (6-10-8-1) and 70% training is 11.09, two hidden layer (6-10-5-1) and 70% training is 13.67 and one hidden layer (6-10-1) and 70% training data is 13.81. This thesis

experiments more with changing hidden layer and training data but this table shows only best three.

Table 4.9: Accuracy testing of Scaled Conjugate Gradient algorithm

Hidden Layer with Neuron and Training data	Average Error (%)	Accuracy (%)
For 6-10-1 and 80% training data	13.67	86.33
For 6-5-5-1 and 70% training data	13.81	86.19
For 6-10-8-1 and 70% training data	11.09	88.91

From Table 4.8 we got the average error and Table 4.9 shows the accuracy of Scale Conjugate Gradient training algorithm, the accuracy of two hidden layer (6-5-5-1) with 70% training data is 86.19, one hidden layer (6-10-1) with 70% training data is 86.33, two hidden layer (6-10-8-1) with 70% training data is 88.91 and this is the best accuracy of Scaled Conjugate Gradient training algorithm.

4.5 QUASI NEWTRON ALGORITHM

4.5.1 With one hidden layer (6-10-1) and 70% training data

Figure 4.34 shows the training data indicates not a good fit. Training regression value is 0.699, testing regression value is 0.685 and validation regression value is 0.687.

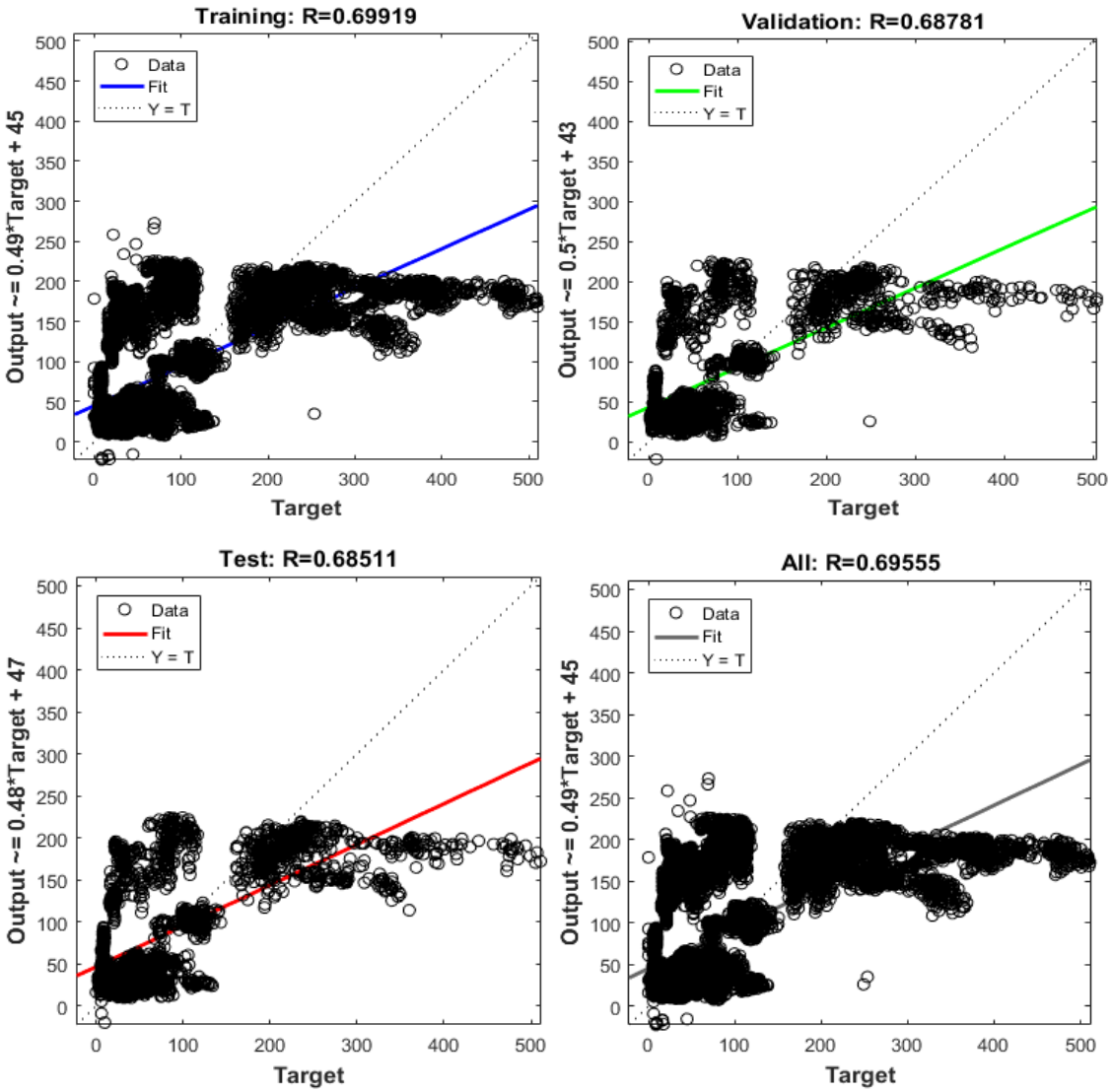

Figure 4.34: Regression value of one hidden layer (6-10-1) and 70% training data

In figure 4.35, at the mid of this plot have a bin corresponding to the error of -10.7 and the height of that bin for training dataset lies above but near to 3000 and test and validation dataset lies between 5000 and 6000. It means that many samples from different datasets have an error lies in that following range.

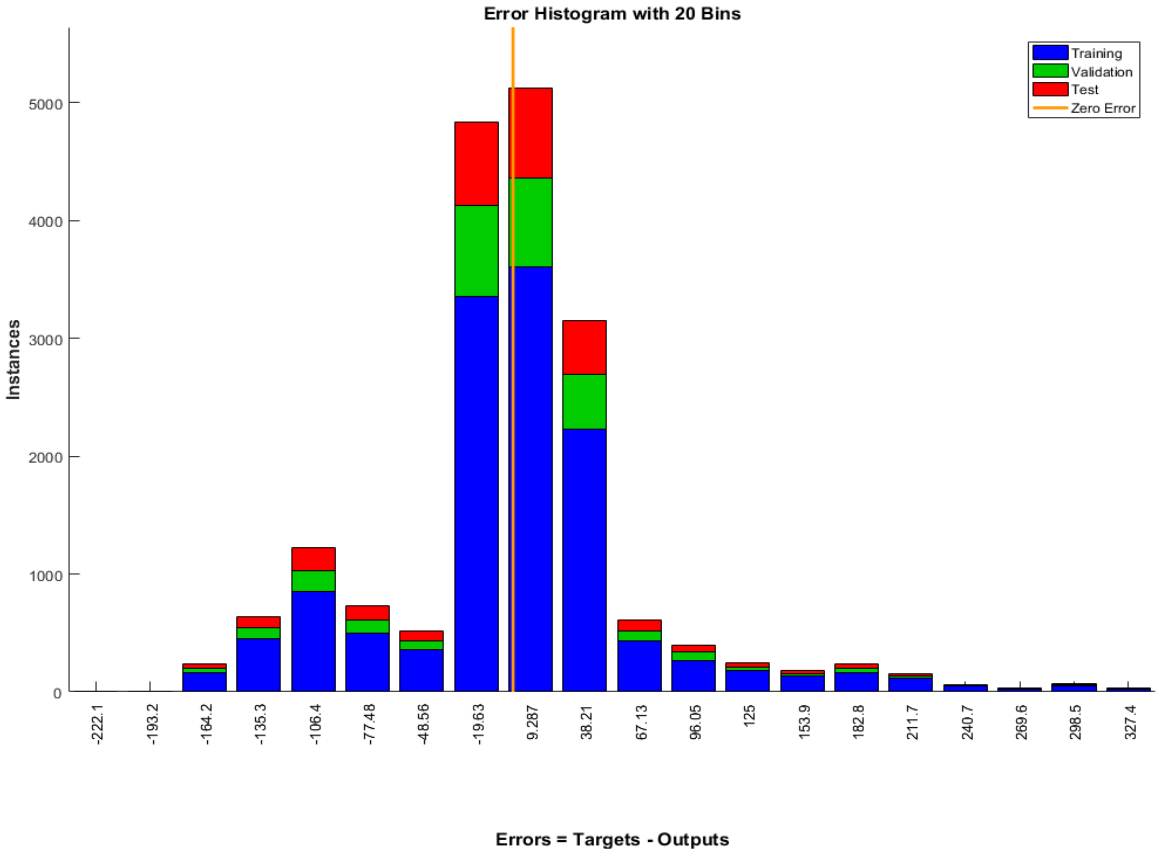

Figure 4.35: Error histogram of one hidden layer (6-10-1) and 70% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the bin 9.27. Figure 4.35 shows most errors fall between -31.78 and 31.45.

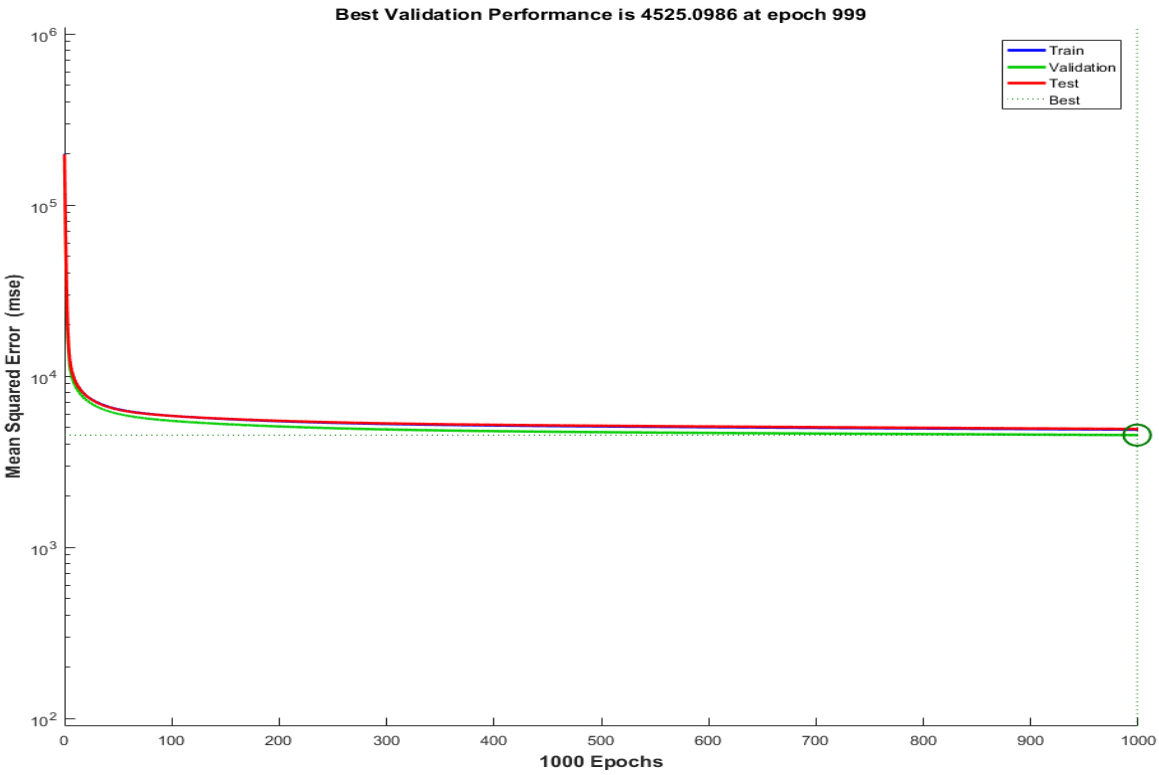

Figure 4.36: Epochs vs MSE of two hidden layer (6-10-1) and 70% training data

In figure 4.36, the training, testing and validation curves are very similar and no significant overfitting has occurred. 1000 epochs indicates that the training stopped at iteration 1000 and at iteration 999 where the best validation performance occurs and error is minimum.

4.5.2 With two hidden layer (6-10-5-1) and 70% training data

Figure 4.37 shows the training data indicates not a good fit. Training regression value is 0.823, testing regression value is 0.807 and validation regression value is 0. 818.

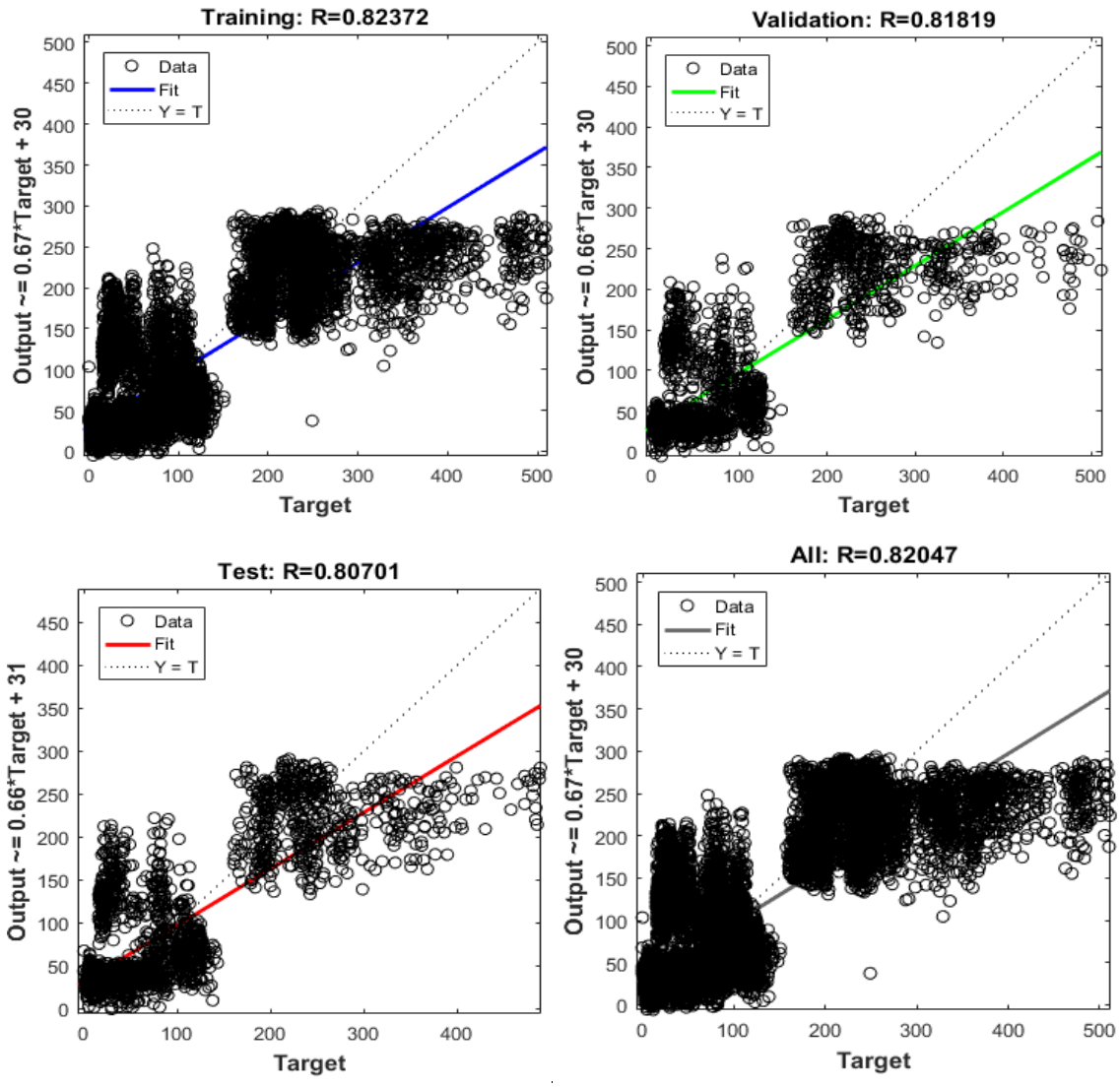

Figure 4.37: Regression value of two hidden layer (6-10-5-1) and 70% training data

In figure 4.38, at the mid of this plot have a bin corresponding to the error of 1.892 and the height of that bin for training dataset lies above but near to 3000 and test dataset lies between 4500 and 5000.

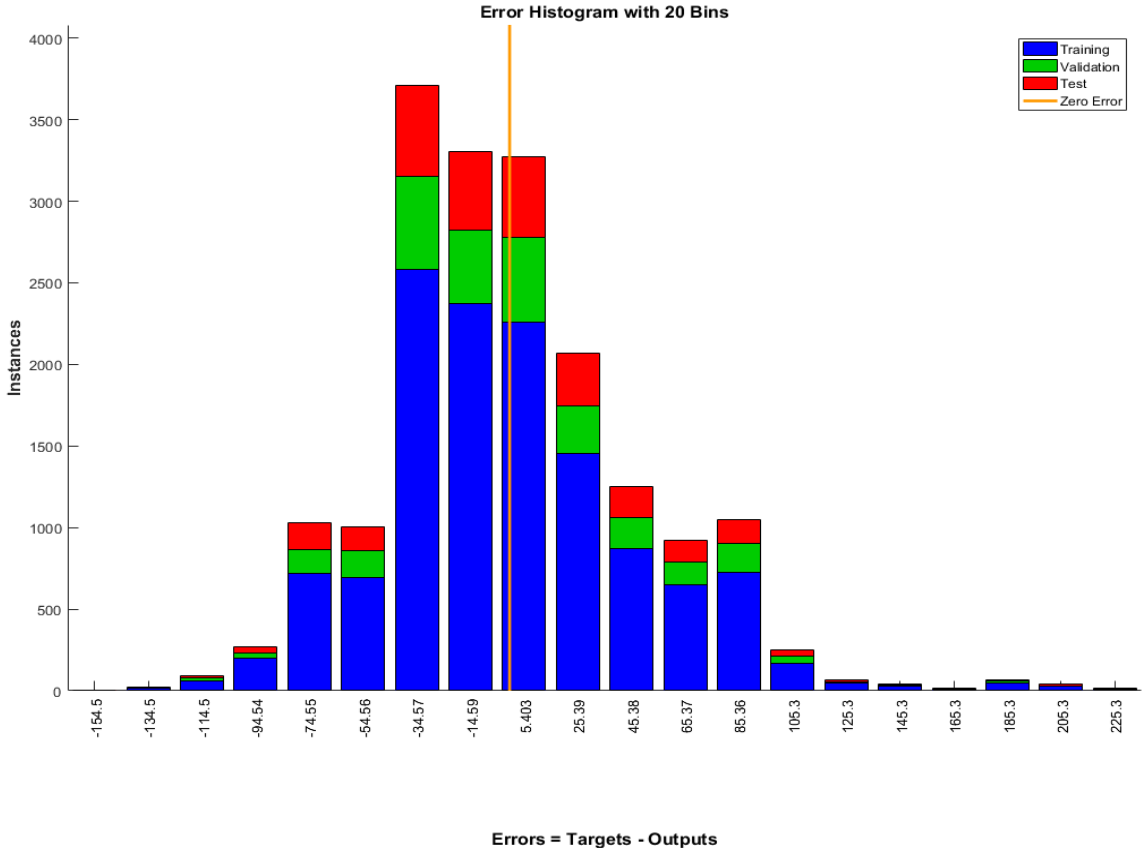

Figure 4.38: Error histogram of two hidden layer (6-10-5-1) and 70% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the bin with center 1.892. Figure 4.38 shows maximum errors fall between -75.17 and 78.95.

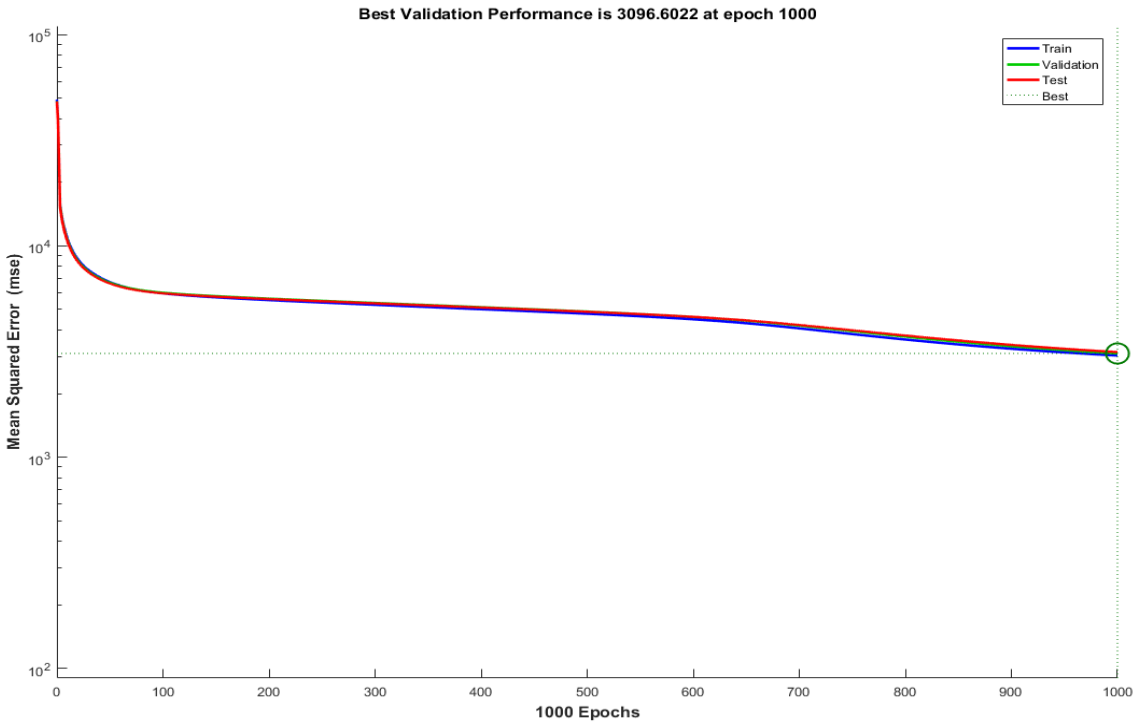

Figure 4.39: Epochs vs MSE of two hidden layer (6-10-5-1) and 70% training data

In figure 4.39, the training, testing and validation curves are very similar and no significant overfitting has occurred. 1000 epochs indicates that the training stopped at iteration 1000 and at iteration 1000 where the best validation performance occurs and error is minimum.

4.5.3 With two hidden layer (6-10-8-1) and 70% training data

Figure 4.40 shows the training data indicates not a good fit. Training regression value is 0.867, testing regression value is 0.864 and validation regression value is 0.867.

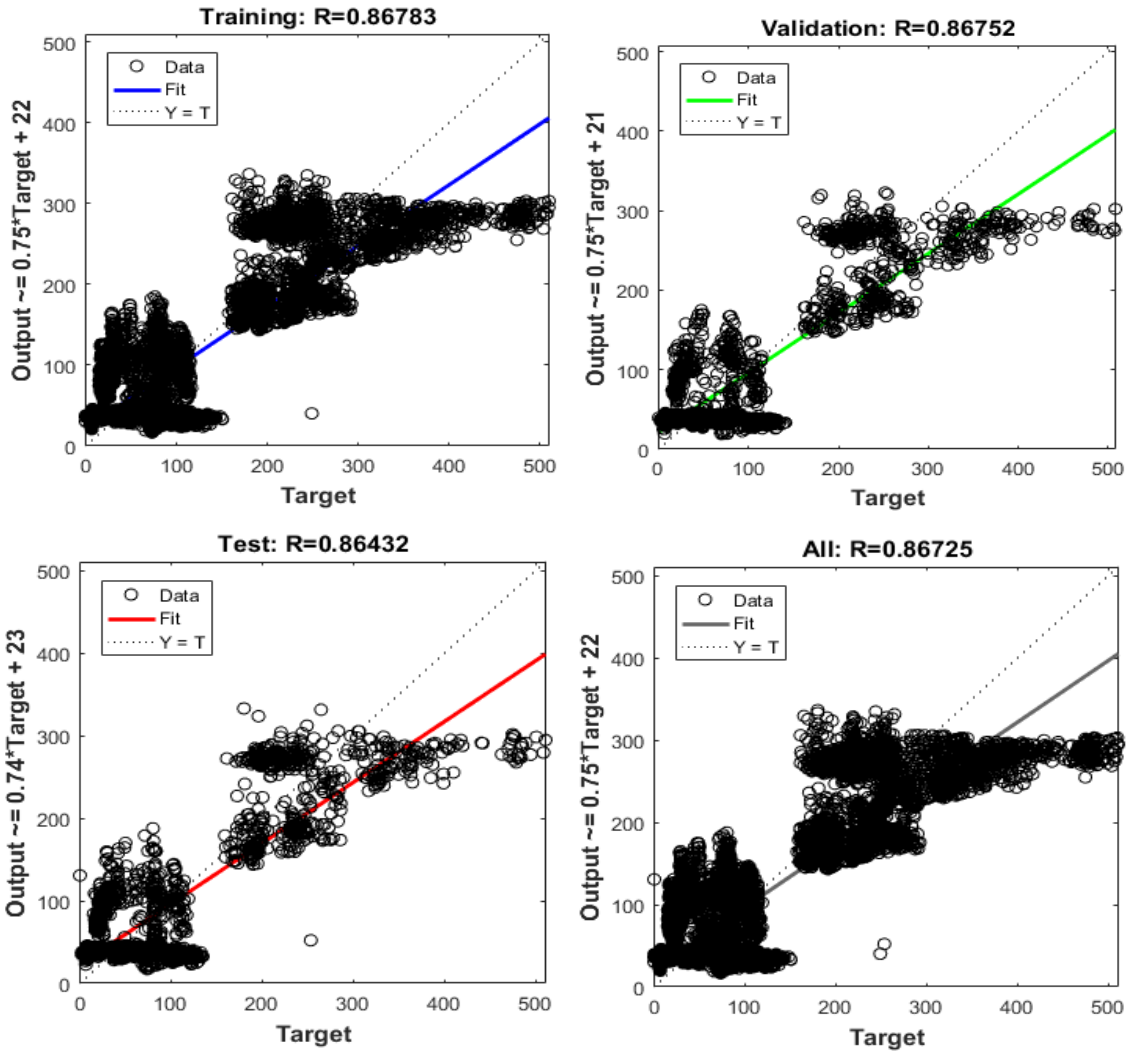

Figure 4.40: Regression value of two hidden layer (6-10-8-1) and 70% training data

In figure 4.41, at the mid of this plot have a bin corresponding to the error of 5.48 and the height of that bin for training dataset lies above but near to 3000 and test dataset lies between 4500 and 5000. It means that many samples from different datasets have an error lies in that following range.

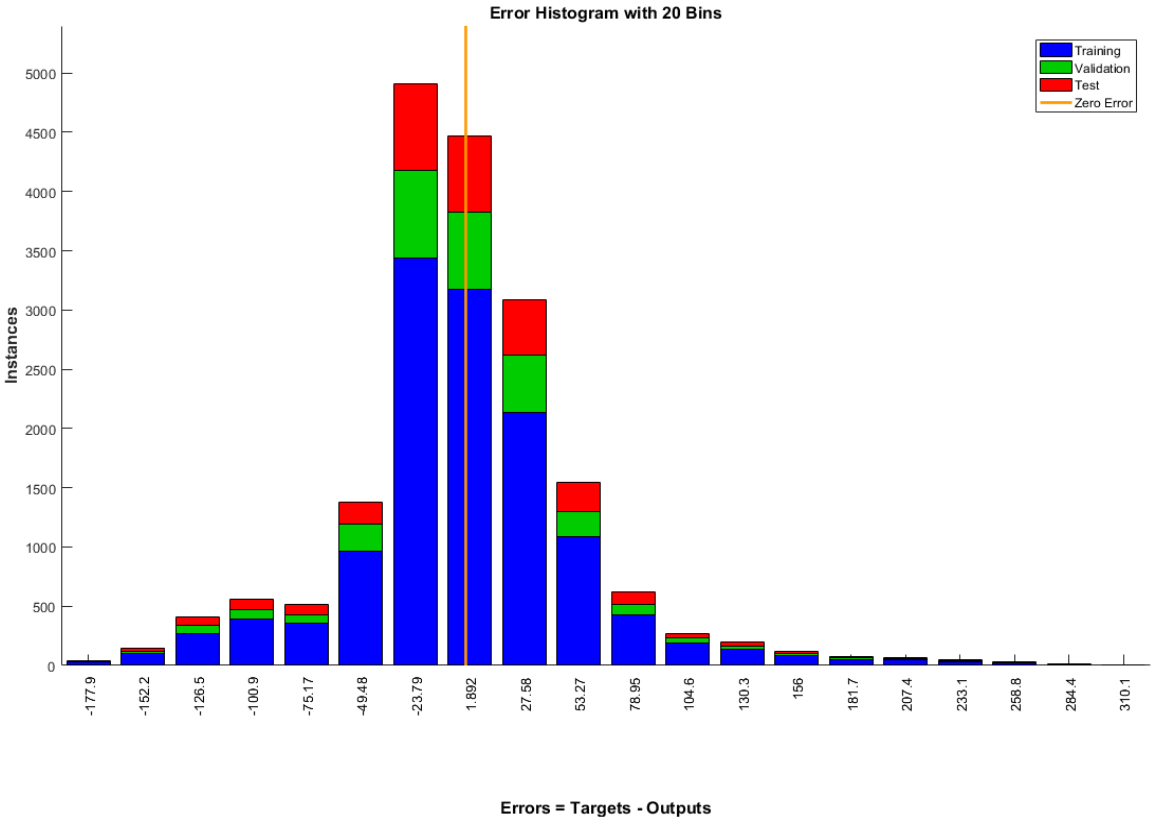

Figure 4.41: Error histogram of two hidden layer (6-10-8-1) and 70% training data

Zero error line corresponding to the zero error value on the error axis (i.e. X-axis). In this case zero error point falls under the bin with center 5.4. Figure 4.41 shows most errors fall between -74.55 and 85.35.

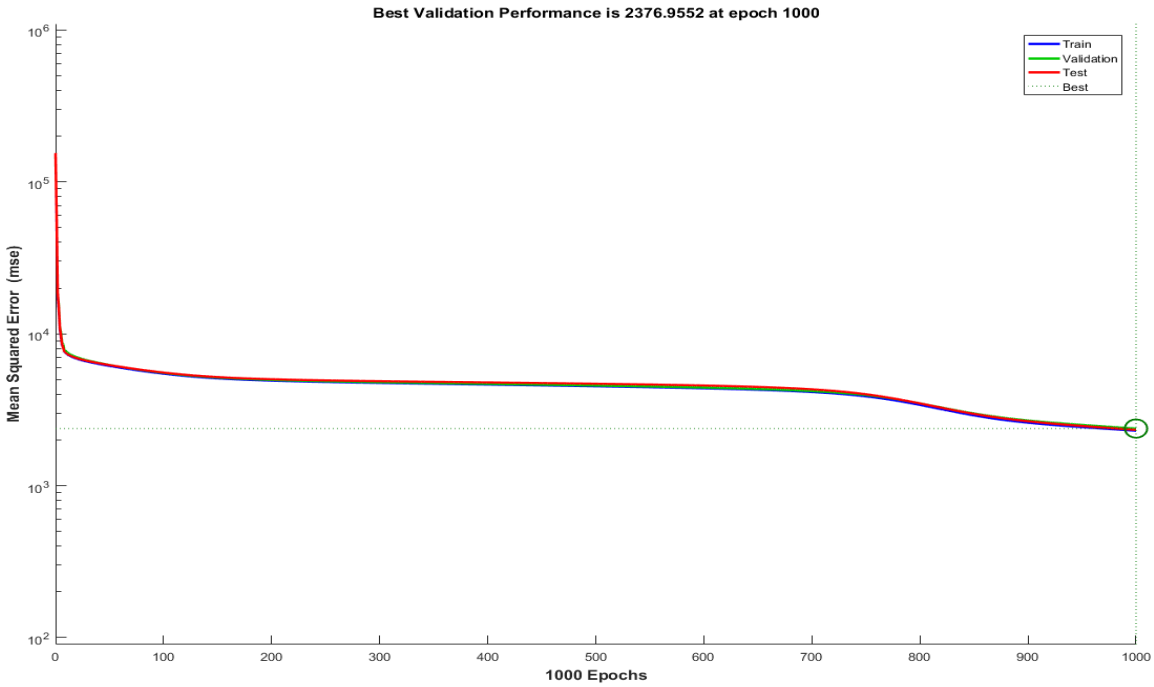

Figure 4.42: Epochs vs MSE of two hidden layer (6-10-8-1) and 70% training data

In figure 4.42, the training, testing and validation curves are very similar and no significant overfitting has occurred. 1000 epochs indicates that the training stopped at iteration 1000 and at iteration 1000 where the best validation performance occurs and error is minimum.

Table 4.10: Summary of Quasi Newton algorithm

Algorithm	Hidden Layer	Hidden Neuron	Training Data	Regression Value	Error Range
Quasi Newton	1	10	70%	0.695	-100.4 to 67.13
	2	10-5	70%	0.820	-75.15 to 78.95
	2	10-8	70%	0.867	-74.78 to 85.35.

Table 4.10 shows that in Quasi Newton, two hidden layer (6-10-8-1) and 70% training, 15% testing and 15% validation data set given the higher regression value and lower error range.

Table 4.11: Error testing of Scaled Conjugate Gradient algorithm

Sample No.	Targets	Predicted	Error (%)	Predicted	Error (%)	Predicted	Error (%)
		For 6-10-1 and 70% training data		For 6-10-8-1 and 70% training data		For 6-10-5-1 and 70% training data	
1	6.3	8.9	41.26	5	20.63	4.8	23.80
2	109.8	90.12	17.92	125.12	13.95	130.12	18.50
3	263.3	357.92	35.93	297.92	13.15	309.92	17.70
4	14.78	9.45	36.06	10.45	29.29	10.45	29.29
5	163.49	193.3	18.23	140.3.67	14.18	185.3	13.34
6	49.6	38.8	21.77	55.13	11.15	41.8	15.72
7	47.66	38.1	20.05	41.09	13.74	39.1	17.96
8	250.4	277.78	10.93	255.78	2.16	260.8	4.16
9	19.62	13	33.74	15.1	22.93	15	23.54
10	15.81	23.2	46.74	11.3	28.52	11.2	29.15
11	218.24	274.49	25.77	260.5	19.35	267.49	22.56
12	34.13	25.2	26.16	27.19	20.33	27.2	20.30
13	105	130.84	24.60	99.79	4.91	110.84	5.56
14	5.7	3.99	30	4.21	26.31	4.02	29.47
15	27.62	37.25	34.86	30.9	11.87	31.25	13.14
16	31.05	22.05	28.98	26.1	15.72	26.05	16.10
17	123	148.4	20.65	135.4	10.08	135.4	10.08
18	17.77	12.5	29.65	13.01	15.53	13.2	25.71
19	19	23.09	21.52	17.09	10.05	21.09	11
20	251.78	210.76	16.29	215.76	14.30	216.76	13.90
21	161.06	129.22	19.76	135.22	16.04	132.22	17.9
22	20	14.3	28.5	15.7	21.3	25.8	29
23	37.4	44.43	18.79	35.2	10.61	43.3	16.12
24	120.5	101.3	15.93	135.7	16.59	105.3	12.61
25	45.2	33.8	25.22	38.3	16.59	53.8	19.02
		Avg error	25.97	Avg error	15.8	Avg error	18.22

Table 4.11 shows the error and average error of Quasi Newton training algorithm. This table shows the average error of Scale Conjugate Gradient with two hidden layer (6-10-8-1) and 70% training is 15.8, two hidden layer (6-10-5-1) and 70% training is 18.22 and one hidden layer (6-10-1) and 70% training data is 25.97. This thesis experiments more with changing hidden layer and training data but this table shows only best three.

Table 4.12: Accuracy testing of Scaled Conjugate Gradient algorithm

Hidden Layer with Neuron and Training data	Average Error (%)	Accuracy (%)
For 6-10-1 and 70% training data	25.97	74.03
For 6-10-8-1 and 70% training data	15.8	84.20
For 6-10-5-1 and 70% training data	18.22	81.78

From Table 4.11 we get the average error and Table 4.12 shows the accuracy of Quasi Newton training algorithm, the accuracy of two hidden layer (6-10-5-1) with 70% training data is 81.78, one hidden layer (6-10-1) with 70% training data is 74.03, two hidden layer (6-10-8-1) with 70% training data is 84.20 and this is the best accuracy of Quasi Newton training algorithm.

4.6 COMPARISON OF RESULTS

Figure 4.43 shows the regression dataset of Levenberg marquardt, Bayesian Regularization and Scaled conjugate grediant and Quasi Newton Algorithm. The regression value of Levenberg Marquardt algorithm is 99.76, Bayesian Regularization is 0.9796, Scaled Conjugate is 0.946 and Quasi Newton is 0.86.

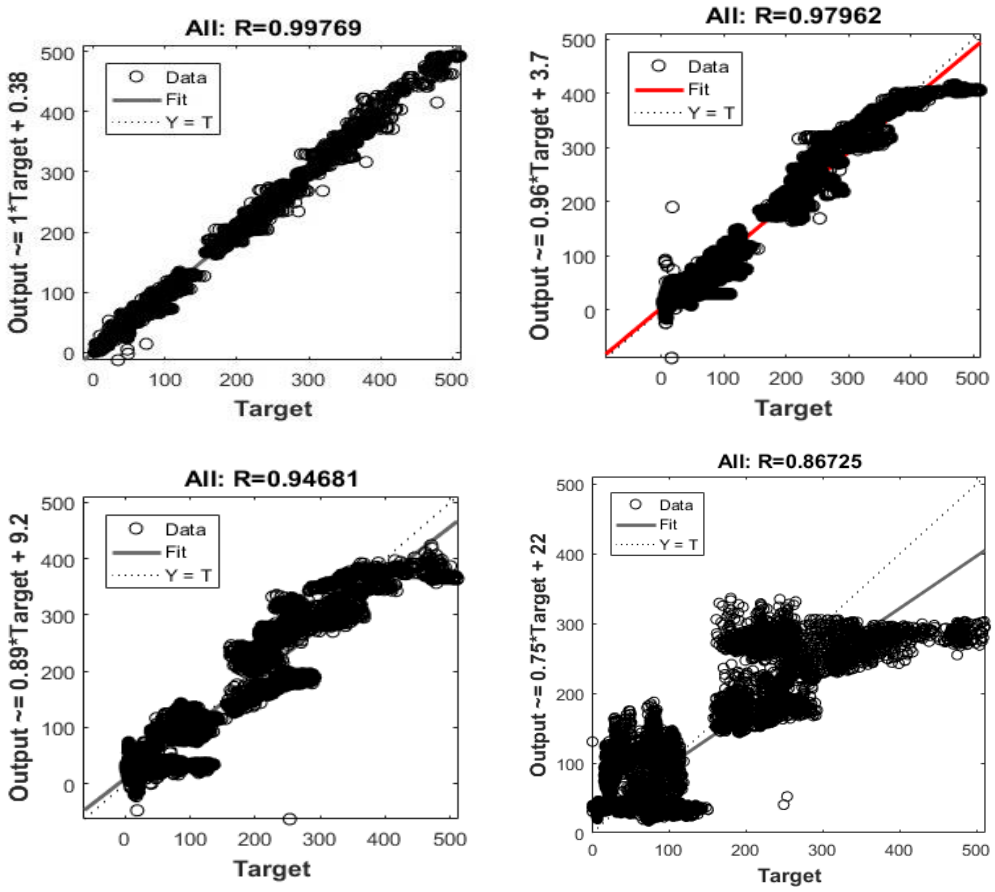

Figure 4.43: Comparison of regression value of these four algorithm

In figure 4.44 shows the comparison of three training algorithms error histogram. In Levenberg Marquardt training algorithm maximum error lies in -5.79 to 5.26. In Bayesian Regularization training algorithm maximum error lies in -26.56 to 15.15. In Scaled Conjugate Grediant that is -31.78 to 31.45 and in Quasi Newton training algorithm maximum error lies in -49.48 to 53.27. Above all, less error have been found from Levenberg Marquardt Algorithm.

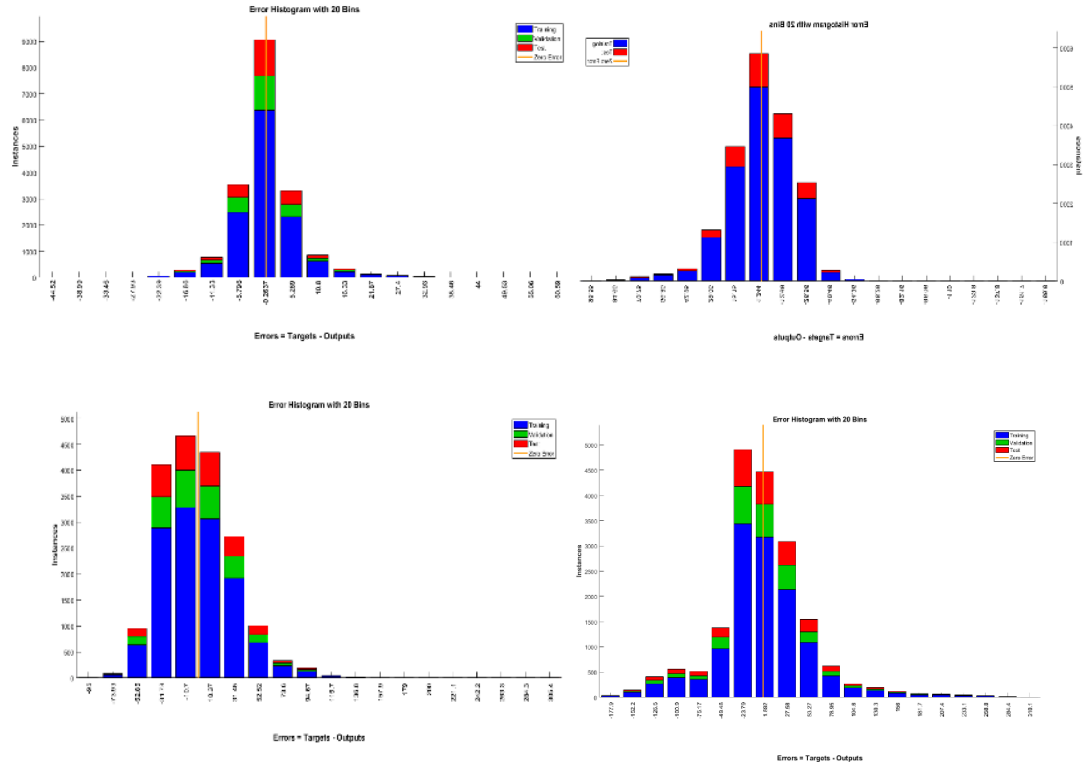

Figure 4.44: Comparison of error histogram of these four algorithm

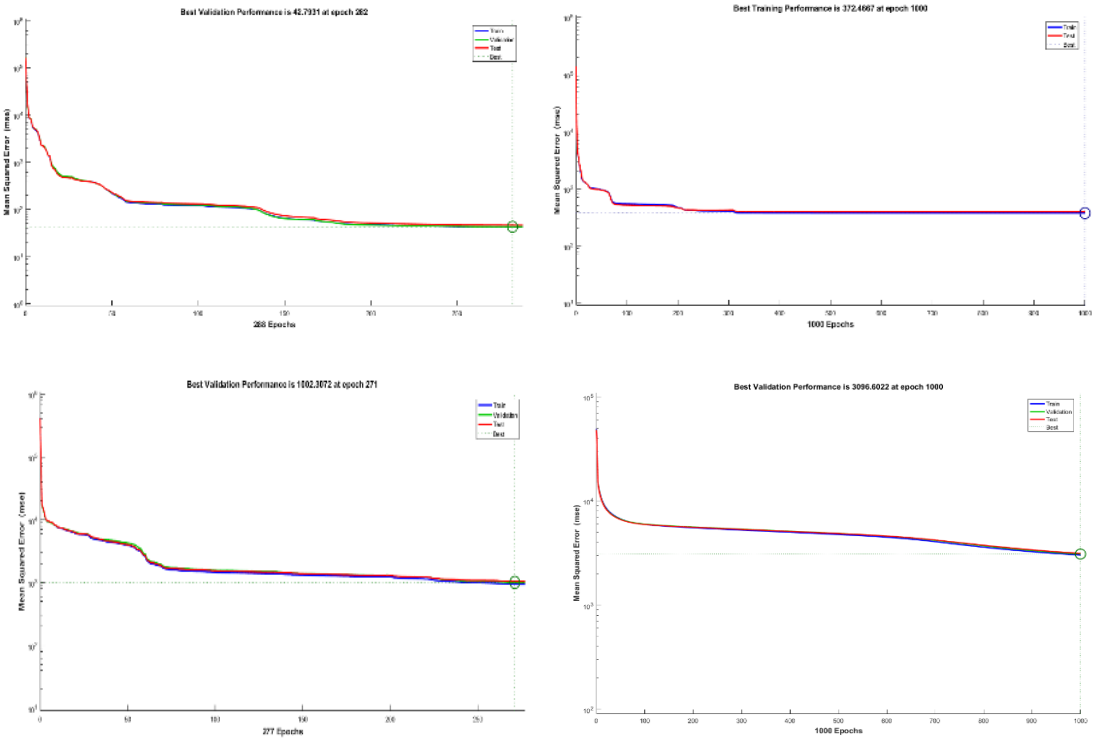

Figure 4.45: Comparison of Epochs vs MSE of these four algorithm

Figure 4.45 show the comparison of Epochs Vs Mean Square Error. All these four the training, testing and validation curves are very similar and no significant overfitting has been occurred.

Table 4.13: Best accuracy comparison among four algorithm

Algorithm	Levenberg-Marquardt	Bayesian Regularization	Scaled Conjugate Gradient	Quasi Newtron
Accuracy	95.64	92.08	88.90	84.20

4.7 ANALYZING RESULT

This thesis, work with Feed-Forward Backpropagation model using Levenberg Marquardt, Bayesian Regularization , Scaled Conjugate and Quasi Newton training algorithm for predicting the financial market stock price. The prediction of financial market is necessary because the investors who invest in the company are usually unaware about the behaviour of stock market. This thesis work will help to the investors to understand the future market situations. In the above result and discussion section, the result is find out that the Levenberg Marquardt algorithm provides better accuracy than Bayesian Regularization and Scaled Conjugate training algorithm on the financial market stock price prediction. The accuracy of Levenberg Marquardt algorithm is 95.64, Bayesian Regularization algorithm is 92.08, Scaled Conjugate algorithm accuracy is 88.90 and Quasi Newton is 84.20. In this work, the split ratio of training and testing and validation data sets is 70:15:15, the resultant output is found that Levenberg Marquardt training algorithm accuracy is more than others.

4.8 SUMMERY

The prediction of financial stock market is a challenging task. The investors who invest in the stock markets are usually unaware about the behavior of stock market. For solving this problem, this thesis work provides an efficient prediction of financial stock market. Among Levenberg Marquardt, Bayesian Regularization, Scaled Conjugate model

and Quasi Newton , in the above result showed that Levenberg Marquardt algorithm provides better accuracy than others model. Using this process, investors can easily understood about the behaviour of financial stock market.

CHAPTER 5

CONCLUSION

This research focus on the use of supervised learning techniques to enhance the prediction accuracy of stock market. It is clear that stock market price can be predicted and from the predicted values and accuracy rate it is also clear that it can be a very useful tool for the people who are related to the stock market. In this research, all the experiment is based on real dataset and Matlab software is used for experiments. Chapter of Introduction of this report have discussed background, problem statement, objectives and scope of the study. The main objective of this study is to predict future stock price with best accuracy. Chapter of Literature Review have discussed previous works which are related to this work and motivation of this study. Chapter of Methodology consists of models, datasets and accuracy of the study. Models and related algorithms are mentioned at methodology section. Result and Discussion section have discussed the results of the different algorithms.

By using Artificial neural network any frequent data from stock market can be trained, tested and then the trained network can be used for predicting the stock market price. This research work used multilayer feedforward neural network and four instinct training algorithm to train these data. Though it is not possible to predict the actual price from future but the predicted price which is almost close to the real value can be very useful for a trader who want to make a good profit from stock market. We collected and analyzed top fifteen companies six years data of Dhaka Stock Exchange and have got best accuracy.

REFERENCES

[1] G. Grudnitzky and L. Osburn, "Forecasting S&P and Gold Futures Prices: An Application of Neural Networks," Journal of Futures Markets, vol. 13, No. 6, pp. 631-643, September 1993.

[2] Sujin Pyo, Jaewook Lee, Mincheol Cha, and Huisu Jang. "Predictability of machine learning techniques to forecast the trends of market index prices: Hypothesis testing for the korean stock markets". PloS one, 12(11):e0188107, May 2017.

[3] M. Pekkaya and C. Hamza A_gebi, "An application on forecasting exchange rate by using neural network". Congress of YAEM, 27, Jun 2007.

[4] Ishmat Pasha, "Stock Market Price Prediction Using Artificial Neural Network". December 2016.

[5] Md. Tawhid Anwar and Saidur Rahman, "Forecasting Stock Market Prices Using Advanced Tools of Machine Learning", April 2019.

[6] K. Tsai and J. Wang, "External technology sourcing and innovation performance in LMT sectors", Research Policy, vol. 38, no. 3, pp. 518-526, August 2015.

[7] J. S. Otteorbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E. "Unsupervised Machine Learning on a Hybrid Quantum Computer", March 2017.

[8] V Kranthi Sai Reddy, "Stock Market Prediction Using Machine Learning", 2018.

[9] K. Senthamarai Kannan, P. Sailapathi Sekar, M.Mohamed Sathik and P. Arumugam. "Financial Stock Market Forecast using Data Mining Techniques", May 2016.

[10] Ayodele A. Adebiyi., Aderemi O. Adewumi, "Stock Market Prediction Using Machine Learning", 2018.

[11] Kai Chen, Yi Zhou, Fangyan Dai, "A LSTM-based method for stock returns prediction: A case study of China stock market", June 2015.

[12] Shunrong Shen, Haomiao Jiang, Tongda Zhang. "Stock Market Forecasting Using Machine Learning Algorithms", November 2018.

[13] Mahdi Pakdaman Naeini, Hamidreza Taremian and Homa Baradaran Hashemi. "Stock Market Value Prediction Using Neural Networks". 2016

[14] Younuf Yetis, Halid Kaplan and Mo Jamshidi, "Stock Market Prediction by Using Artificial Neural Network",IEEE, DOI: 10.1109/WAC.2014.6936118, 2014.

[15] http://www.dse.com [Accessed on August 15, 2020]

[16] http://www.investing.com [Accessed on September 13, 2020]

[17] Fischer, Thomas, and Christopher Krauss. “Deep learning with long short-term memory networks for financial market predictions”. European Journal of Operational Research 270.2: 654-669, 2018.

[18] Attigeri, Girija V., et al. "Stock market prediction: A big data approach." TENCON 2015-2015 IEEE Region 10 Conference. IEEE, 2015.

[19] Kirkpatrick II, Charles D., and Julie A. Dahlquist. “Technical analysis: the complete resource for financial market technicians”. FT press, Jun 2010.

[20] Saad, Emad W., Danil V. Prokhorov, and Donald C. Wunsch. “Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks” IEEE Transactions on neural networks 9.6 : 1456-1470, 1998.

[21] Vlasenko, Alexander, et al. “A Novel Ensemble Neuro-Fuzzy Model for Financial Time Series Forecasting”. Data 4.3: 126, 2019.

[22] M. Nielsen. “Neural Networks and Deep Learning”, December 2014.

[23] H. Harry, “Performance Evaluation of Artificial Neural Networks in the Foreign Exchange Market,” M.S. thesis, Dept. Math., KTH Univ., Sthlm, Sweden, 2012.

[24] G. Carlos. “Artificial Neural Networks for Beginners.” November 30, 2020.

APPENDICES

```
%% Feed_Forward_Multilayer_NN_ with_ all_features
% Solve an Input-Output Fitting problem with a Neural Network
% Dataset
%% This script assumes these variables are defined:
 % houseInputs - input data. % houseTargets - target data.
% Or [inputs,targets] = house_dataset;
x=inputs;
x=multiple_inputs;
x=sample_inputs;
t=target; % Or
t=sample_target;
t=single_target;
t=double_targets;
t=hexa_targets;
t=four_targets;
%Levenberg-Marquardt algorithm (trainlm) for training is the default algorithm
% trainFcn='trainlm'; % for Levenberg-Marquardt algorithm
% or trainFcn='trainlm';
%% or we can also use
% trainFcn='trainbr'; for Bayesian Regularization algorithm
trainFcn='trainscg'; for Scaled Conjugate Gradient algorithm
% trainFcn='trainbr'; for Bayesian Regularization
% trainFcn='trainbfg'; for BFGS Quasi-Newton
% trainFcn='trainrp'; for Resilient Backpropagation
% trainFcn='trainscg'; for Scaled Conjugate Gradient
% trainFcn='traincgb'; for Conjugate Gradient with Powell/Beale Restarts
% trainFcn='traincgf'; for Fletcher-Powell Conjugate Gradient
% trainFcn='traincgp'; for Polak-Ribiére Conjugate Gradient
% trainFcn='trainoss'; for One Step Secant
% trainFcn='traingdx'; for Variable Learning Rate Gradient Descent
```

```
% trainFcn='traingdm'; for Gradient Descent with Momentum
% trainFcn='traingd'; for Gradient Descent
% Create a Fitting Network
hiddenLayerSize = [10];
% feedforward network, with a sigmoid transfer function in the ....
% hidden layer and a linear transfer function in the output layer
% The default network for function fitting (or regression) problems,
% is a feedforward network with the default tan-sigmoid
% transfer function in the hidden layer,
% and a softmax transfer function in the output layer.
net = fitnet(hiddenLayerSize,trainFcn);
% or net = feedforwardnet(hiddenLayerSize,trainFcn);
% or net = feedforwardnet(10,'trainlm');
view(net);
% Configuration to examining the input and target data
net1 = configure(net,x,t)
% net1 = configure(net,x,t); % takes input data x and target data t
% net1 = configure(net,x); % configures only inputs.
% net1= configure(net,'inputs',x,i); % configures the inputs specified with the index
vector i.
% If i is not specified all inputs are configured.
% net1= configure(net,'outputs',t,i); % configures the outputs specified with the index
vector i.
% If i is not specified all targets are configured.
view(net1);
% To set weights and bias in the first layer
% net.IW{1,1}=[1 1];
% net.b{1}=0;
% Set up Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
```

```
% Training occurs according to trainlm training parameters, shown
% here with their default values;you can change these values
net.trainparam.epochs=5000;
net.trainparam.goal=0;
net.trainparam.lr=0.01;
net.trainParam.max_fail=6;
net.trainParam.min_grad=1e-7;
net.trainParam.mu=0.001;
net.trainParam.mu_dec=0.1;
net.trainParam.mu_inc=10;
net.trainParam.mu_max=1e10;
net.trainParam.show=25;
net.trainParam.showCommandLine=false;
net.trainParam.showWindow=true;
net.trainParam.time=inf;
% To view the layers subobject for the first layer with the command
net.layers{1}
% for input layer {1}={0}; as follow
%net.layers{0};
%% for 2nd hidden layer {2}={0}; and so on
net.layers{2};
% If you wanted to change the transfer function to logsig, for example,
% net.layers{1}.transferFcn = 'logsig'; % and similarly for other layers;
% To view the layerWeights subobject for the weight between layer 1 and layer 2,
net.layerWeights{2,1}
%% Train the Network
[net,tr] = train(net,x,t);
% or net=train(net,inputs,targets);
% or [net,tr] = train(net,x,t);
% View the Network
view(net);
% Test the Network or to calculate the network response to any input.
outputs=net(x);
```

```
% or y = net(x);
% or outputs = net(inputs(:,5));
errors = gsubtract(t,outputs);
e=t-outputs;
% or % e = gsubtract(targets,outputs);
% or % e = gsubtract (t,y);
% or % e=t-y;
performance = perform(net,t,outputs)
% or performance = perform(net,t,y)
%% Plots
% Uncomment these lines to enable various plots.
figure, plotperform(tr); % or figure, plotperf(tr)
figure, plottrainstate(tr);
% figure, plotfit(targets,outputs); % It will work only for single input parameter
% or figure, plotfit(net,x,t)
figure, ploterrhist(errors);
% or figure, ploterrhist(e)
figure, plotregression(t,outputs,'Regression');
% or figure, plotregression(t,y)
% or to generates multiple plots.
Outputs = net(inputs);
trOut = Outputs(tr.trainInd);
vOut = Outputs(tr.valInd);
tsOut = Outputs(tr.testInd);
trTarg = targets(tr.trainInd);
vTarg = targets(tr.valInd);
tsTarg = targets(tr.testInd);
plotregression(trTarg,trOut,'Train',vTarg,vOut,'Validation',...
tsTarg,tsOut,'Testing',targets,Outputs,'Overall')
% It is also possible to calculate the network performance
% only on the test set
tInd = tr.testInd;% tr.trainInd, tr.valInd and tr.testInd contain the indices
% (number)of the data points that were used in the training,
```

```
% validation and test sets, respectively.
tstOutputs = net(inputs(:,tInd));
tstPerform = perform(net,targets(tInd),tstOutputs);
% Train and Apply Multilayer Neural Networks to calculate the network response to
any input
y= net(Sample_inputs); % Or
% a = net(inputs(:,5));
% or y=net(samples);
% or outputs = net(inputs(:,5));
% or a=net(new);
% or sampleOutput= Outputs(tr.newsampleInd);
%% or sampleTarget = targets(tr.newsampleInd);
T=Sample_target;
e=T-y;
% To check the training record,tr
% tr
%% If the network is not sufficiently accurate, you can try
% initializing the network and the training again.
%net = init(net);
%net = train(net,inputs,targets);
```