UNIVERSITY OF BELGRADE
FACULTY OF MECHANICAL ENGINEERING

Ali Karkara A. Diryag

MACHINE LEARNING IN INTELLIGENT ROBOTIC SYSTEM

Doctoral Dissertation

Belgrade, 2015

УНИВЕРЗИТЕТ У БЕОГРАДУ
МАШИНСКИ ФАКУЛТЕТ

Ali Karkara A. Diryag

МАШИНСКО УЧЕЊЕ ИНТЕЛИГЕНТНОГ РОБОТСКОГ СИСТЕМА

Докторска дисертација

Београд, 2015.

EXAMINATION COMMITTEE

Supervisor:	Dr. Zoran Miljković Full Professor University of Belgrade-Faculty of Mechanical Engineering
Members:	Dr. Bojan Babić Full Professor University of Belgrade-Faculty of Mechanical Engineering
	Dr. Dragan Aleksendrić Associate Professor University of Belgrade-Faculty of Mechanical Engineering
	Dr. Marko Mitić Assistant Research Professor University of Belgrade-Faculty of Mechanical Engineering
	Dr. Mirko Đapić Associate Professor University of Kragujevac-Faculty of Mechanical and Civil Engineering in Kraljevo
Date of defence:	September 2015

Комисија за оцену и одбрану дисертације:

Ментор:	др Зоран Миљковић, редовни професор Универзитет у Београду - Машински факултет
Чланови комисије:	др Бојан Бабић, редовни професор Универзитет у Београду - Машински факултет
	др Драган Алексендрић, ванредни професор Универзитет у Београду - Машински факултет
	др Марко Митић, научни сарадник Универзитет у Београду - Машински факултет
	др Мирко Ђапић, ванредни професор Факултет за машинство и грађевинарство у Краљеву, Универзитет у Крагујевцу
Датум одбране:	Септембар 2015.

DEDICATION

To my: Parents, Wife and Kids Moutasim, Yaken, Moutaz

Acknowledgment

I would like to express my deep appreciation and gratitude to my supervisor Professor Dr. Zoran Miljković for his guidance and advises that made this work possible.

I want also like to sincerely thank Dr. Marko Mitić for his participation in main experiments described in this thesis. His advices, coding help, dedication and valuable assistance are gratefully acknowledged. Through my research work, he was always ready to help me, guide me and listen to my problems. Without him, this PhD dissertation certainly would not be finished.

My sincere appreciation, thanks and gratitude to the academic staff members of my faculty, I consider you to be all my friends.

A special thanks goes to the examination committee for reading the thesis, and for giving comments and suggestions on my work.

My sincere appreciation, thanks and gratitude to the Libyan General People's Committee of higher education which gave me scholarship and have supported me to my further education.

MACHINE LEARNING IN INTELLIGENT ROBOTIC SYSTEM

Abstract

Nowadays, one of the most desirable features of every robotic system is the ability to adapt to the real world changing conditions. Similarly, failure prediction is equally important in different manufacturing environments in which repairs are often infeasible and failures can have disastrous consequences. In industrial robotics, failure prediction is helpful in reduction of a system down-time by identifying and repairing faulty components. Also, the reliability of a product manufacturing and increased human safety is ensured by implementing fault tolerance and failure prediction unit in the robotic system.

It is known that the supervision and learning of robotic executions is not a trivial problem. In the 21st century, robots must be able to tolerate and predict internal failures in order to successfully continue performing their tasks. This doctoral dissertation presents a novel approach for prediction of robot execution failures based on machine learning technique - neural networks (NNs). Real data consisting of robot forces and torques recorded immediately after the system failure are used for the NN training. Two types of neural networks are used: feedforward and recurrent (Elman) NNs. In total, 7 different learning algorithms and 24 NN architectures are implemented in order to find optimal solution for the problem of robot execution failures prediction. Each multilayer feedforward NN with different learning algorithm and architecture that consists of 1, 2, 3, or 4 hidden layers is evaluated several times, and the same NN architectures are trained using Elman recurrent NN. Experimental results indicate that Bayesian Regularization algorithm is the best choice for the prediction problem with prediction rate of 95.4545 percent, despite having the

erroneous or otherwise incomplete sensor measurements invoked in the dataset. The experimental results show that the NN outperforms state-of-the-art algorithms, such as the Naïve Bayes, Decision Trees and Support Vector Machine based algorithms employed for the prediction of robot execution failures.

Additionally, two independent failure prediction problems are treated in this dissertation. Several experiments in real time are conducted on an real nonholonomic mobile robot *Khepera II* in a laboratory model of manufacturing environment.

First real world failure problem refers to the robot obstacle detection in indoor environment. Six infrared sensors mounted on the mobile robot are used to obtain information of the obstacle located left and right from the platform. Randomly generated failed sensor data is integrated into the training set so as to test the NN performance in this task. The result show that in over 96 percent of all tested cases NN recognized failed value, meaning that the obstacle location is successfully determined after the failed information is replaced with the expected one.

Second real world problem refers to the failure prediction in a mobile robot trajectory tracking problem. Two independent trajectories are employed so as to objectively test the proposed intelligent approach. The tracking of the *M-shaped* and *Labyrinth-type* trajectories showed as a fairly easy task for the developed prediction method. In more than 99 percent of the cases, the neural network predicted the wheel command failure, which is next replaced with the desired value in order to successfully track chosen trajectory. The experiments show that a mobile robot can track desired trajectories with a minimal error in every control iteration, which evidence the robustness and the applicability of the proposed approach.

Finally, all aforementioned experiments and obtained results indicate that the new method based on neural networks can successfully be applied for robot failure prediction, and also that novel neural network based control

system of the mobile robot can be successfully used for solving obstacle detection and trajectory tracking problems in laboratory model of a manufacturing environment.

Keywords: Robotic system, Machine learning, Neural networks, Failure prediction, Obstacle detection, Trajectory tracking, Real world mobile robot control

Scientific field:

Technical science, Mechanical engineering

Narrow scientific field:

Production Engineering

UDC: **004.896:004.85(043.3)**
004.032.26(043.3)
007.52:519.712(043.3)

TABLE OF CONTENT

1. Introduction and motivation 1

1.1 Overview of machine learning in intelligent robotic systems 5
1.1.1 Applications of machine learning to robotics 6
1.1.2 Types of machine learning 8

1.2 Significance of soft computing techniques in domain of robot control and failure prediction 11
1.2.1 Importance of soft computing in robot control 13
1.2.2 Robust soft computing in the failure domain 15

1.3 Organization of the dissertation 16
1.3.1 Main contributions of the dissertation 18

2. Research objectives and approach................................. 20

2.1 Overall objective.. 20

2.2 Specific objective.. 21
2.2.1 Scientific dissertation methods and approach 22

3. State-of-the-art review.. 24

3.1 Problem description.. 24

3.2 Prediction of industrial robot execution failures 25
3.2.1 Fault detection in mobile robotics 26
3.2.3 Failure prediction problem in industrial robotics 26
3.2.4 Use of neural networks in prediction of robot failure 27

3.3 Advantages and disadvantages of different approaches 28
3.3.1 Advantages of neural networks 29
3.3.2 Limitations & Disadvantages of neural networks 30

3.4 Applications of soft computing techniques in the prediction domain.... 32

3.4.1 General NN application 33

3.4.2 NN application with noise data 33

3.4.3 Neural computing and output calculation 34

4. Robotic failure prediction in *MATLAB®* and *BPnet* software 36

4.1 Introduction to neural networks **36**

4.1.1 Biological Neurons 37

4.1.2 Artificial Neurons 38

4.1.3 Characteristics of NNs 40

4.1.4 Operation of NNs 48

4.2 Failure data description **49**

4.2.1 Robot execution data 49

4.2.2 Failure Dataset classes 50

4.3 Prediction algorithms, activation functions and neural network architectures **51**

4.3.1 Neural networks training algorithms 51

4.3.2 Activation function 56

4.3.2 Neural network architectures 61

4.3.4 Neural Network Topology for Modeling Approach of robotic systems 65

4.4 Neural networks training procedure in *MATLAB®* software **70**

4.4.1 Feedforword neural network training procedure 70

4.4.2 Elman neural network training procedure 75

4.5 Neural networks training procedure in BPnet software **76**

4.5.1 Training set for BPnet –software 78

4.5.3 Prediction procedure summary 82

5. Intelligent mobile robot in a manufacturing environment 83

5.1 Introduction to intelligent manufacturing systems (IMS) **83**

5.1.1 Components of an intelligent manufacturing system 84

5.1.2 Intelligent Mobile Robots in manufacturing systems 85

5.2 Control of a mobile robot using AI techniques 86

5.2.1 Neural networks for obstacle detection and avoidance 87

5.2.2 NNs for trajectory tracking and control 87

5.2.3 NNs Control Methodology 89

5.3 Localization of a mobile robot in a laboratory model of manufacturing environment 90

5.3.1 Odometry and mobile robotics 91

5.3. 2 Sensors for Localization 94

5.3.3 Approach employed in this PhD dissertation 96

5.4 Mobile robot obstacle detection: problem and solution 99

5.4.1 Robot Obstacle detection problem based failure prediction 101

5.5 Mobile robot trajectory tracking: problem and solution 103

5.5.1 M-shaped trajectory tracking based failure prediction 103

5.5.2 Labyrinth-type trajectory tracking based failure prediction 106

6. Experimental study: prediction of robot failures using neural networks 110

6.1 Experimental setup 110

6.1.1 Algorithm implementation (setup) 111

6.2 Experimental results... 111

6.2.1 Prediction Performance Using LM and BR algorithm 112

6.2.3 Prediction performance using BFG and GDX algorithms 122

6.2.4 Overall performance of NNs in prediction task 127

6.2.6 Feedforward & Elman neural network comparison results 130

6.3 Experimental results in BPnet software 132

7. Conclusions and future work 135

7.1 Conclusions 135

7.2 Recommendations for future work 138

References 139

Biography 152

Appendix 153

List of Figures

FIGURE 1 : BLOCK DIAGRAM OF SUPERVISED LEARNING [33] 9

FIGURE 2 : BLOCK DIAGRAM OF UNSUPERVISED LEARNING [33] 9

FIGURE 3: BLOCK DIAGRAM OF REINFORCEMENT LEARNING [33] 10

FIGURE 4: THE GENERAL CONTROL SCHEME FOR MOBILE ROBOT NAVIGATION [41] 13

FIGURE 5 : SCHEMATIC DRAWING OF BIOLOGICAL NEURONS 38

FIGURE 6 : ARTIFICIAL NEURON INTERNAL REPRESENTATION 39

FIGURE 7 : VARIOUS CLASSIFICATIONS OF ARTIFICIAL NEURAL NETWORKS [86] 43

FIGURE 8 : SIMPLE PERCEPTION NEURON AND STEP TRANSFER FUNCTION [86] 44

FIGURE 9 : TWO LINEALLY SEPARABLE CLASSES [86] 45

FIGURE 10 : TWO LINEARLY SEPARABLE CLASSES [86] 46

FIGURE 11 : EXCLUSIVE-OR (XOR) PROBLEM [86] 47

FIGURE 12 : 90 DIFFERENT FEATURES OF ONE INSTANCE (I.E. F&T VALUES) 49

FIGURE 13 : AN EXAMPLE OF FORCE AND TORQUE VALUE IN ONE DATASET INSTANCE: 50

FIGURE 14 : SUPERVISED TRAINING OF NNS [101] 56

FIGURE 15 : 2D GRAPHICAL OF COMMON ACTIVATION FUNCTION...... 58

FIGURE 16 : BACKPROPAGATION NEURON USING SIGMOID TRANSFER FUNCTION [92] 59

FIGURE 17 : PERCEPTRONS [110] 63

FIGURE 18 : MULTI LAYER PERCEPTRON [108] 64

FIGURE 19 : STEPS OF MODELING AND IDENTIFYING FOR MOBILE ROBOT SYSTEM 65

FIGURE 20 : FEED FORWARD NEURAL NETWORKS 67

FIGURE 21 : TWO LAYER FEEDBACK-RECURRENT NEURAL NETWORK [112] 68
FIGURE 22 : ELMAN RECURRENT NETWORK [112] 69
FIGURE 23 : MATLAB TRAINING NETWORKS 75
FIGURE 24 : BPNET SOFTWARE: (A) STARTING WINDOW, (B) "CONFIGURATION" MODULE, (C)"CONNECTIONS" MODULE, (D) TRAINING PROCESS [5] 78
FIGURE 25 : BPNET SOFTWARE TRAINING CONTROL PANEL 80
FIGURE 26 : STRUCTURE OF THE MULTILAYER FEEDFORWARD NN [5] . 81
FIGURE 27 : COMPONENTS OF AN INTELLIGENT MANUFACTURING SYSTEM [117]. 85
FIGURE 28 : THE POSITION OF THE MOBILE ROBOT IN THE PLANE 93
FIGURE 29 : LABORATORY MODEL OF MANUFACTURING ENVIRONMENT 97
FIGURE 30 : ROBOT – COMPUTER CONNECTION CONFIGURATION [154] 99
FIGURE 31 : OBSTACLE DETECTION WITH IR SENSORS 100
FIGURE 32 : ROBOT POSITIONS IN THE DETECTION EXPERIMENT: (A) OBSTACLE ON THE LEFT SIDE, (B) OBSTACLE ON THE RIGHT SIDE [5]. 101
FIGURE 33 *M*-SHAPED TRAJECTORY TRACKING EXPERIMENT [141] 105
FIGURE 34 : REAL WORLD EXPERIMENT IN A MANUFACTURING ENVIRONMENT USING KHEPERAII MOBILE ROBOT [141] 106
FIGURE 35 : RESULT OF LABYRINTH-TYPE OF TRAJECTORY TRACKING EXPERIMENTS 108
FIGURE 36 : MOBILE ROBOT TRACKING THE LABYRINTH-TYPE TRAJECTORY: (A)-(T) ROBOT POSES IN CHARACTERISTIC CONTROL ITERATIONS [5] 109
FIGURE 37 : NN TESTING RESULTS: (A) LM ALGORITHM, (B) BR ALGORITHM 112

FIGURE 38 : MEAN-SQUARE ERROR PERFORMANCE FOR TRAINLM NN [10 8 4 3] 114
FIGURE 39 : REGRESSION OF THE OUTPUTS VS. TARGETS FOR THE NETWORK [10 8 4 3].. 115
FIGURE 40 : MEAN-SQUARE ERROR PERFORMANCE FOR TRAINBR NN [8 5 4 3].. 116
FIGURE 41 : REGRESSION OF THE OUTPUTS VS. TARGETS FOR THE NETWORK [8 5 4 3].. 116
FIGURE 42 : NN TESTING RESULTS: (A) RP ALGORITHM, (B) SCG ALGORITHM.. 117
FIGURE 43 : MEAN-SQUARE ERROR PERFORMANCE FOR TRAINRP ANN [8 5 4 3] 119
FIGURE 44 : REGRESSION OF THE OUTPUTS VS. TARGETS FOR THE NETWORK [8 5 4 3].. 120
FIGURE 45 : MEAN-SQUARE ERROR PERFORMANCE FOR TRAINSCG ANN [8 5 4 3].. 121
FIGURE 46 : REGRESSION OF THE OUTPUTS VS. TARGETS FOR NN [8 5 4 3] 121
FIGURE 47 : NN TESTING RESULTS: (A) BFG ALGORITHM, (B) GDX ALGORITHM.. 122
FIGURE 48 : MEAN-SQUARE ERROR PERFORMANCE FOR TRAINBFG NN [10 8 5 4] 123
FIGURE 49 : REGRESSION OF THE OUTPUTS VS. TARGETS FOR THE NETWORK [10 8 5 4].. 124
FIGURE 50 : MEAN-SQUARE ERROR PERFORMANCE FOR TRAINGDX NN [1 1] 126
FIGURE 51 : REGRESSION OF THE OUTPUTS VS. TARGETS FOR THE NETWORK [1 1].. 127
FIGURE 52 : RESULTS OF ELMAN NN [10] TESTING.. 131

FIGURE 53 : ELMAN TRAINING STATES FOR NN $[10]_1$ OF THE SMALLEST MSE 131
FIGURE 55 : NN TESTING IN BPNET SOFTWARE 133
FIGURE 56 : BPNET SOFTWARE TRAINING RESULTS 133

List of Tables

TABLE 1 : NN ARCHITECTURE USED IN EXPERIMENTS IN THIS THESIS. 72
TABLE 2 : LEARNING ALGORITHMS USED IN EXPERIMENTS IN THIS THESIS 73
TABLE 3 : ROBOT CONFIGURATION SPECIFICATIONS 98
TABLE 4 : EXAMPLE OF CORRECT AND FAILED SENSOR INFORMATION 102
TABLE 5 : CORRECT AND FAILED WHEEL COMMENT FOR M-SHAPED TRAJECTORY 104
TABLE 6 : CORRECT AND FAILED WHEEL COMMENT OF LABYRINTH-TYPE TRAJECTORY 107
TABLE 7 : MSE OF LM & BR ALGORITHMS FOR NN ARCHITECTURES ... 113
TABLE 8 : MSE OF RP & SCG ALGORITHMS FOR NN ARCHITECTURES . 118
TABLE 9 : MSE OF GDX & BFG ALGORITHMS FOR NN ARCHITECTURES 125
TABLE 10 : NN PREDICTION RATE IN MATLAB (IN PERCENTAGE) 129
TABLE 11: MAE OF GRADIENT DESCENT BACKPROPAGATION ALGORITHM FOR NN ARCHITECTURES IN BPNET SOFTWARE 134

Nomenclature

AI	Artificial Intelligence
NNs	Neural Networks
ML	Machine learning
ANN	Artificial Neural Network
SC	Soft Computing
SLAM	Simultaneous localization and Mapping
IRS	Intelligent Robotics Systems
LM	Levenberg–Marquardt backpropagation
BR	Bayesian Regularization Backpropagation
RP	Resilence Backpropogation
SCG	Scaled Conjugate Gradient
BFG	BFGS quasi–Newton Backpropagation
GDX	Variable Learning Rate Backpropagation
SLPs	Single Layer Perceptrons
MLP	Multilayer Perceptrons
FNN	Feedforward Neural Networks
FRN	Feedback Recurrent Network
ELM	Elman Networks
BP	BPnet
MSE	Mean Squared Error
MAE	Middle Absolute Error
IMS	Intelligent Manufacturing System
IR	Infrared
WMR	Wheel Mobile Robot

1. Introduction and motivation

Machine learning refers to the process of development of automatic methods for learning, in order to generate predictions or valuable decisions based on determined complex relations. Starting from the late 1990s, it has become a highly successful discipline with applications in many different scientific areas such as robotics. Generally speaking, machine learning today plays crucial role in the formation of versatile, powerful and robust intelligent applications and solutions.

Over the past thirty years, the types of machine learning implementations varied from computational biology to intelligent robotic systems. Moreover, new kind and amount of data influence the development of new techniques. In other words, in order to properly analyze and quantify data, novel machine learning algorithms have been introduced. As a result, new approaches like computational intelligence methods are intensely exploited in research, as well as in industry.

Regarding the aforementioned, this work focuses on the development and implementation of original and advanced machine learning algorithms, specifically applied in the domain of intelligent and cognitive robotics.

The development of a new generation of industrial robots has significantly contributed to increasing the efficiency of the production system, simultaneously reducing the burden of production workers. The use of robots is conducted for those technological tasks in which the presence of manufacturing workers is dangerous, or in case when constant repetition of the same actions leads to a drop in the workers' concentration. Previous experience in industrial robotization for various technological tasks indicate the viability of this approach, given that the introduction and installation of the robots takes care of the humanization of work and that it increases the efficiency of technological systems.

At the beginning of the new century, there is a fundamental paradigm shift in the field of robotics as a scientific and technical discipline that is based on the use of mobile robots. Today's robots have the opportunity to interact with working environment with the use of appropriate sensors for data acquisition and processing the obtained information. At the same time, with the development of science and technology, there are robotic systems use a variety of techniques of artificial intelligence when comes to processing sensory information and identification of the response from technological environment.

Therefore, it is crucial to provide smooth operation of robotic systems in a production facility. Changes in the environment, immeasurable disturbances and errors that occur in the subsystems of the robot indicate that it needs to have intelligent control in order to overcome these problems. One example is the existence of ambiguity of the small errors in the positioning of the robot relative to the object, which accumulates over time. It is clear that, in case of exceeding the limits of positioning errors, the system of industrial robot must undergo reprogramming or different engineering organization. Both of these approaches involve shutdown of industrial robots, or sometimes even a redesign of the entire manufacturing cells and lines. In order to avoid these problems and reduce costs evident, it is necessary to apply advanced artificial intelligence techniques in the management and evaluation of the behavior of the robotic systems of different structures.

Given the aforementioned, mobile robots that work in structured or unstructured environment must be able to deal with dynamic changes in that environment. In other case, mentioned unwanted errors in mobile robot behavior are one of the most challenging problems to deal with. One of the solutions for this is the development and implementation of algorithms and techniques to predict abnormal operations of robotic systems. These algorithms are usually based on machine learning, and their goal is to increase the efficiency and reduce the overall cost in product development. The intelligent algorithms for predicting unwanted behavior of robotic systems should be based on soft computing techniques of artificial intelligence with the aim of eliminating the various problems of stochastic nature in the online mode.

In order to facilitate the smooth functioning of the robot in the working environment, it is necessary to develop such subsystems of industrial robot that collects information about the state of the working environment and the state of the robot. It must be able to process the obtained information, perform decision-making and ultimately act in accordance with the derived conclusions. It is vital that sensory information is processed correctly and that the possible unwanted behavior of the robot is detected. Given the complexity of this problem, current robotic systems use advanced machine learning methods so as to recognize occurrence of a particular failure type.

This dissertation refers to the implementation of the soft computing technique of artificial intelligence to detect and predict irregular robotic systems, and also perform intelligent control, navigation, and tracking of the desired trajectory of mobile robot. This study involves development of the systems of artificial neural networks, and also presents comparisons of different methods for training artificial neural networks in order to accelerate the convergence of the original prediction algorithms.

Artificial Intelligence (AI) enhanced systems are systems designed for detecting knowledge in data without human interruptions. One of the most popular techniques in the domain of AI-based prediction of systems' performance are Neural Networks (NNs). NNs are a well-known tool used as a solution for various engineering problems [1]. They can understand the relationship or mapping between input and output variables during the training process using different learning algorithms. The applications of this machine learning method are very diverse: it can be used for prediction of vehicle reliability performance [2] or in education to predict professional movements of graduates [3]. In robotics, this artificial intelligence technique is often applied for control of a mobile robot [4, 5], or a robot manipulator [6, 7]. For failure problems, the NNs are employed in the assembly tasks [8], prediction of failure rates of large number of the centrifugal pumps [9] or in the robust scheme for robot manipulators [10]. However, despite various mentioned applications, the robot failure prediction based on the soft computing methods has not been reported in the literature so far. This dissertation delivers a novel approach using multilayer feedforward neural networks as a solution for this problem, and also presents performance comparison of different learning algorithms and architectures. Given the aforementioned, it is important to stress out motivation for conducting this research study.

• Motivation of dissertation

In today's industry, it is necessary that the industrial robot has the ability to understand and recognize the state of the environment, and the possibility that under certain conditions it independently decide on future actions. In order to carry out unhindered interaction of robots and manufacture environment, the robotic system must have a subsystem for prediction behavior that would allow working in nominal work, in spite of existing defects and disorders. Given the theoretical and experimental conditions for the accomplishments of the complex tasks in the domain of predicting irregular behavior, the selected following motivation directions are used in this dissertation:

• Using methods and machine learning algorithms, industrial robot predicting subsystem can be developed. It must recognize undesired operations of intelligent robots and subsystems in order to correct the behavior of robots, with final goal to continue the smooth operation of robots in online mode.

• Using soft computing techniques of artificial intelligence it is possible to increase the degree of success in predicting robot failures, errors and irregularities in the industrial robotic system;

• Novel intelligent control system for a mobile robot based on artificial neural networks can ensure the detection of obstacles and characteristic structures in the environment. Likewise, it can provide safe mobile robot trajectory tracking within a defined working area in a laboratory model of a manufacturing environment.

1.1 Overview of machine learning in intelligent robotic systems

One of the most challenging fields in the domain of applications of machine learning techniques is robotics. This complex research area is characterized by the direct interaction with a physical world. In recent years, various studies on implementation of machine learning techniques to specific robotic tasks has been presented. The learning techniques used range from rote learning [11, 12, 13, 14, 15] and inductive learning algorithms [16, 17, 18, 19, 20, 21, 22, 23] over analogical reasoning [24] to Explanation Based Learning [25, 26, 27].

Robotics is one of the most challenging applications of Machine Learning techniques. It is characterized by direct interaction with a real world. In recent years several approaches to apply ML to o specific robotics tasks have been published and have been an increasing interest in applying machine learning techniques to robotics. The applications are manipulator as well as mobile system tasks The learning techniques used range from rote learning [11, 12, 13, 14, 15] and inductive learning algorithms [16, 17, 18, 19, 20, 21, 22, 23] over analogical reasoning [24] to Explanation Based Learning [25, 26, 27]. Many of the systems cited above deal with only very specific robotics problems or with simplifications that make the step from a simulated to a real environment very difficult [11]. This is often due to the fact that at the moment ML-techniques and robotics problems do not match very well. Many ideas in machine learning are applied to quite easy 'worlds' only [11].

1.1.1 Applications of machine learning to robotics

The application of ML techniques in real-world robotic applications is currently a topic gaining a lot of interest. It is known that a successful employment of learning techniques on all levels of robot control is not possible without deeply revising the design criteria that are usually underlying the robot control system [28]. In particular, it is necessary to identify both the tasks of the learning system and the tasks of the robot first and to design an architecture being able to host both the learning and the performance components afterwards [29]. Some possible applications of machine learning to robotics are the following [11]:

1. World model and elementary (sensor-based) actions

   a) Learning of object properties (e.g. mass distribution, stable positions, geometry)
   b) Exploration of the current world (e.g. finding known or prototypically represented objects, determining obstacles)

c) Learning of elementary (sensor-based) actions in the world (e.g. collision-free paths, macro- trajectories, hand-eye coordination, acts of actions)
d) Learning of elementary (sensor-based) actions with objects (e.g. reactive execution of a joining task, manipulation of an object)
e) Optimization and refining of certain actions (e.g. trajectories)
f) Learning to recognize/classify states in the internal world model

2. Sensors

a) Learning of classifiers for objects based on image data
b) Learning of sensor strategies/plans, i.e. how to monitor an action to ensure the correct execution or how to determine certain states of the real world

3. Error analysis

a) Learning of error recognition, error diagnosis and error repairing rules

4. Planning

a) Improvement (speed-up) of the planning module (e.g. planning macros, control rules)
b) Learning of domain knowledge (e.g. general planning rules, orders that have to be taken into ac-count in assembly applications)
c) Learning of action rules or plans, i.e. how to solve a (sub) task in principle
d) Learning of couplings between typical task classes and related action plans (e.g. generalized action plan for a set of tasks)
e) Learning at the task level (e.g. which geometrical arrangements/action plans satisfy certain functional specifications).

1.1.2 Types of machine learning

Three main types of machine learning frameworks can be distinguished, namely supervised learning, self-organized or unsupervised learning, and reinforced learning [30]. The supervised and unsupervised learning are sometimes referred to as classification and clustering tasks respectively [31, 32].

a) Supervised machine Learning

In supervised learning, an external teacher, having the knowledge of the environment represents a set of input-output examples for the neural network which may not have any prior knowledge about that environment [32]. When the teacher and the neural network are both exposed to a training vector drawn from the environment, by virtue of built-in knowledge, the teacher is able to provide the neural network with a desired response for that training vector. The network adjusts its weights and thresholds until the actual response of the network is very close to the desired response. The supervised learning requires a teacher or a supervisor to provide desired or target output signals. The difference (error) can then be used to change the network parameters, which results in an improvement in performance [32].

Examples of supervised learning algorithms for neural networks include the perception learning algorithm, delta rule, the generalized delta rule or back-propagation algorithm, and the learning vector quantization algorithm. As shown in Figure 1 [33], neural network response to inputs is observed and compared with the predefined output. The difference is calculated refer as "error signal" and that is feed back to input layers neurons along with the inputs to reduce the error to get the perfect response of the network as per the predefined outputs [33].

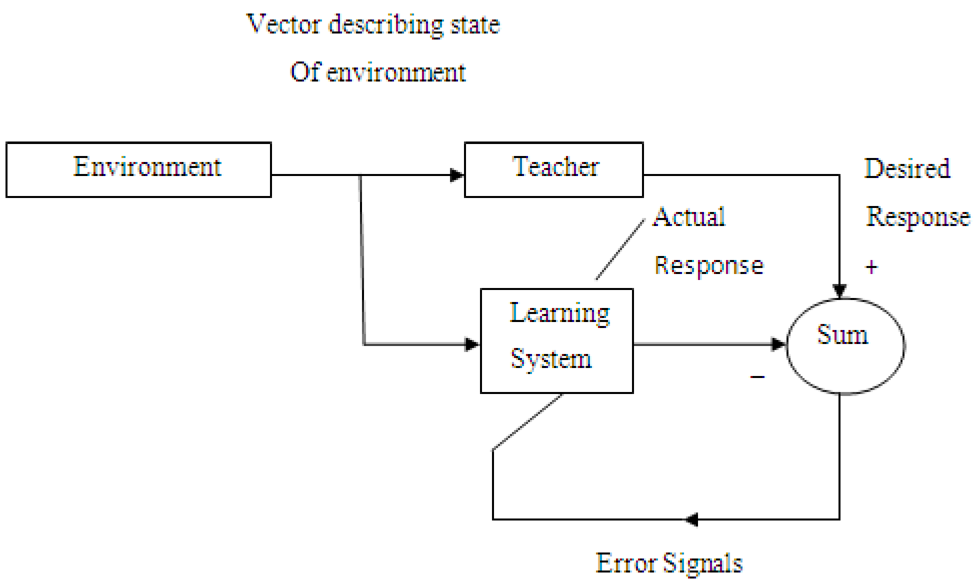


Figure 1 : Block diagram of supervised learning [33]

b) Unsupervised machine Learning

Unsupervised learning has no teacher to guide the system in the right direction, carried out by training vectors with similar properties to produce the same output. The input vectors automatically adjust the weights during training such that input vectors with the similar properties are clustered together. Unsupervised learning includes Kohonen self-organizing maps, k-Means clustering algorithm, adaptive resonance theory, competitive learning algorithms, etc. Main block diagram of this kind of learning is given in Figure 2 [33].

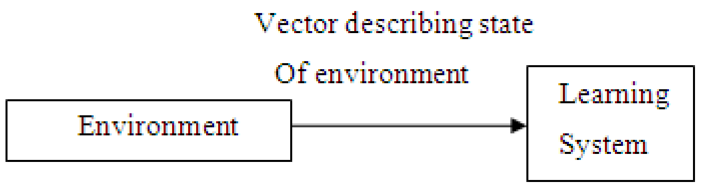


Figure 2 : Block diagram of unsupervised learning [33]

c) Reinforcement machine learning

Reinforcement learning can be described as learning by trial and error. In this, the learning is by interaction whereby an action is performed on the environment and is reinforced by the response (reward) it receives from it. Maximization of the received numerical reward signal is the main objective of each intelligent agent in the reinforcement learning theory. The agent learns this task systematically, by trying various actions in different states and with the reward signal that is assigned within the process. At the same time, the agent changes its knowledge about the environment by modifying current mapping from each state of actions (i.e. the policy).

Reinforcement learning system consists of three elements, see Figure 3. These are:

- Learning element
- Knowledge base
- Performance element

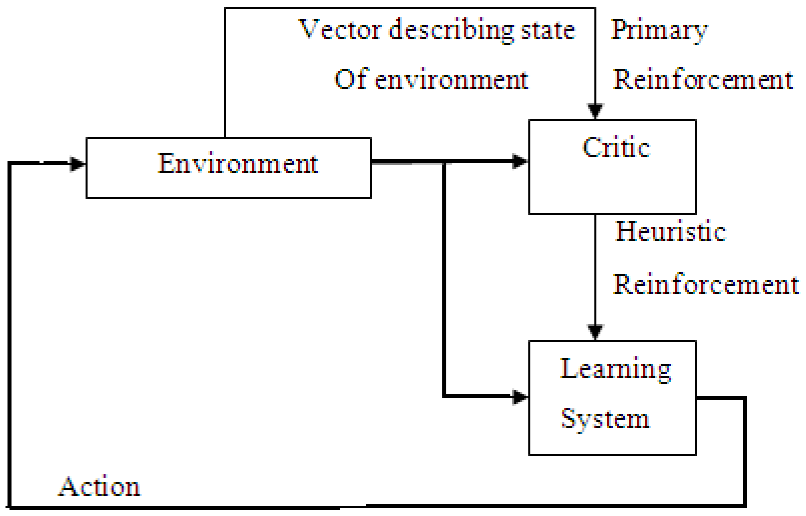


Figure 3: Block diagram of Reinforcement learning [33]

Because no information on way the right output should be provided, the system must employ some random search strategy so that the space of plausible and rational choices is searched until a correct answer is found [34]. Reinforcement learning is usually involved in exploring a new environment when some knowledge (or subjective feeling) about the right response to environmental inputs is available. The system receives an input from the environment and process an output as response. Subsequently, it receives a reward or a penalty from the environment [34].

1.2 Significance of soft computing techniques in domain of robot control and failure prediction

In real world, we have many problems which we have no way to solve logically, or problems which could be solved theoretically but actually impossible due to its requirement of huge resources and huge time required for computation [35]. Soft computing techniques as addressed out by Dr Lotfi Zadeh, have become one of promising tools that can provide practice and reasonable solution using several methodologies, i.e., neural networks [33], fuzzy logic [36], and genetic algorithms (evolutionary programming) [37]. It is also important to stress out that they have some drawbacks in determining the internal parameters of the particular technique, because it requires expert-level knowledge and needs more time and effort depending on the problems and the technique used.

In recent years, the significance of the use of soft computing in various engineering areas is increased in order to identify and resolve some of the problems and improve performance; for example in the industry to come up with an advanced manufacturing in the required quality. Recent advances of soft computing methods and their applications in engineering design and manufacturing can be found in [38]. Likewise, the role of robotic work in the industrial or manufacturing environment has been intensified, so the importance of soft computing in learning is even greater. It is important to develop the robotic control system that can become aware of its present limitations and predict cases of failure and errors in various tasks. Therefore, soft computing techniques contribute to one of the long term goal in robotics, to solve the problems that are unpredictable and imprecise namely in unstructured real-world environments.

In recent years, several adaptive hybrid soft computing frameworks [39] have been developed and provided for model expertise, robotics and complicated automation tasks. It is known that soft computing techniques allow us to develop flexible computing tools to solve complex problems that cannot be solved using traditional algorithms. The main significance of soft computing which related to their application is:

- It can solve nonlinear problems which are not possible using traditional mathematical methods
- It introduced the human knowledge such as prediction, learning and others depends of the scientific field

In general, soft computing methods consist of three essential paradigms: neural networks [33], fuzzy logic [36], and evolutionary programming [37]. Nevertheless, soft computing is an open instead of conservative concept. That is, it is evolving those relevant techniques together with the important advances in other new computing methods such as artificial immune systems [40], memetic computing, evolutionary robotics, etc.

1.2.1 Importance of soft computing in robot control

Nowadays, the development of soft computing methods has attracted considerable research interest over the past decade. They are applied to important fields such as control which need to solve more and more complex problems in industry and many other domains [5, 41, 42]. Soft computing techniques are highly appropriate methods to deal with such complex problems. In many robotic applications, such as mobile robot navigation is shown in Figure 4 [41, 42], It consists of four blocks: perception - the robot must interpret its sensors to extract meaningful data; localization - the robot must determine its position, cognition - the robot must decide how to act to achieve its goals; and motion control - the robot must modulate its motor outputs to achieve the desired trajectory [41, 42].

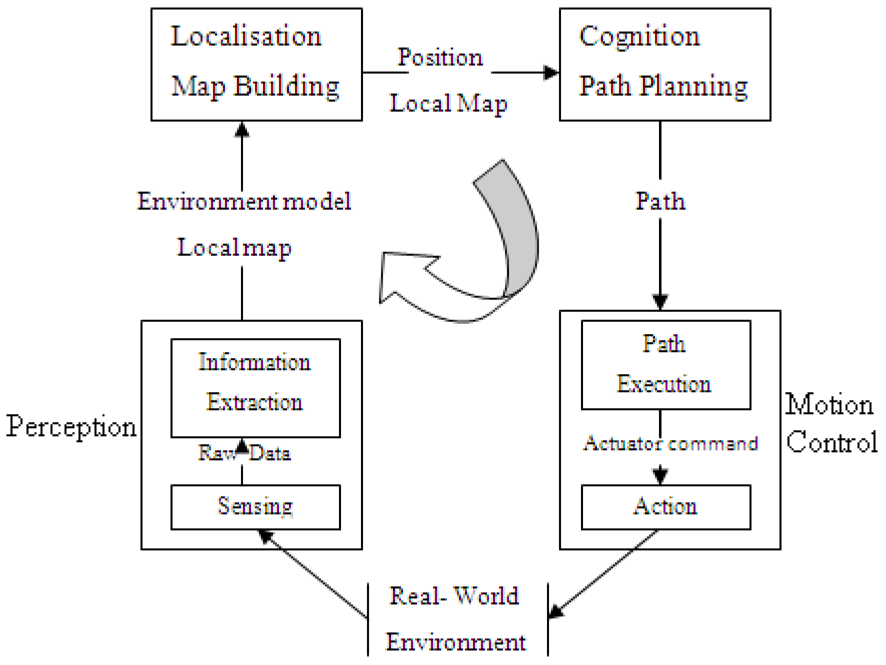


Figure 4: The general control scheme for mobile robot navigation [41]

In situations when precise execution in structured or unstructured environments is of key importance, it is difficult to obtain a precise analytical model of the robot's interaction with its environment. Therefore, the question is: how to make mobile robots move in effective, implement task correctly, safe, and predictable ways? The intelligent robotics systems, whose behaviors change over time, can be effectively used in collaboration with soft computing techniques. These methods allow us to transparently control and simulate several different types of mobile robots. The successful applications of soft computing suggest that the impact of these techniques will be significantly increased in coming years. For example, various methods that use soft computing have been developed to solve mobile robot control problems [40]. Likewise, this artificial intelligence technique is often applied for control of a mobile robot [18, 19], robot manipulator [20, 21], or within the empirical control strategy for industrial robots [41].

Generally speaking, robotic control system must have adaptive capabilities, i.e. the characteristics that enable robot to automatically adapt to environmental changes without a priori knowledge of these changes. In order to do this, robotic system must satisfy following properties:

a) System complexity
b) Nonlinearity
c) Uncertainty

Soft computing today serves as a basic tool for development of many interconnected fundamental problems such as:

- Path Planning for robots. Many methods have been developed for avoiding both static and moving obstacles.
- Localization. The robotic system must use its on-board sensors and wheels to cope with dynamic environmental changes.
- Simultaneous localization and mapping (SLAM) for robotics.

1.2.2 Robust soft computing in the failure domain

It is known that many of systems in nature can have malfunctions and failures due to physical faults in their components. The possibilities of failures increase with the growing complexity of industrial environments. It is therefore essential to pay more attention to the robustness of the industrial robots and systems [5].

The defects in robotic system may occur in sensors, actuators, components of the controlled process, etc. Moreover, faults in their components may develop into failures of the whole system and thus effect the system functioning. To prevent this from happening, the failure of robotic systems has gained more and more attention in the last decade; for example, in fault tolerance [45], failure robot execution [46], failure avoidance [47], layered failure tolerance control structure [48], failure tolerance by trajectory planning [49], and kinematic failure recovery [50].

The failure situations can be classified to many cases and the solution can be achieved by different strategies. The most important and essential requirement for technique of model-based failure analysis is to provide robustness to different kinds of errors [51]. The generation of residuals using parity relations is one example of a method which would be unsuitable for robotic applications [51, 52].

At present time, different efficient robust techniques for aforementioned problems are proposed. Neural networks, fuzzy logic, and evolutionary algorithms are known for addressing and solving these problems to some extent. Neural networks are known for their generalization and can be very useful when analytical models are not available. The NNs are employed in the assembly tasks [8], prediction of failure rates of large number of the centrifugal pumps [9] or in the robust scheme for robot manipulators [10]. These methods are implemented so as to obtain better control and prediction failure of highly non-linear systems behavior. In this thesis, various neural networks and architectures are developed in order to address failure prediction problems in different robotic systems.

1.3 Organization of the dissertation

This dissertation entitled "*Machine learning in intelligent robotic system*" is divided into introduction, and covers the tasks of the research work, theoretical chapters providing background information and hypothesis, chapters with experimental results and conclusions. The following are a general description of the contents of each chapter and the outlines of the structure of the dissertation:

The PhD thesis begins with the background and an introduction with the motivation of the dissertation. In here, first part refers to and overview of machine learning in intelligent robotic systems as well as their disadvantages. Also, the significance and the aim of soft computing techniques in domain of robot contol and failure prediction are given.

Chapter 2: Presents the importance of dissertation objectives and approach. Section 1 gives an overall objective, while Section 2 show main specific objectives and sets the scope of work and overall solutions.

Chapter 3: Express state-of-the-art review including relevant literature and scientific sources. Section 1 describes the main problem, and section 2 presents execution failure prediction related to industrial robotic systems. Section 3 shows advantages and disadvantages of various approaches, while section 4 presents the end of this chapter and includes some of soft computing applications in the domain of prediction analysis.

Chapter 4: Introduces the main methods and approaches given in this work. The problem refers to the robotic failure prediction. The tools include *MATLAB*® and *BPnet* software. Section 1 gives an extensive discussion about the basics and algorithms of neural networks. Section 2 has description of real failure information data, measured immediately after failure detection. Section 3 discusses various prediction algorithms selected and used for the prediction problem, and also gives details about activation functions for the neural networks. Section 4 represents the end of this chapter, and describes entire neural network training procedure in two of software environment- *MATLAB*® and *BPnet* software respectively, with all necessary details.

Chapter 5: Section 1 explains the usage of the intelligent mobile robot in a manufacturing environment. Section 2 describes artificial intelligence techniques implemented for control of the mobile robot. Section 3 explains intelligent robotics localization in a laboratory model of manufacturing environment and in domain of obstacle detection and trajectory tracking. Section 4 presents a lengthy discussion and express two problems in real world domain of obstacle detection and trajectory tracking conducted by real nonholonomic mobile robot.

Chapter 6: Presents experimental study. Section 1 describes experimental setup. Section 2 shows in details the results of all experimental that was created and used in order to develop successful robot prediction system. Moreover, this section show comparison of various tools used in the work.

Chapter 7 is the final chapter. Section 1 presents conclusions that are drawn from this work. Section 2 gives additional recommendations and future research directions.

1.3.1 Main contributions of the dissertation

The thesis contributed to the field of intelligent robotic systems by developing novel machine learning tools for prediction of robot failures. The main experimental results are related to the learning of robotic executions, so that a correct failure prediction can be derived.

The main contributions of the dissertation are:

a) To the author best knowledge, this is the first idea that involves NNs in prediction of robot execution failures using real mobile robot data. Furthermore, the erroneous data is also implemented in the NN training set.

b) Various neural network architectures and learning algorithms are tested in the main experiment. In total, 6 algorithms and 24 neural architectures are tested in the Matlab environment. Additionally, another prediction tool is used in this dissertation - specially designed software titled BPnet [25] which employs most common feedback method for minimizing the error between input and output variables – backpropagation technique [26]. The experimental results confirmed that NN can successfully predict robot execution failures from partially corrupted sensor measurements.

c) This is also the first study that treats prediction of robot failures in the domain of obstacle detection and trajectory tracking using neural networks.

According to the set hypotheses, the main scientific results presented in this dissertation are:

- Methods of predicting undesired behavior of robots based on a system of artificial neural networks.
- The technique for comparison and analysis of different algorithms used for training artificial neural network so as to determine the optimal network architecture.
- Experimental methods for the verification of the developed approach for failure prediction in the domain of object detection problem of unknown dimensions in technological environment.
- Experimental methods for the verification of the developed approach for failure prediction in the domain of the trajectory tracking for mobile robots in indoor environment.

2. Research objectives and approach

2.1 Overall objective

The general scientific objective of the dissertation is the development of an experimental system for prediction of failures in subsystems of industrial robots. This prediction technique is based on obtained sensor information and computational intelligence algorithms such as neural networks. Overall objective is to verify the developed method in the laboratory model of the technological environment using real nonholonomic mobile robot. In order to realize intelligent behavior of the robotic system, the research objective must include the following directions:

- Development of algorithms for the prediction of unwanted behavior of industrial robots in the manufacturing environment based on artificial neural networks and the information obtained from external and/or internal sensors.
- Analysis and comparison of different learning algorithms so as to determine the optimal architecture of the artificial neural network in terms of predicting irregular work in online mode.
- An experimental verification of a new subsystem for failure prediction in intelligent mobile robot, which is used for solving typical problems of obstacle detection and trajectory tracking in manufacturing environments.

At the end of the 20th century and early 21st century various scientific papers, books and PhD dissertation are published, which from different viewpoints treat problems of prediction unwanted behavior for industrial robots. These studies treat robots in different technological environments, as well as their work in domains of detection of obstacle and trajectory tracking. However, none of the research studies has given attention to the use of artificial neural networks in these areas so far; therefore, one can note the significance and importance of this dissertation.

2.2 Specific objective

A robotic systems working in a structured or unstructured environment is exposed to severe conditions such as, increased working hours, changeable working demands, possibility of collision with known or unknown objects, and/or presence of human workers near the robot workspace. Therefore, research presented here must focus on elimination of the aforementioned problems, preferably using the intelligent techniques because of their generalization ability and overall robustness.

In this context, the specific objective of this dissertation has the following:

- To evaluate a possibility use of artificial neural networks for predicting mobile robot failure according to erroneous data from internal sensors.
- To show the application of these techniques on real robotic system working in manufacturing environment.
- To explore using the NNs as a tools to analyze the classification of possible failures.
- To test various prediction algorithms and compare the predictive accuracy of the artificial neural network algorithms in reaching optimal solution.

- To test and confirm that the NNs are able for predict robot execution failure from partially computed sensor measurements.
- To discuss and explain the power of soft computing for predicting the robot failure [53, 46], and also to stress out the advantages of the approaches given in this thesis.

2.2.1 Scientific dissertation methods and approach

After providing the prerequisites for the development of this dissertation, the developed prediction algorithms are based on using following approaches and methodologies:

- Approach of classification of selected real execution task for intelligent industrial robot.
- Approach for failure prediction of intelligent mobile robot based on soft computing techniques; artificial neural networks are the main algorithm for detection and classification of failures.
- Approach for comparing different training algorithms of artificial neural networks.
- Approach for control and programming of mobile robots in the field of localization so as to determine the position and orientation in current pose. Also, this approach treats problems of trajectory tracking and obstacle detection in a laboratory model of the manufacturing environment.
- Test different software implementation of the developed algorithms in order to increase the work efficiency of intelligent industrial robots.

This dissertation uses two software products. In the Matlab environment, various NN training algorithms and architectures are tested by means of mean square error between desired and obtained output values. Additionally, another prediction tool is used in this dissertation - specially designed software titled BPnet [54] which employs most common feedback method for minimizing the error between input and output variables – backpropagation technique [55]. The BPnet software used for the training of backpropagation artificial neural networks, while the Matlab programming environment is used for comparison of different neural network training algorithms. Moreover, real world experiments are conducted on a KheperaII mobile robot in indoor environment for solving obstacle detection and trajectory tracking problems with the aim to additionally verify the method and prove the robustness of the propose prediction algorithms.

3. State-of-the-art review

3.1 Problem description

The problem treated in this dissertation refers to the failure detection in a robot system; more specifically, this thesis treats the robotic failure prediction problem using neural networks and a set of recorded sensor measurements. Consider a robotic system working in manufacturing environment exposed to severe conditions given in previous sections: increased working hours, changeable working demands, possibility of collision with known/unknown objects, and/or presence of human workers near the robot workspace. In these cases it is crucial to ensure maximum safety and smallest deviation from the nominal operating mode by recognizing irregularities in robot behavior. The prediction of industrial robot failures is equally important, since this can provide a continuous and undisturbed work using a backup emergency control commands.

In order to successfully predict execution failures, some sort of safety unit must be employed in the robotic system. In this case, the artificial neural networks are used in the control system as an element for predicting misbehavior based on the corrupted internal and/or external measurements. For example, one can consider obstacle detection problem and an irregular work of several infrared sensors. Given the set of correct sensor values for a particular case (for example, obstacle on the left side of the robot), the robot with the installed NN-based safety element can predict if one or more sensors are malfunctioning. After this, the incorrect sensor measurements can be ignored or replaced with their initial (i.e. nominal) value. In that way using this prediction approach, the system is enabled to work uninherently and to successfully detect different obstacles. Likewise, the trajectory tracking problem can be treated in the same manner; for example, the NN-based unit can be used to predict irregular behavior in wheel control domain. Consider that mobile robot wheels command unit is not working properly all the time, and that in certain control iterations it gives unexplainable large or small commands for tracking the specific trajectory. In this case, NNs can predict these irregularities, with the aim to invoke a nominal control value in the command dataset. In this manner, the bad wheel command is replaced with the desired (calculated) value, and the robot motion is continued without difficulties.

3.2 Prediction of industrial robot execution failures

Nowadays, one of the most desirable features of every robotic system is the ability to adapt to the real world changing conditions [5]. This is especially important for robots working in the hazardous and dangerous surroundings where unwanted events frequently interfere in task accomplishment. Likewise, failure prediction is equally important in these environments in which repairs are often infeasible and failures can have disastrous consequences [56].

In this known that failure prediction and fault tolerance are helpful in reduction of a system down-time. Particularly, with the overcome of failures robot's lifespan is increased, and also the identification of faulty components can significantly speed up the repair process [57]. Also, the reliability of a product manufacturing and increased human safety is ensured by implementing fault tolerance and failure prediction unit in the robotic system.

Failure tolerance has been addressed in various applications for robot manipulators. Usually; redundancy approach in actuation [58], sensors [59] or joints [60] is used. Likewise, different methods are employed for solving the failure detection problem such as second-order sliding-mode algorithm [61], robust nonlinear analytic redundancy technique [62], or partial least squares approach [63].

3.2.1 Fault detection in mobile robotics

The term "fault detection" is commonly referred to as the detection of an abnormal condition that may prevent a functional unit performing required function [64]. Nowadays, the fault detection is solved by implementing a torque filtering technique [65], multiple model adaptive estimation method [66] or using an interacting approach [67].

3.2.3 Failure prediction problem in industrial robotics

Several interesting studies have been reported regarding the failure prediction problem in general. In [68], the method that utilizes the concept of augmented global analytical redundancy relations to handle failures with both parametric and non-parametric nature is presented. Additionally, multiple hybrid particle swarm optimization algorithm is employed in order to realize multiple failures prediction.

Twala addressed the robot execution failure prediction using incomplete data in [53]. Here, this prediction is formulated as a classification problem which is solved by developing a novel probabilistic approach. Likewise, the work given in [46] presents the performance comparison of base-level and meta-level classifiers on the same problem. The results show the superiority of Bagged Naïve Bayes classifier across different settings

However, none of the aforementioned studies incorporate learning techniques in order to improve presented solutions. In this study, neural networks (NNs) are employed for prediction of robot execution failures in order to solve the nonlinear dependencies between input and output variables.

3.2.4 Use of neural networks in prediction of robot failure

Neural Networks are one of the various methods of artificial intelligence that have proved to be useful for many engineering applications. Due to their widely parallel structure, NNs can deal with many multivariables non-linear modeling for which an accurate analytical solution is very difficult to obtain. NNs has already been used for various engineering problems, for example in the areas of image and speech recognition, classification and control of dynamic systems. The ability to learn by example is one of the key aspects of NNs. As a main advantage of this, the system can be considered as a black box where the user does not need to know the details of the internal behavior. These networks may therefore offer an accurate and cost effective approach for modeling problem of failures in mechanical systems. If trained adequately, the NN can simply be used to obtain the prediction of failures in different robots. In this domain, NNs can give accurate prediction if not better than those obtained by conventional methods. However, to develop a reliable prediction model, the appropriate NN architecture, the number of hidden layers and the number of neurons in each hidden layer must be experimentally determined.

This dissertation delivers a novel approach using multilayer feedforward neural networks as a solution for the problem of failure prediction, and also presents performance comparison of different learning algorithms and architectures. In different experiments, NNs are employed for prediction of robot failures in order to solve the nonlinear dependencies between input and output variables. In addition, to check prediction accuracy of different learning systems, other types of NN structures were used - ELMAN neural network is compared to achieve the abovementioned objective. The obtained results indicate that these NNs can also be successfully implemented for failure prediction in robotic applications.

3.3 Advantages and disadvantages of different approaches

Fault and failure detection and their prediction in robotics is critical for the utilization and effectiveness of these systems. There are many quantitative techniques that have been successfully researched and implemented for such kind of failure detection or prediction and have been introduced in various approaches.

Fault tolerance and detection, as well as failure prediction are complex issues for intelligent systems and autonomous robotics. Generally, choosing approaches and techniques are important in order to achieve good failure prediction or detection in many cases; this still remains a challenge to the researchers due to the absence of efficient prediction approaches. Fortunately, NNs is a quantitative approach that is widely employed for pattern recognition, classification, function approximation, and system identification, so it is applicable in failure domain also. The NNs based approach for prediction is able to learn from examples, and is able to catch hidden and strongly non-linear dependencies, even when there is a significant noise in the training set, The ability to learn a mapping between input and output is the main advantage the NNs very attractive to use. Efficient learning algorithms have been developed and proposed to determine the weights of the network, according to the data of the failure task in hand. Considerable research has been carried out to improve accuracy of learning algorithms. Although training algorithms appear in recent neural network literature, in terms of convergence speed and accuracy, it is difficult to know which algorithm works best and is most suitable for the given problem. A number of factors, including the complexity of the problem, the number of datasets used in training, the number of weights and biases in the network, the error goal, and whether the NN is used for classification or regression seem to have influence [69].

3.3.1 Advantages of neural networks

The advantages of NNs are due to their components and abilities, such as the learning mechanisms, their structure, and activation functions. They are able to classify both linearly and nonlinearly separable problems due to the nonlinear transformation they perform on the learned data. This allows them to fit linearly separable problems as well as more complex nonlinearly separable problems [70]. Many learning algorithms and neural structures have emerged, giving neural networks a wide selection of methods to improve performance. Neural networks are also error tolerant. This is largely due to the relatively large number of neurons they contain. Errors in the form of missing data, noise or glitches get averaged out over the entire network [71]. Neural networks are also very robust in that for given a dataset, neural networks can adjust themselves to fit the given data automatically via chosen learning algorithm [33]. The true power of neural networks is demonstrated when they are applied to complex multivariate nonlinear problems [71]. Neural networks require no prior assumptions or knowledge regarding the underlying relationships between variables of a given problem, since they learn directly from the data in a robust manner [71].

Neural networks can successfully represent many statistical techniques, i.e., regression models from simple linear regression to projection pursuit regression, nonparametric regression, generalized additive models, logistic regression, Fisher's linear discriminated function, classification trees, etc. [71, 72]. The prediction problem in this thesis is transformed into a classification problem, similarly to research work in [53]. Although neural networks are effective, there are still many ways to improve their classification accuracy. Many techniques, such as the input preprocessing, modular approach [73] and the ensemble technique [74, 75, 76, 77] can be used for this purpose.

The advantages of using neural nets in prediction can be express as following:

- They can be used in various applications, ranging from classification, to control and optimization. Different to conventional algorithms, NNs are incremental learning algorithms because at any stage during the training process training can be stopped, NN would still serve as a model of function being learned, even though it may not be quite accurate.
- They can be used in developing the empirical models based on experimental and observational knowledge.
- They are best suited for fast computations on parallel architectures.
- They have good generalization capabilities.
- They can learn from experience and give accurate results from incomplete and noisy data.
- They do not require any a priori knowledge of mathematical function that map the input to the output. They need only input-output examples to train the network (in supervised learning).

3.3.2 Limitations & Disadvantages of neural networks

A major disadvantage of NNs is in the difficulty to interpret the meaning of its structure. That is, given a trained network, it is not easy to derive meanings from the weights of the network to understand the underlying relationships between the inputs and the outputs. Although the network is excellent at detecting significant features and relationships, it is difficult to understand them [77]. Neural networks require a large number of training instances to be able to generalize well on a given problem. Moreover, they require knowing, prior to training the network, what features of the data are more indicative to the class since neural networks do not learn such information [71]. Attribute selection and preprocessing, such as normalization, discretization, and others are often required [77], as to be discussed shortly in Chapter 4. Moreover, it is difficult to determine the best neural network structure and learning time for a given problem. Although many techniques are presented to deal with this problem, no state-of-the-art algorithm is able to determine the best neural structure [71].

The disadvantages of using feed forward neural nets as predicting tool for robot failures are [78]:

- The largest drawback with feedforward back-propagation algorithm appears to be its convergence time. Training sessions can require hundreds or thousands of iterations. Realistic applications may have thousands of examples in a training set, and it may take days of computing time or more for complete training. Usually, this lengthy training needs to be done only during the development of the network, because most applications require a trained network and do not need on line re-training of the net.
- Lack of proper guidelines for networks architecture (number of hidden layers and number of nodes in each layer) hinders the use of these networks fully. However, the flexibility of the network's paradigm is enhanced by the large number of design choices available: choices for the number of layers, learning constant, and data representations.

It is important to note that there are some limitations to neural computing. The key limitation is the neural network's inability to explain the model it has built in a useful way. Analysts often want to know why the model is behaving as it is. Neural networks get better answers but they have a hard time explaining how they got there [79]. There are a few other limitations that should be understood. First, it is difficult to extract rules from neural networks. This is sometimes important to people who have to explain their answer to others and to people who have been involved with artificial intelligence, particularly expert systems which are rule-based.

As with most analytical methods, you cannot just throw data at a neural net and get a good answer. You have to spend time understanding the problem or the outcome you are trying to predict [79]. And, you must be sure that the data used to train the system are appropriate and are measured in a way that reflects the behavior of the factors. If the data are not representative for the problem, neural computing will not product good results [79]. Finally, it can take time to train a model from a very complex data set. Neural techniques are computer intensive and will be slow on low end PCs or machines without math coprocessors. It is important to remember though that the overall time to results can still be faster than other data analysis approaches, even when the system takes longer to train [79].

3.4 Applications of soft computing techniques in the prediction domain

Nowadays many research studies have been using soft computing in various fields [80]. They included the application of neural net works, fuzzy logic, genetic algorithms, etc. The popular soft computing technique is NN which is considered as a main computational tool in this dissertation and is used for performing the nonlinear mapping between inputs and outputs. For example, NNs can be used for prediction of vehicle reliability performance [2] or in education to predict professional movements of graduates [3]. In this subsection, some specific applications of NNs in prediction analysis are mentioned.

3.4.1 General NN application

As stated before, NN are mostly employed for solving many types of non-linear problems that are difficult to solve by traditional techniques. The NNs have been found to be both reliable and effective when applied to applications involving prediction, classification, and clustering [81]. The most frequent areas of NNs applications are production/operations (53.5%) and finance (25.4%) [82].

3.4.2 NN application with noise data

NNs often find usage in cases when dealing with noise in data, in the situations when data contains complex relationships between many factors, or when other mathematical techniques or methods are not adequate [83]. By adjusting weights iteratively between the neurons in different layers, the network is able to find hidden rules between the data [1]. The main advantages of NNs are their information processing abilities such as nonlinearity, high parallelism, robustness, fault and failure tolerance, learning, ability to handle imprecise information, and their capability to generalize [84].

3.4.3 Neural computing and output calculation

The robot failure prediction based on the soft computing methods has not been reported in the literature so far, and with the stated advantages the prediction of robot execution failures appears to be an appropriate assignment for NNs. Neural network are inspired by biological neurological system, and are composed of simple processing elements called artificial neurons or nodes capable of performing massive parallel computations for data processing and knowledge representation [1, 84]. The neurons are able to communicate between themselves and to exchange information through the biased or weighted connections.

Each neuron in NN is active or non-active based on the adding value and activation function value. Adding value is determined by summarizing all inputs to the particular cell modified by their weighting coefficients, while activation function affects amplitude of the neuron output. After defining these neuron components, the training process in a supervised manner is set to start. Firstly, an input to the each neuron in the first (i.e. input) layer must be defined. The weights between an input neuron and the neurons in hidden layer indicate the degree of importance between these units. Thus, the strength of connections between neurons is given by the numerical value between -1 and 1 which represents aforementioned weight number. Secondly, the output value for each neuron is calculated by using weighted input through the activation function. If that value is larger than the neuron internal threshold, the processing unit is activated; otherwise, there is no output from that particular neuron. After the calculation of outputs from every neuron in the network, the error between the output values in the last (i.e. output) layer and the pre-defined desired output is calculated. Then, that error is propagated backwards from the output to the input layer, in order to determine new network weights that will decrease the difference between the desired and actual output. This iterative procedure is finished when these values are close enough, i.e. when they are bellow the pre-defined learning threshold. After the training step is over, a validation and testing are active next. In the validation phase, the length of NN training, learning parameters and number of units in hidden layers are optimized. The testing phase represents network performance evaluation on a new sample, and the result is taken as the assessment of the NN. Finally, the network with the optimal performance is used as a solution for the problem in hand.

4. Robotic failure prediction in *MATLAB®* and *BPnet* software

4.1 Introduction to neural networks

Essentially there are two types of neural networks: biological neural networks and artificial neural networks. The human brain is an example of a biological neural network, composed of billions of neurons organized in a fashion so that it can perform complex tasks such as vision and speech recognition [86, 87]. Artificial neural networks are a product of attempts to enable computers to do the types of things that the human brain does well. Computers are high speed, serial machines designed to carry out a set of instructions, one after another, extremely rapidly [86]. They can typically carry out millions of operations per second, which enables them to be very good at tasks such as adding long lists of large numbers. However, unlike the human brain, computers are not good at complex tasks such as pattern recognition. This is because the problem of pattern recognition is a parallel one, requiring the processing of many different items of information which all interact to form a solution [86, 88].

The early goal of neural computing was to model the human brain and to capture the underlying principles that allow it to solve complex problems [86]. Early artificial neural networks consisted of individual electronic devices; the neurons were actual hardware in the computer. The first "neural network" was built in 1951 by Martin Minsky and Dean Edmonds. It was a large scale device that consisted of 300 tubes, motors, clutches and a gyro from a World War II bomber, all used to move 40 control knobs [86]. The position of these knobs represented the memory of the machine [88].

Nowadays, artificial neural networks are composed of a set of computer instructions which simulates the neurons and the connections between the neurons [86]. Information is stored as patterns, not a series of information bits as in normal computer programs. An artificial neural network does not work using a series of instructions; instead the network architecture and training method determine how the system will work [86]. Artificial neural networks do not have separate memory for storing data; data is stored throughout the system in patterns.

4.1.1 Biological Neurons

The human brain contains approximately 10 billion (1010) basic units called neurons. Each of these neurons is connected on average to about 10,000 (104) other neurons [86]. Biological neurons are complicated devices that have a number of parts, sub-systems and control mechanisms. The operation of the biological neuron is a complicated and not fully understood process, but the basic details are simple. The neuron accepts inputs and adds them up in some fashion. If the neuron receives enough active inputs at once, the neuron will be stimulated and "fire;" if not the neuron will remain in an inactive state [86, 88].

A representation of the basic components of a biological neuron, the soma, the axon, synapses, and dendrites, is shown in Figure 5**.**

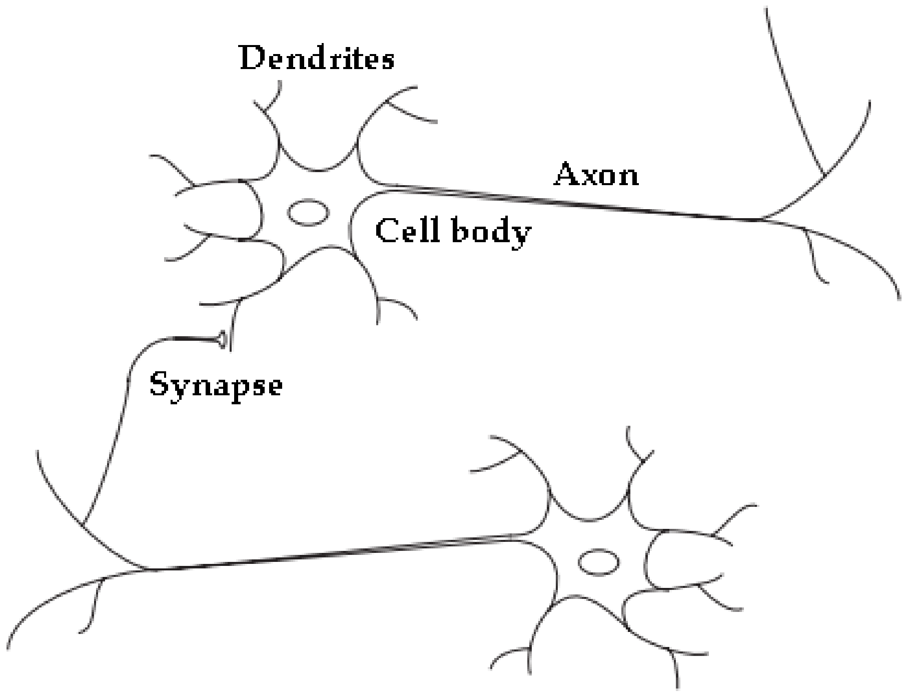


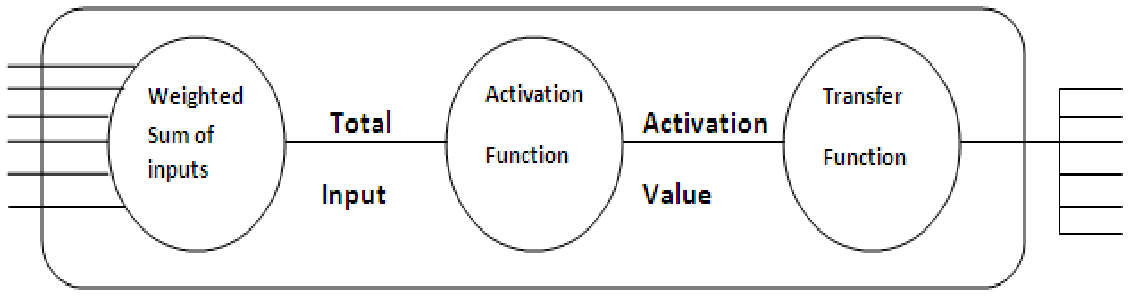
Figure 5 : Schematic drawing of biological neurons

A brain neuron receives signals from many other neurons through synapses, which regulate how much of each incoming signal passes to the dendrites, which are the input channels to the soma [86]. The soma is the body of the neuron. In the soma, incoming signals are added up and a determination made of when and how to respond to the inputs when the neuron "fires," a pulse is sent down the axon, an extension of the nerve cell body. The axon is the output channel of the neuron, carrying impulses to other neurons in the brain [86].

4.1.2 Artificial Neurons

Artificial network neurons work in much the same way as biological neurons. A typical neuron used in artificial neural networks is shown in Figure

6. The neuron is receiving six distinct inputs from other neurons. This neuron is shown sending an output to six other neurons in the system.



Inputs **Output To other Neurons**

Figure 6 : Artificial Neuron Internal Representation

The inputs may be excitatory, tending to increase the activity of the neuron, or inhibitory, tending to decrease the neuron's activity. Once in the neuron, the inputs are weighted and combined into a single value in the box labeled weighted sum of inputs [86]. Usually the inputs are simply multiplied by some weight and added together, but in some artificial neurons the calculation is more complex. Inhibitory signals can have a negative value, and thus can be added to excitatory signals but reduce the activation value. The result is the total input, which is transformed by another function know as the activation function [86].

Artificial neurons are sometimes compared to latches [86]. A latch is a digital circuit with a feedback loop which causes it to retain or store its state. A latch can hold that piece of data indefinitely. Neurons do not hold specific on/off information, instead they keep track of how they respond to the neurons connected to them and fire based upon their input. When a neuron fires it sends out a signal. The length of time spent firing a signal is constant but the overall firing frequency is variable. Higher firing frequencies signal that the neuron is more excited [86, 87].

4.1.3 Characteristics of NNs

Many types of artificial neural networks exist today [86]. It is beneficial to understand some of the terms that define and describe different types of neural networks before discussing them in detail [86]. Various terms and simple definitions that describe behavior and abilities are presented in the remainder of this section [86].

Adaptability is the ability to modify a response to changing conditions in the network. Four separate processes produce this ability: Learning, training, self-organization, and generalization [86]. Learning is the process by which a network modifies its connection weights in the activation function of the neuron. There are two types of learning: supervised and unsupervised.

Supervised learning is characterized by an outside influence (either a set of training facts or an observer) telling the network whether or not its output is corrects [86]. The network's output is compared to the correct output, and the synaptic weights in the individual neurons are adjusted to make the next output closer to the desired output.

In *unsupervised learning* the network does not use a set of training facts nor is it coached by an outside observer [86]. Rather, it classifies inputs as patterns that share common features with other input patterns, with no regard to actual output [86, 87].

Training is the process in which the connection weights are modified in some fashion, using the learning method. Self-organization is how artificial neural networks train themselves according to the learning rule. Typically all of the network's neuron weights are modified at the same time.

Generalization is the network's ability to classify patterns that have not been previously presented to the network [86]. Networks generalize by comparing input patterns to the patterns held In the synaptic weights of the individual neurons. A pattern that the network previously has not seen is classified with other patterns that share the same distinguishing features as those on which the network has been trained [86].

In typical computers, if a sector of memory is lost, the program will fail. However, an artificial neural network will continue to function, but at a reduced speed and capacity. Plasticity is the ability of a group of neurons to adapt to different functions over time. When a portion of the network is damaged, other neurons adapt to take over functions that the damaged portions performed. Fault tolerance is the ability to keep processing, at a reduce speed and capacity, when a portion of the network is damaged [86, 87].

Most training data sets will typically have outliers in the data, that is, observations that are outside the "normal" range for the set of observations. Dynamic stability is the ability of the network to be given an extreme observation and yet remain within its functional boundaries and reach a stable state. Convergence is the changing state of the network as it moves towards that steady state [86].

4.1.3.1 Layers

A neural network consists of groups of neurons arranged in structural units known as layers [86]. **A** layer of neurons is a group of neurons that share a functional feature. There are three possible types of neurons in a neural network, each type relating to the layer in which it lies in the network [86].

The input layer neurons receive data from the outside world, from data files, keyboards or other transmitting devices. *The output layer neurons* send information back to the user in a form defined **by** the setup of the network. *The hidden layer neurons* are all of the neurons lying in the layers between the input and output layers. Neural networks may have only one hidden layer, no hidden layers, or many hidden layers, u-pending on the architecture and complexity of the network and the computing capacity of the user computer. The user will not see the inputs and outputs of the hidden neurons because they connect only to other neurons [86, 87].

4.1.3.2 Network classification and description

This section explains the various classifications of artificial neural networks shown in Figure 7, and briefly explains the theories behind the networks [86]. Because this dissertation uses the backpropagation learning algorithm as its basic artificial neural network, much of the remainder of this section is devoted to backpropagation and its predecessor, the perceptron. A basic mathematical foundation for these types of artificial neural networks is provided [86]. The remainder of this section provides a short description of other artificial neural networks not used in this thesis, but used in other areas nowadays [86].

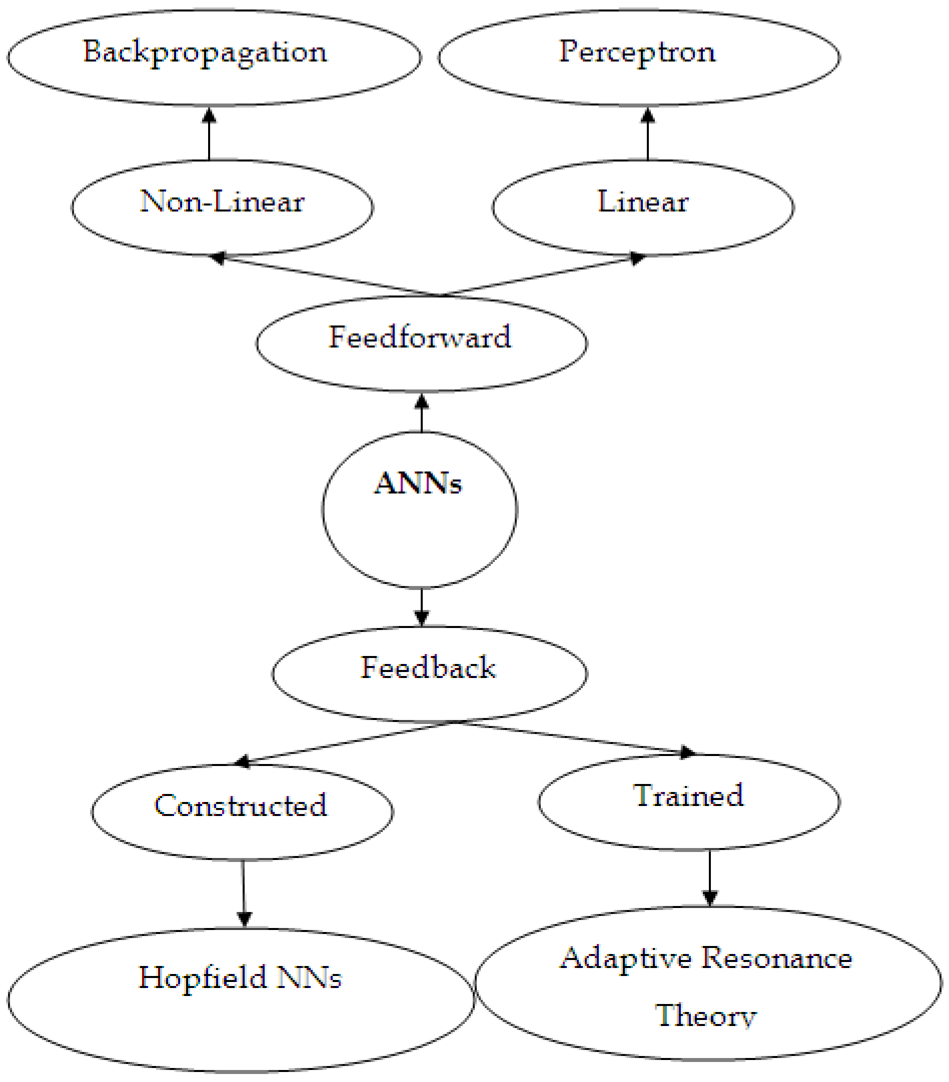


Figure 7 : Various classifications of artificial neural networks [86]

4.1.3.4.1 Perceptrons

The perceptron, developed in 1957 by Frank Rosenblatt of Cornell University, was the result of one of the first major research projects in the field of artificial neural networks [86]. A simple perception neuron with two inputs and one output is shown in **Figure 8**. The term X_0 is always positive one, and the weight W_0 is referred to as the bias, and operates like the constant in a regression equation [86].

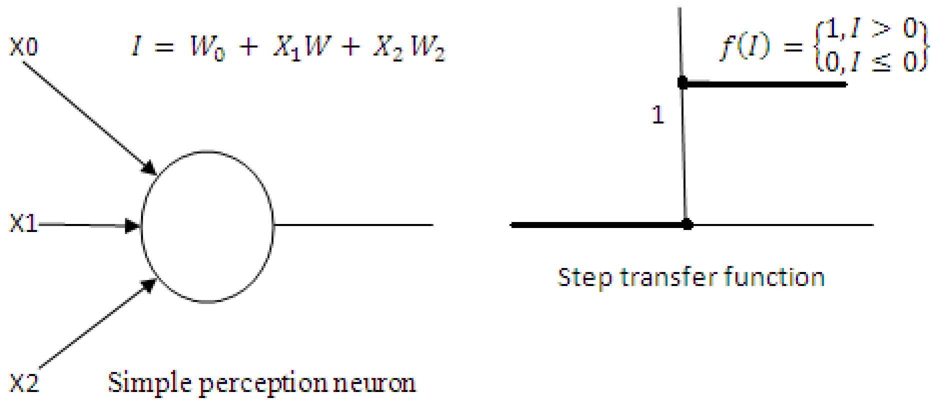

X0
$I = W_0 + X_1W + X_2W_2$
X1
X2
Simple perception neuron
$f(I) = \begin{cases} 1, I > 0 \\ 0, I \leq 0 \end{cases}$
1
Step transfer function

Figure 8 : Simple perception neuron and step transfer function [86]

The perception network is essentially a linear separator. If we assume a simple network with two neurons in the input layer and one neuron in the output layer, the network can be used to separate the two classes of output shown in Figure 9 [86]**.** When the network begins with random weights, occasionally the inputs to the network will result in a correct output [86]. However, some of the input combinations will result in incorrect outputs. In these cases the weights need to be adjusted so that future sets of inputs will yield correct outputs. This adjustment of weights is referred to as learning. The learning algorithm for the perceptron network, as modified by Windrow and Hoff in 1960 follows [86]:

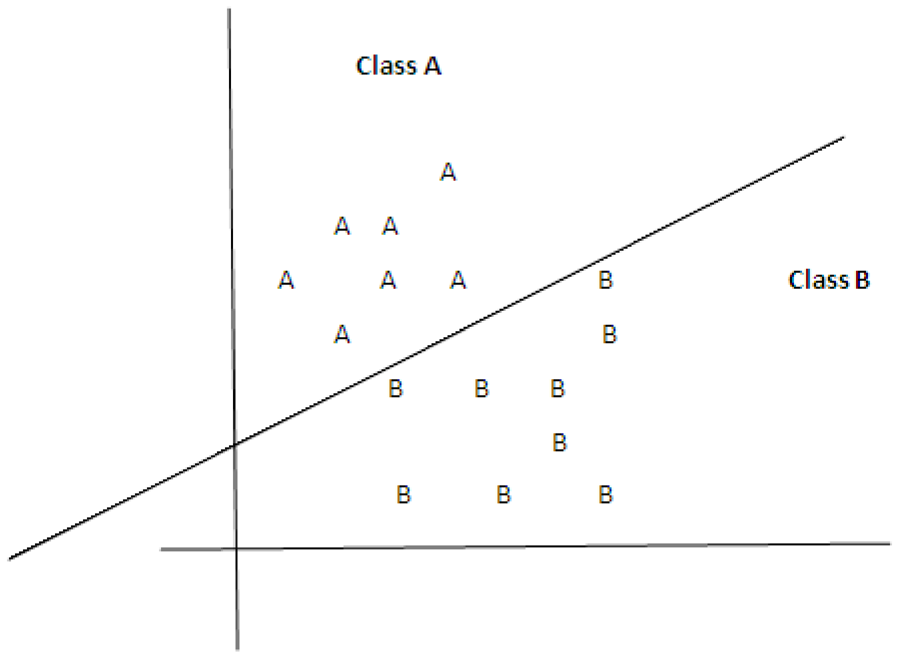


Figure 9 : Two lineally separable classes [86]

1. Randomly initialize the weights and the bias
2. Present an input pattern $(X_{1t}, X_{2t}, \ldots \ldots, X_{nt})$and a desired output d_t to the network
3. Calculate the actual output of input $t, y_t,$

   from the network: $y_t = f[\sum X_{it} \; yW_{it}$
4. Compute the error of output t, e_t: $e_t = d_t - y_t,$
5. Compute the new weights for input $t + 1$: $W_{it+1} = W_{it} + \alpha \; e_t x_{it}$

   Where α is the learning rate, $0 < \alpha < 1$
6. Repeat steps one through four for each new input pattern $(X_1, X_2, \ldots \ldots, X_n)$. Repeat steps one through five until error is less than some preset tolerance

For the above example $d_t = 1$ if the desired output is from class$A$, and $d_t = 0$ if the desired output is from class$B$. If W, and W_2 initially are randomly set to one and the bias is set to zero, the initial line will have a slope of negative one and an intercept of zero [86]. As the perceptron is fed input patterns and learning is accomplished through the Windrow Hoff delta rule, the line separating the two categories will gradually shift until the slope is equal to$-X_2 / X_1$, and the intercept is equal to -W_0 [86]. This gradual shifting of the linear separator is shown in Figure 10. Line one (L_1) is the beginning line, with initial weights of positive one, and line five (L_5) is the hypothetical ending line that the network produces that separates class A from class B [86].

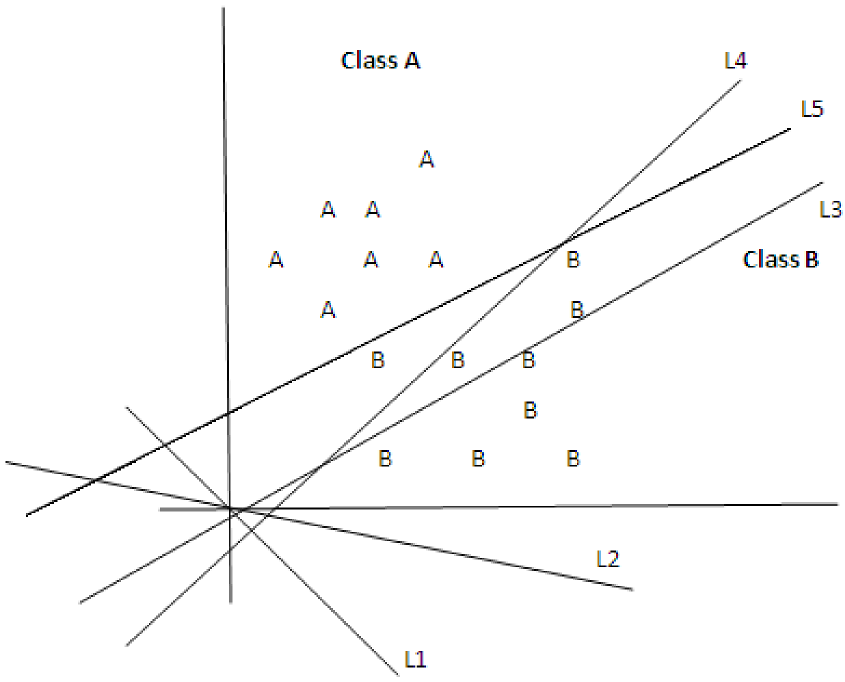


Figure 10 : Two linearly separable classes [86]

As previously stated, the perception was the result of early work in the field of artificial neural networks. As with any model, the perception has limitations to its capabilities [86]. It will learn a solution if the problem is linearly separable. In many cases however, the separation between classes is much more complex. The classic simple problem that the perception is unable to solve is the case of the exclusive-or (XOR) problem [86]. The XOR logic function has two inputs and one output. It produces an output only if either one or the other of the inputs is on, but does not produce an output if both inputs are off or both inputs are on [86]. The exclusive-or (XOR) problem is shown in both tabular and graphic form in Figure 11 [86].

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

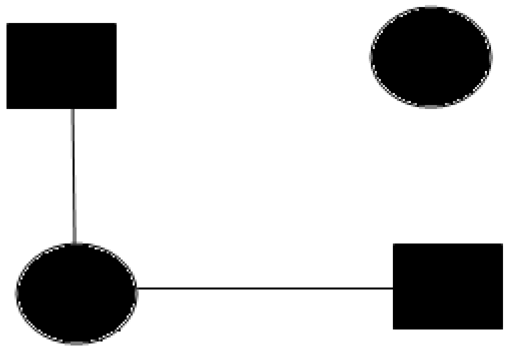


Figure 11 : Exclusive-or ($\boldsymbol{XOR}$) problem [86]

4.1.3.4.2 Backpropagation

In 1986 a breakthrough in the study of artificial neural networks was put forth by Rumelhart, McClelland, and Williams in their book Parallel Distributed Processing [86, 89]. Their breakthrough was a way to use a smooth transfer function in a multi-layer perceptron network, combined with a learning rule which "backpropagated" the error from the output layer to the input layer, thus solving the credit-assignment problem [86]. The term "backpropagation" refers to a type of learning algorithm for adjusting the weights in a multiple layer feed-forward network. However, the term has become synonymous with the type of network itself. In backpropagation, the responsibility for output error is assumed to be the problem of all the connection weights in the network. Errors are calculated at the output layer, then using a sum of products to the previous layer, the previous artificial neurons are assigned error [86]. The errors are then used in adjusting the incoming weights so as to produce an output closer to the correct output for the next set of learning inputs [86, 90].

4.1.4 Operation of NNs

The normal operation of a neural network is a selective response to a signal pattern [86]. How each specific network learns is determined by type of connections between the neuron, the weight assigned to a signal, and the rules which change the input function. An example which helps to explain the operation of a neural network is that of a network trained to predict dependent numerical outputs from a set of inputs, or explanatory variables. A feed-forward, backpropagating network is used in this case. Each of the explanatory variables is assigned to an input neuron, which in turn sends signals to the next layer of neurons, the hidden layer [86]. Each hidden neuron receives signals from all the neurons in the preceding layer. The signals are assigned connection weights and summed in the activation function of the neuron. If the activation value is greater than the threshold value, the neuron "fires" and sends a signal to the next layer. If less than the threshold value, the neuron remains in an inactive state [86]. Once all of the inputs have been passed through the hidden layer the outputs are sent to the output layer of neurons. The output layer of neurons, in this case only the one neuron associated with the dependent variable that is being predicted, is compared to a value known as the training value. The training value is the actual value of the dependent variable for the explanatory variables in the observation [86]. In the back propagation learning method the predicted value is compared with the actual value of the dependent variable, and if there is a difference, an error signal is fed back throughout the network, altering the connection weights in each of the neuron's activation functions. The network iteratively moves to the next observation in the data set, until a pattern is formed and the network can successfully predict and match all of the output values to their actual values At this point the network is considered trained and ready for testing by the user. Testing is accomplished in much the same manner as training [86]. A separate testing data set with new explanatory and dependent observations is input into the network. The predicted outputs are compared with the actual dependent values to determine how well the network is performing on data separate from the training data set [86].

4.2 Failure data description

As mentioned before, this work considers the NN prediction ability concerning robot failures so as to successfully detect and classify failures and to dependently track and monitor the action execution.

4.2.1 Robot execution data

The data used in this dissertation is obtained from a real system, and refers to the evolution of forces and torques during execution of a specific task. In order to correctly evaluate and compare various NN algorithms and architectures, the failures in approach to grasp position are considered. Each feature in the dataset represents a force or torque value measured immediately after failure detection. Total number of instances is 88, and each instance consists of sensor measurements (i.e. samples) collected at regular time intervals. Three values of forces and torques are founded in each sample.

$$\begin{array}{cccccc} F_{x1} & F_{y1} & F_{z1} & T_{x1} & T_{y1} & T_{z1} \\ F_{x2} & F_{y2} & F_{z2} & T_{x2} & T_{y2} & T_{z2} \\ \ldots\ldots & & & & & \\ \ldots\ldots & & & & & \\ F_{x15} & F_{y15} & F_{z15} & T_{x15} & T_{y15} & T_{z15} \end{array}$$

Figure 12 : 90 different features of one instance (i.e. F&T values)

Therefore, Figure 12 shows one instance has 90 different features (i.e. the values of F and T). This data is publicly available via well-known machine learning repository [91].

4.2.2 Failure Dataset classes

In the failure dataset, 4 different robot situations (i.e. data classes) can be identified: normal, collision, obstruction and front collision with the distribution of 24%19%18% 39%, respectively. The identification of particular class is based on the values and/or relationships between measured forces and torques. As an example, in Figure 13 the Fx and Tx in one instance for each robot situation are presented. It is obvious that the values are very different, which is especially suitable for NN prediction purposes [1, 5].

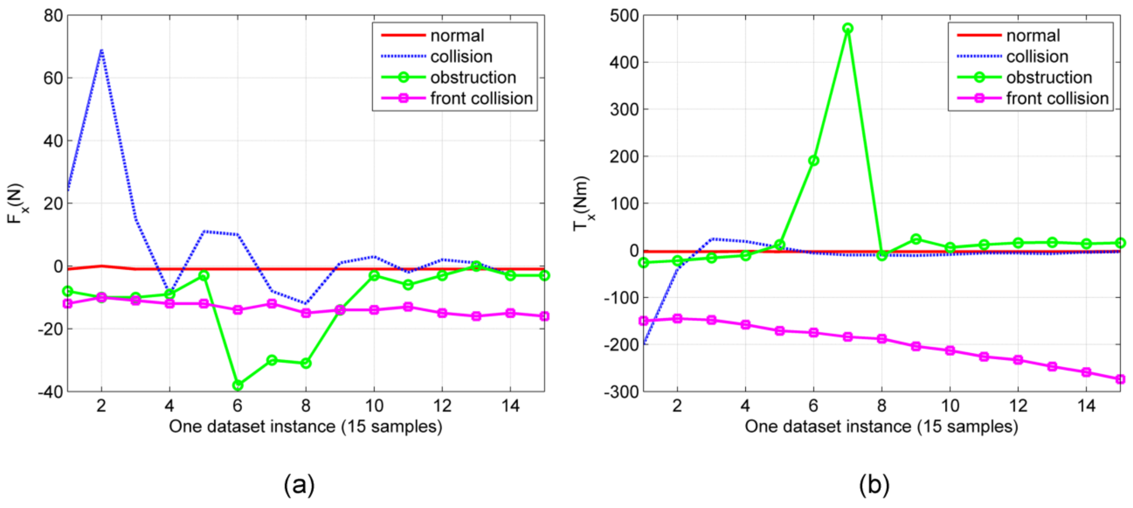


Figure 13 : An example of force and torque value in one dataset instance:

(a) F_x, (b) T_x

4.3 Prediction algorithms, activation functions and neural network architectures

The attractiveness of using NNs lie in the ability to 'learn' between inputs and outputs by using various learning algorithms [5]. These methods have been developed and proposed to determine the weights of the network, according to the data of the computational task to be performed. The learning ability of the NNs makes them useful to solve non-linear problem structures such prediction, and others. Considerable research has been carried out to accelerate the convergence of learning algorithms which can be broadly classified into two categories [92]:

(1) Development of ad-hoc heuristic techniques which include such ideas as varying the learning rate, using momentum and rescaling variables;

(2) Development of standard numerical optimization techniques. The three types of numerical optimization techniques commonly used for NN training include the conjugate gradient algorithms, quasi-Newton algorithms, and the Levenberg-Marquardt algorithm [92, 93].

4.3.1 Neural networks training algorithms

There are number of batch training algorithms which can be used to train a network. Here, several types of training algorithms have been evaluated for classification purposes. The following sub-sections briefly describe the various NN training algorithms considered in this dissertation:

1. Levenberg–Marquardt (LM) backpropagation algorithm - *trainlm* [92]:
The LM second-order numerical technique combines the advantages of Gauss–Newton and steepest descent algorithms. It locates the minimum of a

multivariate function that can be expressed as the sum of squares of non-linear real-valued functions [92]. It is an iterative technique that works in such a way that performance function will always be reduced each iteration of the algorithm. The LM training algorithm is considered to be very efficient when training networks which have up to a few hundred weights. Although the computational requirements are much higher for each iteration of the LM training algorithm, this feature makes *trainlm* the fastest training algorithm for networks of moderate size. Similar to BFGS quasi–Newton Backpropagation, *trainlm* algorithm has drawback of memory and computation overhead caused due to the calculation of the gradient and approximated Hessian matrix [92, 94].

2. Bayesian Regularization (BR) Backpropagation - *trainbr* [92]:

The BR training algorithm is considered as one of the best approaches to overcome the over-fitting tendencies of NNs so that their prediction accuracies for unseen data can be further enhanced [92]. This approach minimizes the over-fitting problem by taking into account the goodness-of-fit as well as the network architecture. The BR network training function updates the weight and bias values according to Levenberg-Marquardt optimization [92]. It minimizes a combination of squared errors and weights, and then determines the correct combination so as to produce a network that generalizes well [92]. This process is called Bayesian regularization

3. Resilence Backpropagation (RP algorithm) - *trainrp* [92]:

Is the one of the most popular training algorithms that implements basic gradient descent algorithm and updates weights and biases in the direction of the negative gradient of the performance function and it is training algorithm eliminates the effects of the magnitudes of the partial derivatives [92, 95]. In this sign of the derivative is used to determine the direction of the weight update and the magnitude of the derivative have no effect on the weight update. The size of the weight change is determined by a separate update value. The update value for each weight and bias is increased by a factor whenever the derivative of the performance function with respect to that weight has the same sign for two successive iterations [92, 96]. The update value is decreased by a factor whenever the derivative with respect that weight changes sign from the previous iteration. If the derivative is zero, then the update value remains the same [92]. Whenever the weights are oscillating weight change will be reduced. Resilient Backpropagation is generally much faster than the standard steepest descent algorithm although it requires only a modest increase in memory requirement [92].

4. Scaled Conjugate Gradient (SCG) - *trainscg* [92]:

The basic gradient descent algorithm adjusts the weights in the negative of the gradient, the direction in which the performance function is decreasing most rapidly [92]. This does not necessarily produce the fastest convergence. In the conjugate gradient algorithms a search is performed along conjugate directions, which produces generally faster convergence than steepest descent directions. The conjugate gradient algorithms require only a little more storage than the other algorithms [92]. Therefore, these algorithms are good for networks with a large number of weights [92, 97]. Algorithm *trainscg* is helping to minimize goal functions of several variables and does not require line search at each iteration step like other conjugate training functions. Step size scaling mechanism is used which avoids a time consuming line search per learning iteration. The SCG training algorithm was developed to avoid this time-consuming line search. The (*trainscg*) function requires more iteration to converge than the other conjugate gradient algorithms, but the number of computations in each iteration is significantly reduced because no line search is performed [92, 98]

5. Broyden–Fletcher–Goldfarb–Shanno (BFGS) quasi–Newton Backpropagation - *trainbfg* [92]:

BFGS (trainbfg) algorithm approximates Newton's method is an alternative to the conjugate gradient methods for fast optimization. a class of hill-climbing optimization techniques that seeks a stationary point of a function. . For such problems, a necessary condition for optimality is that the gradient be zero. The Broyden–Fletcher–Golfarb–Shanno (BFGS) algorithm is one of the most popular of the quasi-Newton algorithms [33, 92, 99]. The basic step of Newton's method is to form the Hessian Matrix (second derivatives). This method often converges faster than conjugate gradient methods but it is complex and expensive to compute the Hessian Matrix for feedforward neural networks [92]. For smaller networks, however, BFGS can be an efficient training function. BFGS have good performance even for non smooth optimizations and an efficient training function for smaller networks [92].

6. Variable Learning Rate Backpropagation (GDX) - *traingdx* [92]:

The GDX training algorithm combines adaptive learning rate with momentum training [92]. It is similar to Gradient Descent with Adaptive Learning Rate Backpropagation (GDA) algorithm except that it has a momentum coefficient as an additional training parameter. Thus, the weight vector update is carried out the same way as in Gradient Descent with Momentum Backpropagation (GDM) except that a varying learning rate is used as in GDA [92].

7. Gradient descent backpropagation algorithm [92]:

The gradient descent backpropagation training algorithm is based on minimizing the mean square error between the network's output and the desired output [92]. Once the network's error has decreased to the specified threshold level, the network is said to have converged and is considered to be trained. The backpropagation algorithm updates synaptic weights and biases along the negative gradient of the error function [92].

8. Elman NN [100]:
The Elman neural network is a simple recurrent neural network (SRN) developed by Jeffrey L. Elman in 1990. This network type consists of an input layer, a hidden layer, and an output layer. In this way it resembles a three layer feedforward neural network. However, it also has a context layer. This context layer is fed, without weighting, the output from the hidden layer. The Elman network then remembers these values and outputs them on the next run of the neural network. These values are then sent, using a trainable weighted connection, back into the hidden layer. Elman neural networks are very useful for predicting sequences, since they have a limited short-term memory [100].

4.3.1.1 NN training

The training process of the feed-forward NN proceeds in a supervised manner [101]. During the supervised learning, the desired response is provided for each input instance. The set of N available input patterns can be expressed as [101]:

$$T = \{(x(t)\ ,\ d(t)\ ,\ t = 1......n)\} \tag{4.1}$$

Here $x\ (t) = [x_1(t)\ ......,\ x_n(t)]$ denotes the input n-dimensional vector and $d(t)$ is the desired output. The task of the training process is to minimize the error $e(t)$ with respect to the desired output for each input pattern [101]. The supervised training process of the NN is schematically depicted in Figure 14.

The performance of the trained NN is tested on input patterns [101]. Hence, set *T* can be partitioned into a training set used during the training phase and a testing set used for performance evaluation [101]. Furthermore, a validation set can be created in order to validate the generalization performance during the training process of the training data [102, 103].

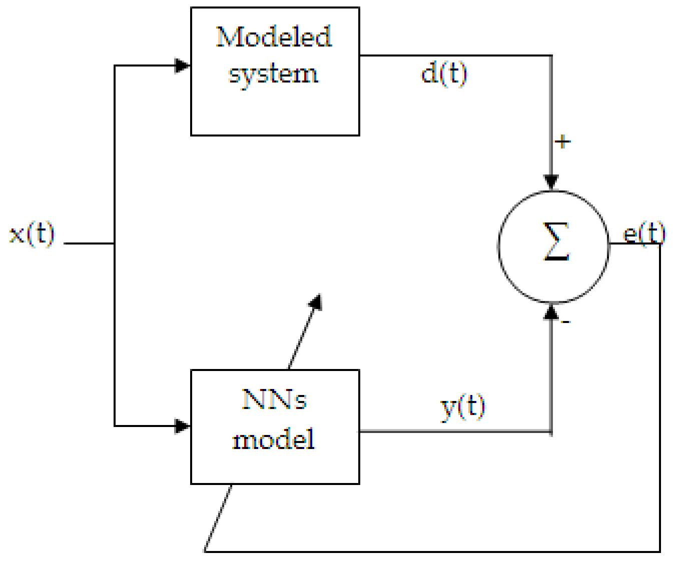


Figure 14 : Supervised training of NNs [101]

4.3.2 Activation function

The activation function specifies what the neuron is to do with the signals after the weights have had their effect [92]. In the simplest models the activation function is the weighted sum of the neuron's inputs; the previous state is not taken into account. In more complicated models, the activation function also uses the previous output value of the neuron, so that the neuron can self-excite [92]. In most artificial neural networks the activation function is deterministic, but may be stochastic in more complex networks. The activation value is then passed through the neuron transfer function [92, 79].

The transfer function defines how the activation value is output to the rest of the network [92]. In some models the transfer function is a threshold function, or an "all or nothing" function. If the activation value is greater than some threshold amount then the neuron will output a one; conversely an activation value less than the threshold value will result in a zero output. In this model the neuron's activation must reach a certain level before the neuron adds to the total network state. Most common artificial neural networks use a transfer function known as the saturation function in which more excitation above some maximum firing level has no further effect on the output of the neuron [92]. Examples of saturation functions that are widely used in artificial neural networks today are the sigmoid function and the hyperbolic tangent function. These functions yield output which is a continuous, monotonic function of the input. Both the functions and their derivatives are continuous everywhere, and their values asymptotically approach a high and low value, with a smooth transition in between [92]. The sigmoid (logistic) transfer function's output shown in Figure 15 approaches zero when its input is a large negative number, and approaches one when the input is a large positive number. The hyperbolic transfer function's output shown in Figure 15 approaches negative one when its input is a large negative number, and approaches positive one when its input is a large positive number. The sigmoid transfer function is typically employed in those networks which are used for classification, while the hyperbolic transfer function is used in those networks involved in prediction [92, 79].

The mathematical equations of the activation function are:

a. Linear activation function

$$y = x \tag{4.2}$$

b. Sigmoid activation function

$$y = \frac{1}{1 + e^{-x}} \tag{4.3}$$

c. Hyperbolic function

$$y = \frac{1 - e^{-2x}}{1 + e^{2x}} \tag{4.4}$$

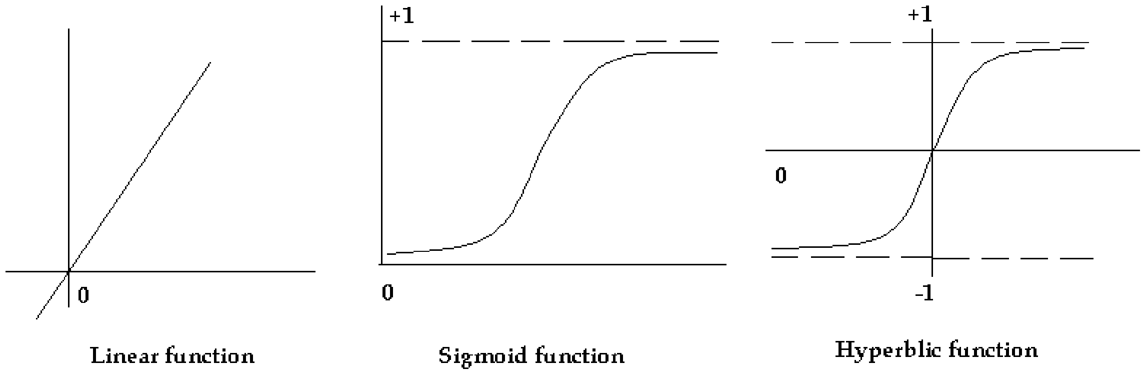
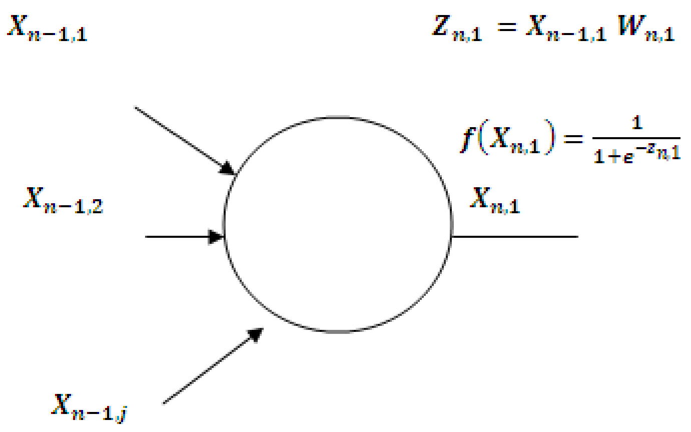


Figure 15 : 2D graphical of common activation function

Basically, the activation functions are mathematical formulae that determine the output of a processing node [104]. Each unit takes its net input and applies an activation function to it. The purpose of the transfer function is to prevent output from reaching very large value which can paralyze neural networks and thereby inhibit training. Transfer function such as sigmoid are commonly used because they are nonlinear continuously differentiable which are desirable for network learning [104]. An artificial neuron that uses the sigmoid transfer function is shown in Figure 16.



$X_{n-1,1}$
$Z_{n,1} = X_{n-1,1}\, W_{n,1}$
$f(X_{n,1}) = \frac{1}{1+e^{-z_{n,1}}}$
$X_{n-1,2}$
$X_{n,1}$
$X_{n-1,j}$

Figure 16 : Backpropagation neuron using sigmoid transfer function [92]

Where: $X_{n,j}$ output of ith neuron in the nth layer and $W_{n,i,j}$ weight of the output of the jth neuron in the (n-1)st layer to the ith neuron in nth layer

The general procedure for backpropagation follows [92]:

1. Initialize weights, $W_{n,i,j}$ randomly
2. Present an input pattern $(X_{1t}, X_{2t} \ldots . X_{nt})$ and a desired output d_t to the network
3. Calculate the actual output for the input pattern $(X_{1t}, X_{2t} \ldots . X_{nt})$, y_t, from the network: $y_t = f[\sum X_{it}\, W_{it}$
4. Compute the total sum of squares error for the network
5. for input t, e_t : $e_t = 0.5 * sum_t(d_t - y_t)$
6. Calculate $\Delta W_{n,i,j}$ (Described in following paragraphs)
7. Feedback: Correct the weights $W_{n,i,j}(new) = W_{n,i,j}(old) + \Delta W_{n,i,j}$
8. Repeat steps one through five for all training patterns
9. Repeat steps one through six until the error is less than some pre-determined tolerance.

The basic formula for changing the weights is:

$$\Delta W_{n,i,j} = alpha * X_{n-1} * e_{n,j}, \tag{4.5}$$

where: X_{n-i} is output from neuron i of layer $n - i$, $e_{n,j}$ error of neuron j in layer n, Alpha is learning rate $(0 < alpha < 1)$.

There are two formulas for calculating a specific neuron's error. The formula for a neuron's error in the output layer is directly proportional to the difference between the desired output and the actual output of the output neuron. It also depends on the derivative of the transfer function for the neuron in the output layer. This formula is [92]:

$$e_{j,out} = f'(z_{j,out}) * (d_j - y_j) \qquad (4.6)$$

The formula for a neuron's error in any layer below the output is proportional to the backpropagated error. This means that the error in these nodes depends on the errors of the nodes above and the connecting weights to the above nodes. The neuron's error in any layer below the output layer also depends upon the derivative of its transfer function at its current output level. This formula is [92]:

$$e_{j,n} = f'(z_{j,n}) * sum(e_{k,n+1} * w_{k,j,n+1}) \qquad (4.7)$$

Thus, the change in an incoming weight is proportional to the error of a neuron times the value of the input on the connection being adjusted. One modification to the backpropagation procedure, developed to avoid local minima in the error structure is the "generalized Delta rule" [92]. This modification adds a momentum term to the change in the $W_{n,i,j}$'s This momentum term is a constant, **β,** multiplied by the weight vector of a neuron from the previous presentation of an input pattern, which is then added to the next change in the weights to avoid local minima in the error structure [92]. The new formula for changing the weights by the generalized Delta rule is:

$$\Delta W_{n,i,j} = alpha * X_{n-1,i} * e_{n,i} + \beta[W_{n,i,j(old)} - W_{n,i,j\{new)prev} \qquad (4.8)$$

Backpropagation is thus able to solve the XOR problem because outputs from the neurons can take on intermediate values between either zero or one (for the sigmoid transfer function), or negative one and positive one (for the Tan H transfer function). This allows a network to slowly readjust its weights in the individual neurons, and to move down the error structure until some preset error tolerance level is reached [92].

The number of applications for multiple layers, backpropagating artificial neural networks is continually increasing. Some of the areas in which they have been used are sonar interpretation, machine vision, converting English text to phonemes, airline seat marketing, and forecasting in the economic and banking areas [92]. They have applications in pattern classification, modeling complex non-linear functions, and signal processing problems. Additionally, they are beginning to see wide use in the field of robotics [92, 81].

4.3.2 Neural network architectures

Neural network architecture defines its structure including number of hidden layers, number of hidden nodes and number of output nodes etc [104].

- Neural of hidden layers:

  The hidden layers provide the networks with its ability to generalize [104]. In theory, a neural with one hidden layers with sufficient number of hidden neurons is capable of approximating any continuous function.

In practice, neural network with one and occasionally two hidden layers are widely used and perform very well.

- Number of hidden nodes:

  There is no magic formula for selecting the optimum number of hidden neurons [104]. However, some thumb rules are available for calculating number of hidden neurons. A rough approximation can be obtained by the geometric pyramid rule. For a three layer network with *n* input and *m* output neurons, the hidden layer would have *sqrt(n*m)* neurons.

- Number of out nodes :

  Neural network with multiple outputs, especial if these outputs are widely spaced, will produce inferior results as compared to a network with a single output [104].

4.3.2.1 Single Layer Perceptrons

The Rosenblatt perceptron was built around the McCulloch-Pitt model of a neuron [105]. The Single Layer Perceptrons (SLPs) are suitable for simple linear separable or linear discriminants problem for pattern classification into one or two classes [105, 106, 107, 108, 109]. The training technique used is called the perceptron learning rule and is capable of learning by generalizing from its training vectors and learning from randomly distributed connections.

The perceptron model is made up of a linear combiner and a hard limit transfer function [105]. A high is produced if the net input is equal to or greater than 0; and 0 if otherwise. The perceptron learning rule is applied to each neuron in order to calculate the new weight and bias. Input vectors are classified by dividing the input space into two decision regions separated by a hyperplane defined by [105]:

$$\sum_{i=1}^{m} wi\, xi + b = 0 \tag{4.9}$$

where m is the number of input variables, $w \in \Re^m$ is the vector of the weight, $x \in \Re^m$ is the vector of the input stimulus, and b is the bias.

Perceptions are trained on examples by using a set of inputs-output pairs where p is a vector of the input to the network is and t is the corresponding correct output target vector as shown in Figure 17 [105].

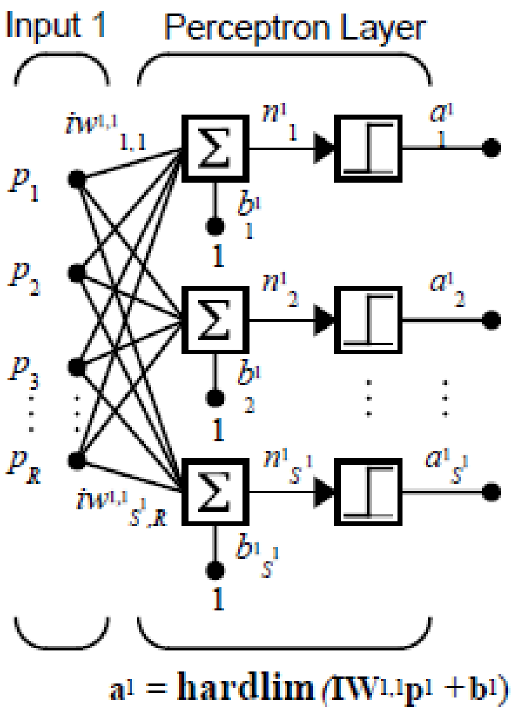


Figure 17 : Perceptrons [110]

4.3.2.2 Multilayer Perceptions (MLP)

The MLP consist of the input layer, hidden layer, and an output layer. The input layer and hidden layer are referred to as source nodes, while the output layers are regarded as computational nodes [105]. The input layer propagates signals through the network in a forward direction from layer to layer. MLP have been reported in the literature to be successful in complex problem application through supervised training based on the back-propagation learning algorithm [105, 106,107].

Figure 18 [108] show a typical example of the MLP. One can see that an input signal propagates forward through the network and emerges at the output end. Also, an error signal is computed at the output of the network and is propagated backward through the network.

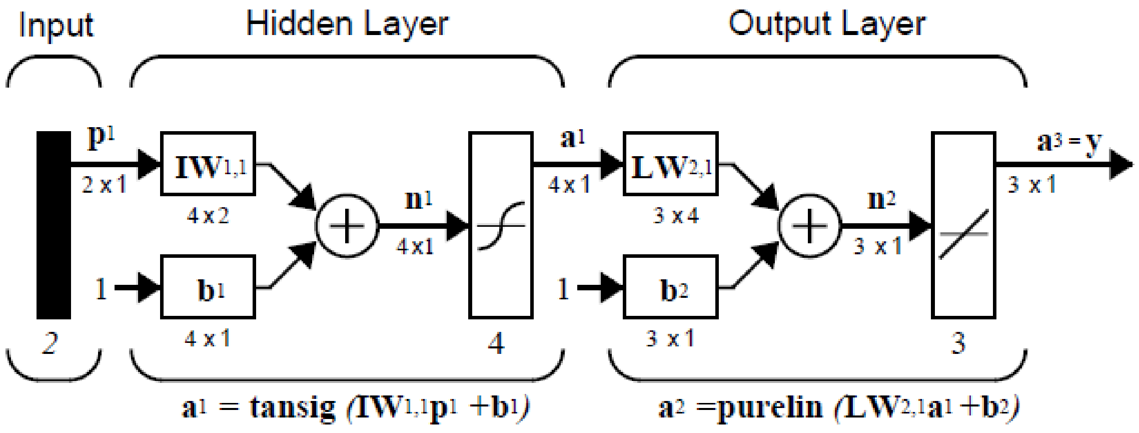


Figure 18 : Multi Layer Perceptron [108]

This forms the basis of the error back-propagation algorithm [105]. The back-propagation learning rule is implemented by adjusting the weights and biases of networks, in order to minimize the error of the network. The value of the network weights and biases are continuously changed in the direction of steepest descent with respect to the error [105].

4.3.4 Neural Network Topology for Modeling Approach of robotic systems

To describe the kinematics and dynamics model of the mobile robot by using artificial neurons as the basic building element for the development of multi-layered and higher order neural network, the five basic steps shown in Figure 19 are used in order to overcome the challenge in the identification and of the mobile robot system.

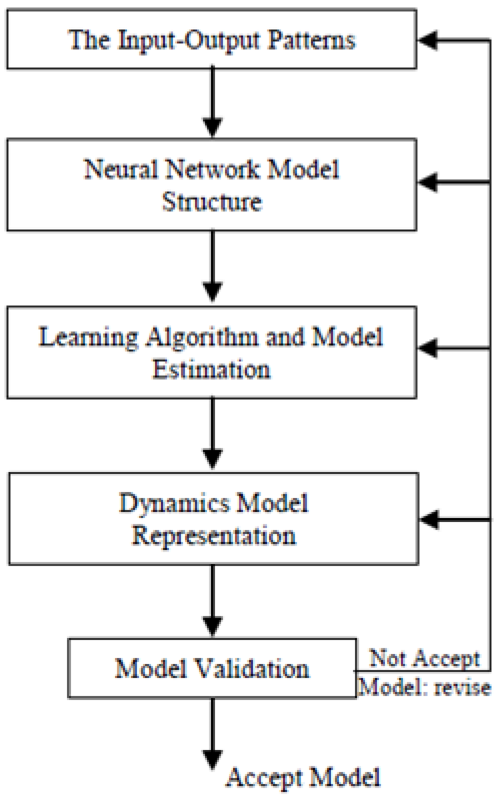

The Input-Output Patterns
Neural Network Model Structure
Learning Algorithm and Model Estimation
Dynamics Model Representation
Model Validation
Not Accept Model: revise
Accept Model

Figure 19 : Steps of modeling and identifying for mobile robot system

4.3.4.1 Types of Neural Network

The most widely NN structures used in this dissertation are the following:

1. Feedforward Neural Networks (FNN)
2. Feedback Recurrent Network (FRN)
3. Elman Networks (ELM)

1. Feed forward backpropagation neural networks

FNN in general consists of a layer of input neurons, a layer of output neurons and one or more layers of hidden neurons [109, 110]. Neurons in each layer are interconnected fully to previous and next layer neurons with each interconnection have associated connection strength or weight [109]. The activation function used in the hidden and output layers' neurons is non-linear, where as for the input layer no activation function is used since no computation is involved in that layer. Information flows from one layer to the other layer in a feedforward manner. Various functions are used to model the neuron activity such as sigmoid, tanh or radial (Gaussian) functions [109]. Figure 20 shows a feed forward neural network.

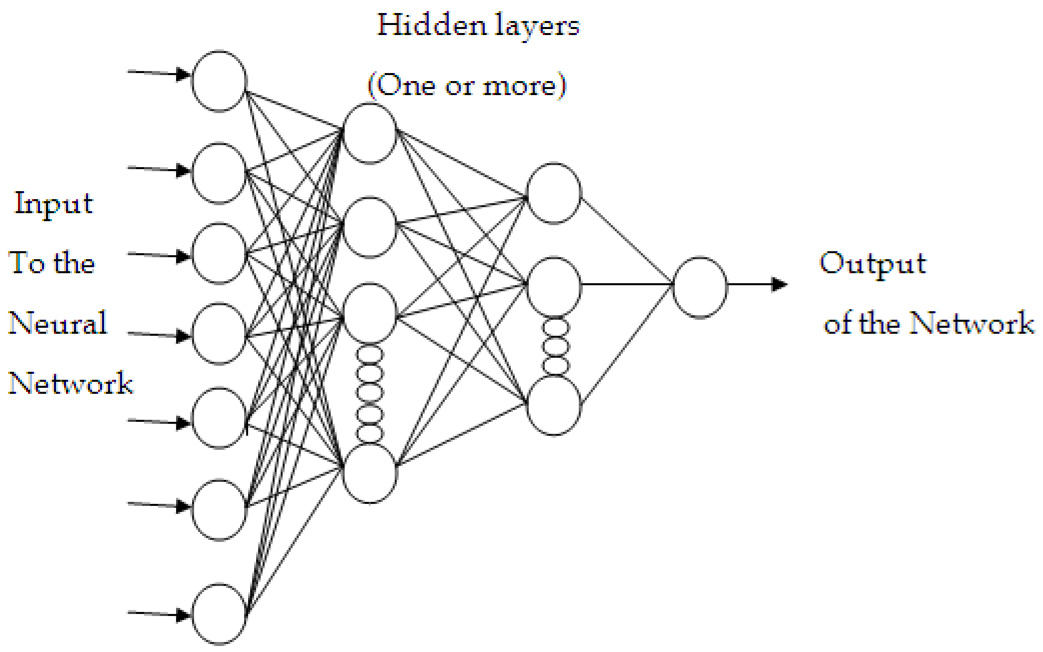


Figure 20 : Feed forward neural networks

The input to a node i in the k^{th} layer is given by [109, 111]:

$$net_{i,k} = \left[\sum_j w_{i,j,k} out_{j,k-1} \right] + \theta_{i,k} \tag{4.10}$$

Where, $w_{i,j,k}$ represents the weight connection strengths for node j in the $(k - I)^{th}$ layer to node i in the k^{th} layer, out i, k is the output of node i in the k^{th} layer and $\theta_{i,k}$ is the threshold associated with node i in the k^{th} layer.

2. Feedback-Recurrent Network (FRN)

The next dynamic network to be introduced is the FRN. An earlier simplified version of this network was introduced by Elman [109]. In the FRN there is a feedback loop, with a single delay, around each layer of the network except for the last layer. The original Elman (ELM) network had only two layers, and used a tansig transfer function for the hidden layer and a *purelin* transfer function for the output layer [109]. The original Elman network was trained using an approximation to the backpropagation algorithm. The *newlrn* command generalizes the Elman network to have an arbitrary number of layers and to have arbitrary transfer functions in each layer [109]. Various toolbox softwares trains the FRN using exact versions of the gradient-based algorithms. Figure 21 shows two layers- FRN [109, 112].

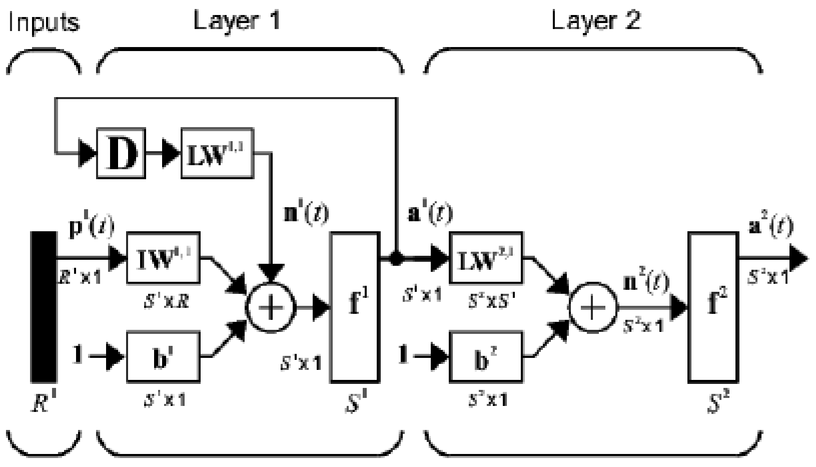


Figure 21 : Two Layer feedback-recurrent neural network [112]

3. Elman recurrent neural network

The Elman network (ELM) is commonly a two-layer network with feedback from the first-layer output to the first-layer input [109]. This recurrent connection allows the Elman network to both detect and generate time-varying patterns. A two-layer ELM network is shown in Figure 22.

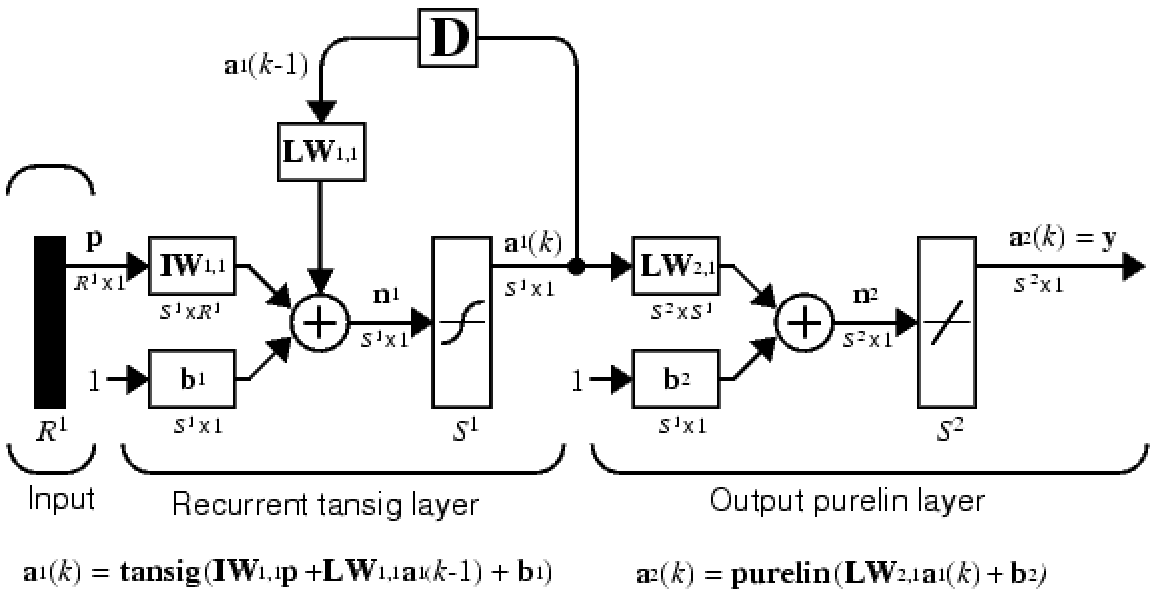


Figure 22 : Elman recurrent network [112]

The ELM has tansig neurons in its hidden (recurrent) layer, and *purelin* neurons in its output layer [109]. This combination is special in that two-layer networks with these transfer functions can approximate any function (with a finite number of discontinuities) with arbitrary accuracy. The only requirement is that the hidden layer must have enough neurons. More hidden neurons are needed as the function being fitted increases in complexity. Note that the ELM differs from conventional two layer networks in that the first layer has a recurrent connection [109]. The delay in this connection stores values from the previous time step, which can be used in the current time step. Thus, even if two ELM, with the same weights and biases, are given identical inputs at a given time step, their outputs can be different because of different feedback states. Because the network can store information for future reference, it is able to learn temporal patterns as well as spatial patterns. The ELM can be trained to respond to, and to generate, both kinds of patterns [109, 112].

4.4 Neural networks training procedure in *MATLAB®* software

In a typical supervised learning scenario, a training set is given and the goal is to form a description that can be used to predict problem. Thus, in this section Matlab will deals with two neural networks architectures - multilayer feed forward and Elman recurrent respectively, then end of the section with other software called BPnet which specializes in backpropagation technique. The process of training the network involves set of "training sets" that show the proper network behavior and target outputs. For the analysis of neural networks, a different training algorithm will be implemented for a given problem. These algorithms include learning methods such as Backpropagation, conjugate gradient algorithm, Quasi-Newton algorithm, etc.

4.4.1 Feedforword neural network training procedure

The collected data needs to be distributed on training and testing set. Generally, different variables are represented in different order of magnitude; thus, in order to ensure that every data unit receives the same influence in the training procedure, it needs to be normalized. In this work for, the data is divided in training and testing randomly in ratio 7:3. In other words, the Matlab training set includes 70% of 1320 sensor measurements (88 instances x 15 samples in each instance), randomly chosen over the entire dataset. The rest of the data is used to test the NN network. In addition, a small portion of data is replaced with erroneous samples; 0 to 3 percent of real sensor measurement is randomly replaced in the training and testing process.

4.4.1.2 Neural network definition in Matlab

In this dissertation, multilayer feedforward NNs with different learning algorithms are used. The tested architectures include 1, 2, 3, or 4 hidden layers. The NN input parameters are 6 scaled values (i.e one sample) of forces and torques from the dataset. The output neurons represent 4 possible cases that correspond to the particular input: normal, collision, obstruction or front collision [5]. Therefore, the prediction problem is formulated as a classification problem (similarly to [49, 3]), which is solved by developing a novel approach using NNs. In Matlab implementation, sigmoid and linear activation functions are used in hidden and output layer, respectively. The leaning parameter is set to be 0.5 for all networks.

4.4.1.3 Architectures of selected NNs

To obtain the optimal NN, we need to test various architectures. So far, there is no explicitly determined rule or pattern for selection of the number of the hidden layers reported in the literature. Likewise, the selection of number of neurons is not universally determined. This is usually done empirically, although there are different advices for solving this problem [1]. In this work different architectures were investigated, including the networks with one, two or three hidden layers. The network structure marked as means that there are neurons in the first hidden layer, in the second hidden layer, and in the third hidden layer. As mentioned, the NN input and output are single column vectors since they represent scaled values of recorded sensor measurements and corresponding robot situations [5]. Employed network architectures are listed in Table 1.

Table 1 : NN architecture used in experiments in this thesis

No.	NN architecture	No.	NN architecture	No.	NN architecture	No.	NN architecture
1	1	7	1-1	13	2-2-2	19	3-3-3-3
2	2	8	2-2	14	3-2-2	20	4-3-3-3
3	3	9	3-2	15	4-3-2	21	5-4-3-3
4	5	10	5-2	16	5-3-2	22	8-5-4-3
5	8	11	8-4	17	8-3-2	23	10-8-4-3
6	10	12	10-4	18	8-4-2	24	10-8-5-4

4.4.1.4 Algorithms selection

After determining the architectures listed in Table 1, several learning algorithms are employed in order to investigate the best possible NN behavior. The specific problem under consideration represents the main problem in algorithm selection (i.e. the problem mainly influences the performance of the learning algorithms). Thus, the same algorithm can have different performance depending on the considered task. Therefore, we tested in Matlab all the main algorithms that proved to have best performance over classification, pattern recognition or nonlinear function approximation in order to find optimal solution for the problem of robot failure prediction. Likewise, we tested one of the most popular gradient descent algorithms outside of Matlab so as to discover the best NN based prediction method. These seven algorithms used in Matlab with corresponding acronyms are listed in Table 2. Note that these acronyms and NN ordinal numbers will be used in the next section. As stated before, in Matlab we use sigmoid and linear activation function in the hidden and output layer, respectively.

Table 2 : Learning algorithms used in experiments in this thesis

Neural Network Type	No.	Learning Algorithm	Software	Acronym
Feedforward	1	Levenberg–Marquardt	Matlab	LM
	2	Bayesian Regularisation	Matlab	BR
	3	Resilient Backpropagation	Matlab	RP
	4	Scaled Conjugate Gradient	Matlab	SCG
	5	BFGS quasi–Newton Backpropagation	Matlab	BFG
	6	Variable Learning Rate Backpropagation	Matlab	GDX
	7	Gradient Descent Backpropagation	BPnet	BP
Recurrent	8	Elman NN	Matlab	ELM

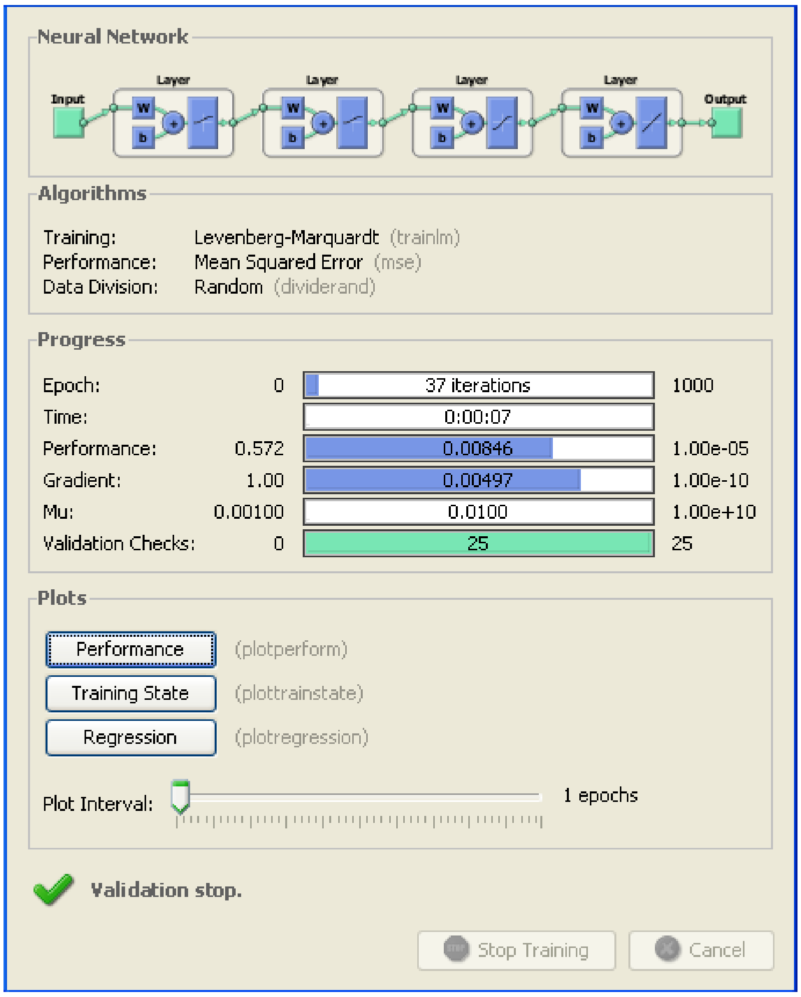
4.4.1.5.1 Mean-squared error and training in Matlab

The stopping criterion for NN training in Matlab software is defined in terms of goal MSE or maximum number of learning iterations. These values are defined to be 10 (MSE) and 1000 (maximum iterations). Nevertheless, the experimental results show that this difference in MSE has no crucial influence in the overall prediction outcome [5].

The NN prediction performance is evaluated using the MSE (equation 2) on test data in Matlab. The NN ability to predict execution failures is tested several times for each architecture and learning algorithm. The best results are obtained and presented in the following sixth chapter

$$MSE = \frac{\sum_{i=1}^{N}(y_i - O_i)^2}{N} \qquad (4.11)$$

where y_i is the NN _output and O_i is the target_output and N number of data set. Figure 23 shows an example of neural networks training implantation using Levenberg - Marquardt algorithm in Matlab software.



Algorithms

Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Data Division: Random (dividerand)

Progress

Epoch:	0	37 iterations	1000
Time:		0:00:07	
Performance:	0.572	0.00846	1.00e-05
Gradient:	1.00	0.00497	1.00e-10
Mu:	0.00100	0.0100	1.00e+10
Validation Checks:	0	25	25

Plots

Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

Plot Interval: 1 epochs

✓ Validation stop.

Figure 23 : MATLAB training networks

4.4.2 Elman neural network training procedure

In this work, another type of neural network - Elman NN with different architectures is employed. This is done in order to obtain overall best solution and to make comparison with feedforward NNs for the problem of failure prediction. Likewise, the training architectures include 1, 2, 3, or 4 hidden layers. The NN input parameters are 6 scaled values (i.e. one sample) of forces and torques from the dataset. The output neurons represent 4 possible cases that correspond to the particular input: normal, collision, obstruction or front collision. The training procedure in terms of Mean Squared Error (MSE) for Elman NN is the same as training procedure for feedforward NN. In order to evaluate and to find optimal NN, the architecture and algorithm are training several times, the best performance will be presented in the Chapter 6.

4.5 Neural networks training procedure in BPnet software

The software **BPnet** employs backpropagation technique and is developed in the Laboratory for Industrial Robotics and Artificial Intelligence at the Faculty of Mechanical Engineering in Belgrade, primary for the needs of implementing sensor-motor coordination of learning robot and camera calibration [50]. Wide ranges of applications using BPnet are established; for example, it is involved in the domain of intelligent robot control [41] as well as for predicting professional choices of secondary school graduates [3]. In this dissertation, the basic idea for BPnet engagement was to test our method using two independent software packages. In that way, the obtained results are more credible regarding prediction problem solved by NNs. Likewise, the software proved to be very useful in previous applications in the robotic domain [50,3], which represents an additional argument for its utilization.

BPnet software was developed in *Visual Basic* programming language [100]. User friendly interface of the software enables that NN topology and initial weighting coefficients are easily defined. The starting window Figure 24a shows basic information about the software. After the *„proceed"* button is pressed and the project is named, four different steps for defining backpropagation NN in BPnet software are available. The training procedure is explained in the next four steps.

Firstly, one must define the number of layers and number of neurons in each layer in the "configuration" step. By using three buttons located on the top of the window Figure 24b, each NN architecture can be easily set. The end of this step is conducted using the "check" mark on the left side. Likewise, by pressing on the "x" button mis-entered neurons in each layer can be deleted.

Secondly, the weighting coefficients are defined in the next module – in the "connections" step. Initial weighting coefficients are defined by default and are given in Figure 24c. These can be varied in order to obtain their optimal value for the problem in hand. This stage is completed in the same manner as previous - by confirming weighting coefficients and bias values using the "check" mark.

Third step implies implementation of input/output training pairs. In this module we can also open earlier work or save a new training data in a text file.

Finally, the training and testing phase is conducted in the "train" module. Here, we can define expected (i.e. goal) middle absolute error (MAE) as well as network learning parameters. Testing is also conducted here by invoking the new input/output pairs after the training phase is over. An example of the BPnet engagement during the training process with NN architecture 6-10-4 (6 neurons in input layer, 10 and 4 in hidden and output layer, respectively) is given in Figure 24d. One can notice that the backpropagation algorithm successfully decreased the NN error below the previously defined value after ~25.000 iterations.

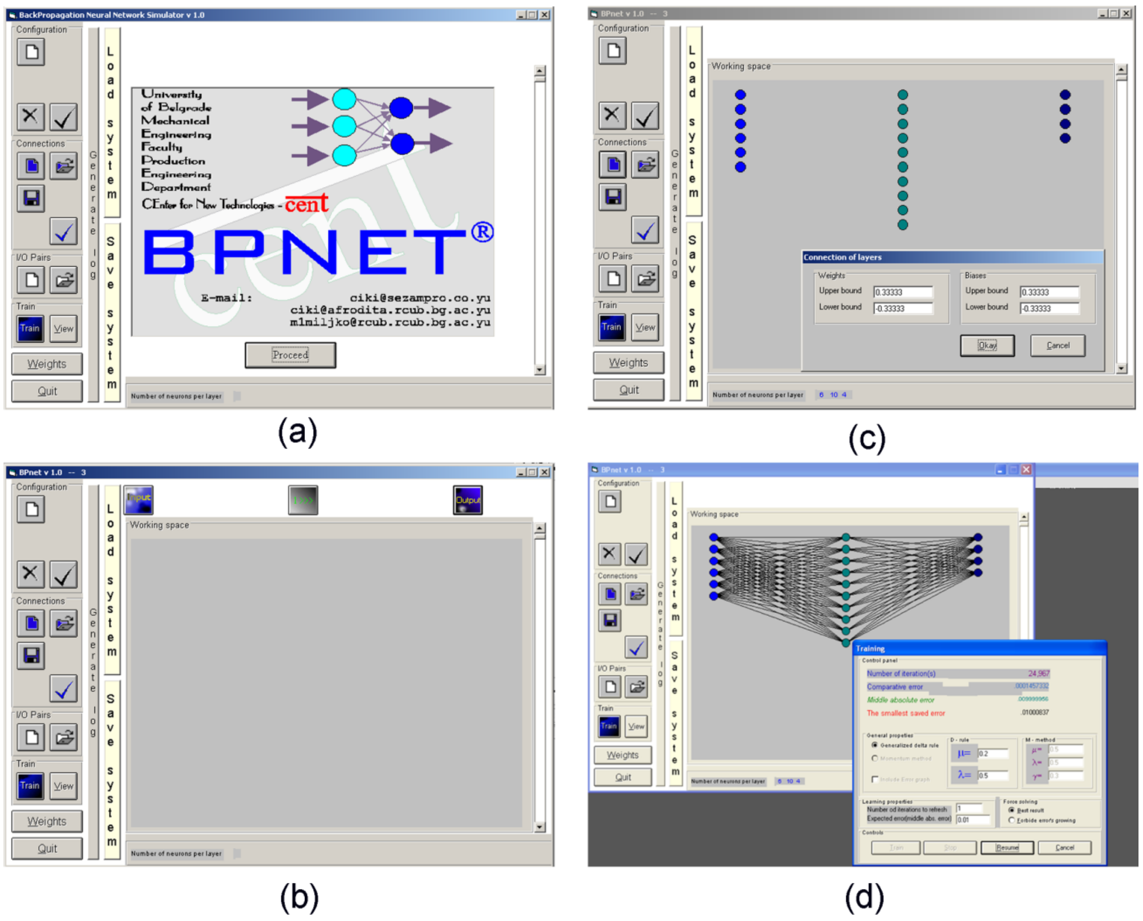


Figure 24 : BPnet software: (a) starting window, (b) “configuration” module, (c)“connections” module, (d) training process [5]

4.5.1 Training set for BPnet –software

For BPnet software, the training and testing set are much smaller in order to significantly reduce the computational cost and to speed up the NN training. In total, we use 64 randomly chosen sensor measurements, divided into the training and testing data in the same ratio. The bad data is implemented here in the same manner as before (0-3 percent of total training data) [5].

4.5.1.1 Data normalization

Before the start of the training phase, input and output data were scaled to be between the upper and lower bounds of transfer functions ,the data is scaled in the [-1 1] interval. Normalization of data helps artificial networks to better understand the relationship between input and output data as well as increasing the accuracy of prediction so high efficiency will be achieved during testing step [113]. This is done so as to enhance the network performance and to speed up the learning process. The following equation is used in this purpose [1]:

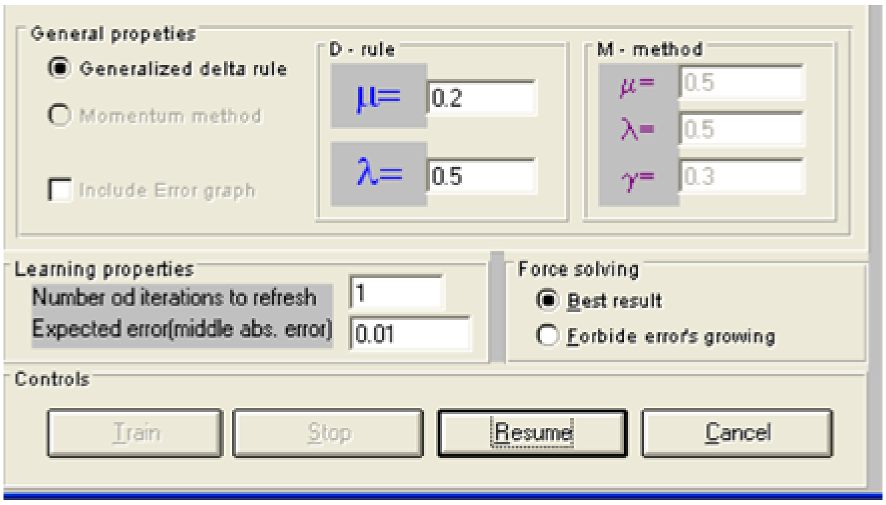
$$x_{scaled} = \bar{x}_{\min} + \frac{x - x_{\min}}{x_{\max} - x_{\min}} \cdot \left(\bar{x}_{\max} - \bar{x}_{\min}\right) \tag{4.12}$$

Where x_{scaled} denotes scaled data value, $\bar{x}_{\min}$ and $\bar{x}_{\max}$ are minimum and maximum values in chosen range (i.e. -1 and 1, respectively), $x_{\min}$ and $x_{\max}$ represent minimum and maximum values to be scaled, respectively.

4.5.1.2 BPnet – software parameter settings

In software itself, sigmoid function was used as an activation function with delta rule as the learning rule, and with parameters for all networks as given below and in Figure 25:

- Momentum λ = 0.5
- Learning parameter μ = 0.2
- Expected error MAE = 0.01


General propeties
[● Generalized delta rule]
○ Momentum method
☐ Include Error graph

D - rule
$\mu=$ 0.2
$\lambda=$ 0.5

M - method
$\mu=$ 0.5
$\lambda=$ 0.5
$\gamma=$ 0.3

Learning properties
Number od iterations to refresh: 1
Expected error(middle abs. error): 0.01

Force solving
[● Best result]
○ Forbide error's growing

Controls

Figure 25 : BPnet software training control panel

An example of a neural network with 5 neurons in first and 2 neurons in second hidden layer is presented in Figure 26. Input values are measured forces and torques in robotic system, while outputs represent four cases that correspond to these forces. As mentioned before, the prediction problem is transformed into classification problem, in which defined failure case must be identified according to the values of the forces. This is extremely important in various industrial tasks: the detection of the failure should be quick in order to prevent further damage or malfunctioning of the whole system.

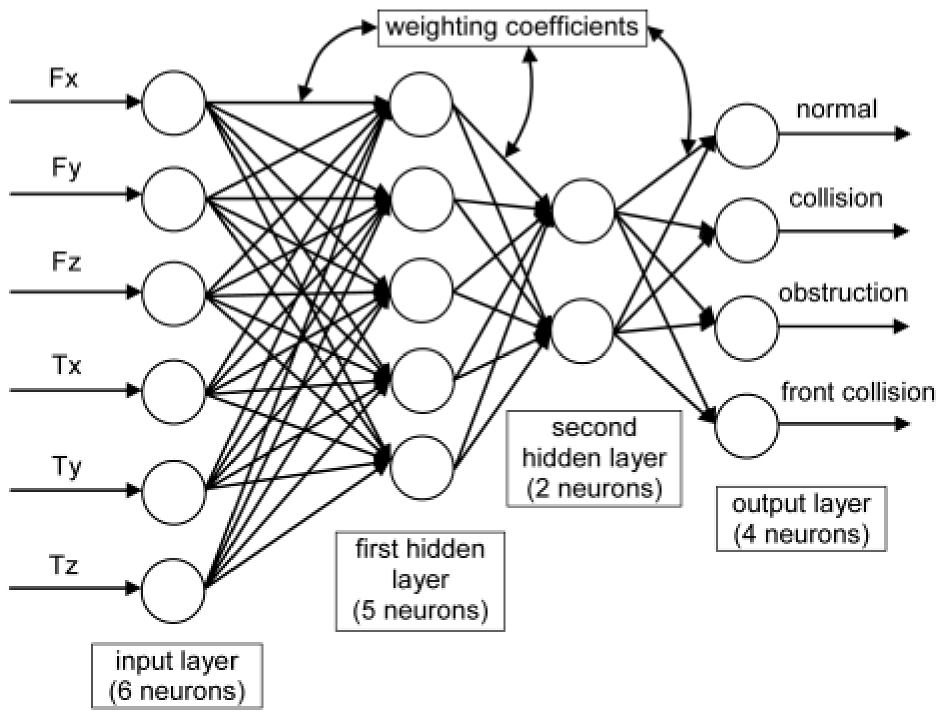


Figure 26 : Structure of the multilayer feedforward NN [5]

4.5.2.2 Testing criteria for BPnet software

The NN prediction performance in BPnet and Matlab is also evaluated using the mean squared error - MAE (equation (4.12)) on test data. The goal MAE is set to be 0.01.

$$MAE\_test = \frac{1}{n}\sum_{i=1}^{n}|NN\_out - target\_output| \tag{4.13}$$

The NN ability to predict execution failures is tested for each architecture. The best results are obtained and presented in the chapter 6

4.5.3 Prediction procedure summary

In this section, the prediction procedure is concisely given again. This is crucial part of experimental initial setup, so it is necessary to emphasize it again. In order to evaluate the NN performance, the collected data needs to be distributed on training and testing set. Generally, different variables are represented in different order of magnitude; thus, in order to ensure that every data unit receives the same influence in the training procedure, it needs to be normalized. In this work, the data is divided in training and testing randomly in ratio 7:3. In other words, the Matlab training set includes 70% of 1320 sensor measurements (88 instances x 15 samples in each instance), randomly chosen over the entire dataset. The rest of the data is used to test the NN network. In addition, a small portion of data is replaced with erroneous samples; 0 to 3 percent of real sensor measurement is randomly replaced in the training and testing process.

For BPnet software, the training and testing set are much smaller in order to significantly reduce the computational cost and to speed up the NN training. In total, we use 64 randomly chosen sensor measurements, divided into the training and testing data in the same ratio. The bad data is implemented here in the same manner as before.

Multilayer feedforward NNs with different learning algorithms are used in this dissertation. The tested architectures include 1, 2, 3, or 4 hidden layers. The NN input parameters are 6 scaled values (i.e. one sample) of forces and torques from the dataset. The output neurons represent 4 possible cases that correspond to the particular input: normal, collision, obstruction or front collision. Therefore, the prediction problem is formulated as a classification problem (similarly to [49, 3]), which is solved by developing a novel approach using NNs.

5. Intelligent mobile robot in a manufacturing environment

5.1 Introduction to intelligent manufacturing systems (IMS)

Basic definition of IMS is [116]: Intelligent manufacturing system presents system with autonomous ability to adapt to unexpected changes, i.e. change of assortment, market requests, technology changes, social needs etc. In specific type of construction of IMS should be cared about following requests [116]:

a) low production costs,
b) universality, adaptation of production system to specific product,
c) precision and high quality of manufactured products,
d) expressive shortening of main and incidental production times,
e) exclusion of man in production process,
f) Safety.

With growth of requirements to manufacturing systems, come other criteria, which would widen abilities of manufacturing system. Requirements can be defined by changing character of production.

Goal is to create such a system, which is capable to react flexible to various situation in production process [116]:

a) to change of shape of manufactured product,
b) change of measurement properties of product,
c) packing of subsystems with components,
d) unexpected switch to different type of products,
e) time variation in production process,
f) change of technological parameters,
g) securing against crash situations.

Intelligent manufacturing system is possible to consider as higher phase of flexible manufacturing systems [116]. Intelligent manufacturing systems like flexible manufacturing systems consist of individual subsystems (technological, transportation and handling, control, store and operative). Each subsystem has to contain of intelligence elements, which give to these subsystem certain degree of intelligence.

To the basic elements of machining intelligence belong visualization of production process (monitoring), which enables to observe own status of system and changing conditions of environment. Primary information for realization of production tasks in required order, come in to the operative system of IMS over basic elements of machining intelligence – over sensorial elements, which expressively increase degree of intelligence of manufacturing system [116].

5.1.1 Components of an intelligent manufacturing system

As mentioned in the previous section, the manufacturing process is a complex one and can be decomposed into several components. In [117] Rao decomposed IMS into the following components: intelligent design, intelligent operation, intelligent control, intelligent planning and intelligent maintenance. We modify this decomposition slightly to reflect the current trends in the literature on IMS as shown in Figure 27. This now fully reflects current understanding of a modern IMS.

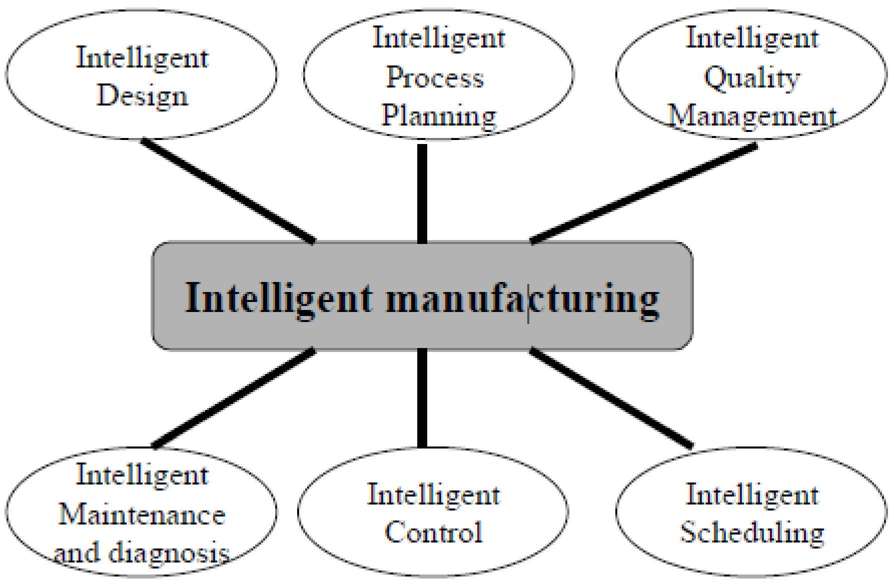

Intelligent Design
Intelligent Process Planning
Intelligent Quality Management
Intelligent manufacturing
Intelligent Maintenance and diagnosis
Intelligent Control
Intelligent Scheduling

Figure 27 : Components of an intelligent manufacturing system [117].

5.1.2 Intelligent Mobile Robots in manufacturing systems

At the beginning of the 21st century manufacturing is more closely than ever related to fast growing market requirements and intensively coupled with diverse customer demands [118]. The increasing complexity of products and growing tendency for delivery time cutting as well as the need for "make to order" rather than "make to stock" manufacturing, imposes newly developed solutions able to tackle with these sophisticated issues. New methods, fast growing research fields, design principles and newly developed and defined paradigms, guarantee improvement of the existing technology as well as the quality of everyday life [118].

Intensive research in the field of robotics has resulted in a great number of robots able to perform complex and sophisticated tasks they had been previously designed to do [118]. Throughout years robotics has achieved a number of important great successes in various fields of application such as manufacturing, museum touring, cargo handling etc. However, one of the greatest successes to date is in the world of industrial manufacturing where industrial manipulators are able to move with great speed and accuracy performing all sorts of tasks, such as welding, painting, cutting etc. [119]. Needless to say, implementation of industrial robots for manufacturing purposes is a standard for highly-developed companies [118].

The implementation of Intelligent Mobile Robots in manufacturing systems [120] is still a challenge for the research community. Operating on the shop floor, as a component of material transport system, Intelligent Mobile Robots would need a particular kind of behavior exclusively developed for these purposes [118].

5.2 Control of a mobile robot using AI techniques

The traditional control methods for mobile robots have used linear or non-linear feedback control [121] while artificial intelligent controllers were carried out using neural networks [122, 123], evolutionary algorithms [124], or fuzzy inference [125]. Neural networks are recommended for AI control as a part of a well-known structure [126]. Much research has been done on the applications of neural networks for control of nonlinear of mobile robot systems and has been supported by two of the most important capabilities of neural networks: their ability to learn and their good performance for the approximation of nonlinear functions [126]. The neural network based control of mobile robots is usual to work with kinematic models of mobile robot to obtain stable motion control strategy for goal reaching [126, 127]. The NNs in demined of an adaptive dynamics control of nonholonomic mobile robots was addressed in [126, 128]. For tracking control of wheeled mobile robot new method by using two cascade controllers is proposed in [129]. Second subsystem is the main one, and consists of adaptive neural fuzzy inference system controller for the direct solution of trajectory tracking problem of mobile robots.

5.2.1 Neural networks for obstacle detection and avoidance

There are always static or dynamic obstacles in the environment [130]. Hence, robotics needs to autonomously navigate themselves in environments by detecting or avoiding obstacles. The neural networks, which have been designed for obstacle detection by mobile robots, should take the sensor data from the environment as their inputs, and output the direction for the robot to proceed [130]. In [131] authors presented a multilayered neural network with error backpropagation through Q-learning for mobile robot obstacle avoidance in unknown environment. In [132] obstacle detection and avoidance problem of a mobile robot in unknown environments is addressed by C. Silva using MONODA (MOdular Network for Obstacle Detection and Avoidance), which consists of four three-layered feedforward neural network modules and every module detects the probability of obstacles in one direction of the robot [130]. Parhi et al. presented a approach of real-time obstacle-avoidance, and wall-following tasks using separate neural networks to solve each of the target seeking, [133, 134]. In their approach, based on certain criteria one of the networks is selected at each time step to control the mobile robot allowing it to move safely in a crowded real world and unknown environment and to reach a specified target while avoiding static as well as dynamic obstacles.

5.2.2 NNs for trajectory tracking and control

NNs have been known for being good approach for solving complex control problems. The control using NNs is generally based on learning ability of the mobile robot [135]. The control of neural network for trajectory tracking of mobile robots has been addressed by Fierro et al [136], which refers to the control of neural network for trajectory tracking based on the neural network function approximation property and can deal with unmodeled bounded disturbances and unstructured unmodeled dynamics of the mobile robot. The neural network is combined with the backstepping controller to learn the full dynamics of the mobile robot and convert the velocity output of the backstepping controller to a torque input for the actual vehicle. The advantage of having neural networks in this approach is that there is no need to know the dynamic model of the robot and the neural network will learn it online without a priori knowledge of the dynamics [135].

In [137], J. Ye presented control of neural network for trajectory tracking uses the learning property of the neural network to make an adaptive controller which adapts the backstepping controller gains [135]. The approach has the properties to quickly drive the position error to zero and to indicate better smooth movement in the tracking performance process. This control approach integrated the backstepping controller with compound orthogonal networks and improves its performance by using the learning property of the neural network [135].

In [138] adaptive control methods for trajectory tracking of a wheeled mobile robot in dynamics level is given; in other words, the mobile robots with unknown dynamic parameters in proposed [139]. Adaptive controls are derived for mobile robots, using backstepping technique, for tracking of a reference trajectory and stabilization to a fixed posture. For the tracking problem, the controller guarantees the asymptotic convergence of the tracking error to zero [139]. For stabilization, the problem is converted to an equivalent tracking problem, using a time varying error feedback, before the tracking control is applied. The designed controller ensures the asymptotic zeroing of the stabilization error. The proposed control laws include a velocity/acceleration limiter that prevents the robot's wheels from slipping [139]. An artificial potential field is used to navigate the wheeled robot in the controller [140]. Easy design, fast convergence, and adaptability to other nonholonomic mobile are obvious advantages. In contrast to adaptive certainty equivalence controllers for mobile robots, the proposed control law takes into consideration the estimates' uncertainty, thereby leading to improved tracking performance [140]. Finally, novel approaches test various learning algorithms and architectures so as to find optimal NN for mobile robot trajectory tracking problem [141].

5.2.3 NNs Control Methodology

The control of a nonlinear mobile robot depends on the information available about the system and the control objectives [142]. The information of the unknown nonlinear system can be determined by the input-output data only and this system is considered through the implementation of feedforward neural networks which are considered in this dissertation. The first step in the procedure of building the control structure is the identification of the kinematical mobile robot from the input-output data, and then a fee forward kinematical neural networks controller is used because the robustness of feedforward NNs enable achieving good tracking of the reference trajectory. The control mobile robot using NNs consists of [142]:

1- Position and Orientation Neural Networks Predictor.

2- Feed forward Kinematics Neural Networks Controller.

5.3 Localization of a mobile robot in a laboratory model of manufacturing environment

The robot also needs the information about its position in the world. One can think about different ways to express this information [143]. It could be in relation to some global coordinate system, but it could also be relative to some object. A combination is likely to be needed as every physical contact requires the robot to position itself relative to the object, whereas the robot will need its global position when reasoning about how to go from one place to another [143].

A problem that is sometimes difficult for a human being as well, Is the problem of finding the position when there is no information about the history of movements [144]. That is, there is no information about how the present position was achieved. This is the problem of initializing the position of the robot. Initializing the position is more difficult than keeping track of the position when the initial position is known. Traditionally, most robot systems have only shown position tracking capabilities and have relied on manual initialization [144]. This is not adequate if full autonomy is one of the goals. In order to do anything meaningful a model of the world is needed. This model, or map, can be of man different types, the way the map is acquired also varies from system to system, but for a fully autonomous system the robot must acquire the map on its own.

Localization can be separated into two sub problems as follows [144]:

- **Position tracking**

  Or position estimation refers to the problem of estimating the location of the robot while it is moving. Drift and slippage reduces the precision of the robot position within its global map.

- **Global localization**

  It is the problem of determining the position of the robot under global uncertainty. This problem arises, for example, when a robot uses a map that has been generated previously and when it is not informed about its initial location within the map.

If the operate of mobile robot in such a dynamic environment, like manufacturing environment, it must be able to determine its position and orientation. Therefore, most robotics problems ultimately should provide answers to the following questions [145, 39]

- Where am I?
- Where have I been?
- Where am I going?
- What's the best way there?

The first two questions is the localization and map making, fall in the realm of mobile robot localization. Mobile robot localization is the problem of determining the pose (position and orientation) of a robot relative to a given map of the environment, and quite often is referred to as the pose estimation problem. The third and the fourth questions are related to planning and control ability of a mobile robot [118].

5.3.1 Odometry and mobile robotics

Odometry is a method of localization for land vehicles and the general ability for any system to know its own position [146]. It is therefore an issue of primary importance in autonomous mobile robotics. Although it's not always necessary to know the robot's position to reach a certain goal, it's useful in many applications such as trajectory tracking, path following and map building [146]. The basic idea of odometry is to retrieve information from different sensors and process it to estimate the position of the robot. The odometry is known as dead reckoning as well (derived from deduced reckoning) and can be expressed through following:

- Odometry is used by mobile robots to estimate their position relative to a starting location.
- Uses data from the rotation of wheels or legs to estimate change in position over time.
- Often very sensitive to error.
- Rapid and accurate data collection, equipment calibration, and processing are required in most cases for odometry to be used effectively.

Implementation of odometry consists of repeated use of wheel counters in order to update the pose of the robot. The pose of mobile robot is calculated in the global coordinate frame, i.e. the pose of the robot is made of three values: (x, y, θ) where x and y are the absolute cartesian coordinates, and θ the orientation of the robot measured from x axis, as shown in Figure 28.

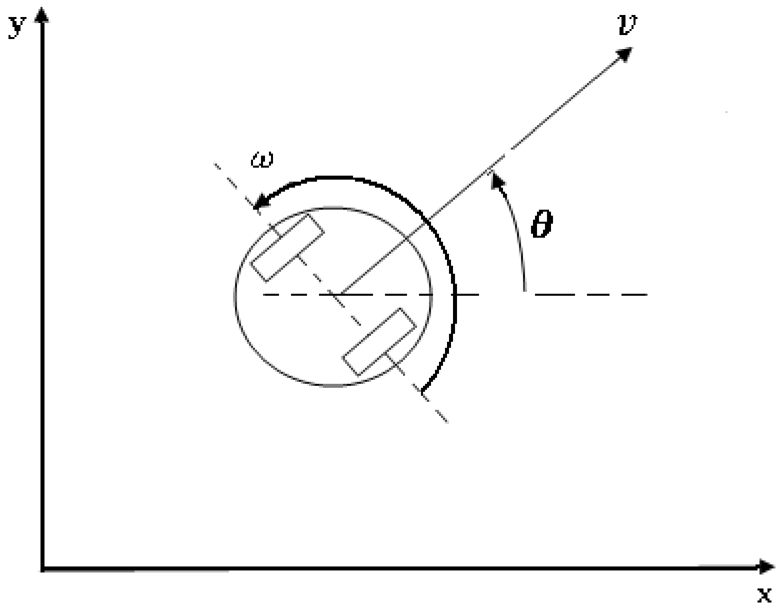


Figure 28 : The position of the mobile robot in the plane

The kinematic model of the trajectory tracking for an autonomous vehicle is introduced next. The position of the mobile robot in the plane is shown in Figure 28. The plane of motion and the moving frame is attached to the mobile robot [147].

The position of the mobile robot in the base frame is expressed as:

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{5.1}$$

And the rotation matrix expressing the orientation of the base frame b with respect to the moving frame m is given by:

$$R(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

The robot motion is obtained by driving the independent active wheels and providing of the two independent wheels velocities, $wl(t)$ and $\omega r(t)$, or the body linear and angular velocities, $v(t)$ and $\omega(t)$, which can be converted in terms of each wheel velocity.

Taking a mathematical modeling for this motion, we can consider two input variables: $v(t)$ and $\omega(t)$, and three state variables: the robot position and orientation $(x(t), y(t), \theta(t))$:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} cos\theta(t) & 0 \\ sin\theta(t) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} \tag{5.3}$$

where: $\dot{x}(t), \dot{y}(t)$, and $\dot{\theta}(t)$ are derivatives of $x(t), y(t), \theta(t)$ respectively.

The input variables are also called control variables, and constitute the way to command the robot so as to provide the desired motion. The robot motion history, i.e. the executed trajectory, is recorded to plot graphics that allow easier visualization of motion topological properties [148].

5.3. 2 Sensors for Localization

A mobile robot must to identify where it is or how it got there, or to be able to reason about where it has gone, sensors are necessary for measuring the distance of wheels have traveled along the environment and also measuring inertial changes and external structure [130]. In order to implement a mobile robot, the robot needs to be equipped with sensors. A robot can be equipped with numerous sensors. Additionally In order to fully understand the problem of localization it is very important to know the characteristic of the sensor that is available [130]. The most common sensor is IR sensor, as it's the sensor which this thesis is dealing with in this chapter by implement real experiment and focuses of obstacle detection problem and related to main approach. The sensors such as infrared sensors provide the external information about the environment. The data from sensors can be applied for recognizing a place or a situation, or be used to construct a map of the environment. Infrared sensor can obtain distant and directional information about an object and needed to get information about the environment. As we humans have different senses, there are sensors which measure different entities, such as color, distance, light, etc [130]. From a localization point of view, it is also important to understand how the sensors work as they are the input to the algorithms, i.e. we need good models for the sensors. In [149] Durrant-Whyte et al say: We will maintain that the only way to understand and utilize the disparity between different sensor views is to explicitly model the sensor and the information it provides...", where a sensor model is defined as an abstraction of the physical sensing process whose purpose is to describe the ability of a sensor to extract descriptions of the environment in terms of the information available to the sensor itself [144]. Crowley [150] says that the sensor model can be viewed as a form of logical sensor which provides the sensor information in a standard form". Independent of the definition used for a sensor model, it is clear that good physical understanding is needed to construct such a model. Borenstein et al presented out that the odometric information is very good most of the time [151]. That is, trusting in an odometric model is warranted almost all the time [144].

The neural networks have many processing units, they provide of robustness or fault tolerance for interpretation of the sensor data [130, 152]. Feedforward multi-layer perception neural network, trained by the backpropagation algorithm, has been applied for pattern classification, pattern recognition and function approximation. In [153] Thrun has employed a feedforward neural network to "translate" the readings of sonar sensors into occupancy values of each grid cell for building metric maps [130].

5.3.3 Approach employed in this PhD dissertation

In order to verify our work, this section provides procedure related to real world experiments are conducted on a mobile robot for obstacle detection and trajectory tracking problems in a laboratory model of manufacturing environment (see Figure 29). This is done in order to prove the robustness of the proposed prediction approach and to give evidence the usefulness and the applicability of the developed intelligent methods [5].

The sensors on the mobile robot can measure the light reflected by obstacles, those six sensors increment or decrement according to the position of the robot corresponds to distance of the sensor, the relationship can be found without any prior knowledge about the geometry of the mobile robot, by supervised learning techniques [146]. This learning is achieved by training a neural network with data collected manually. This method has two major advantages. The first one is simplicity. Indeed, no complicated model of the robot is needed, it has two position of the robot, and the principle remains the same [146]. The second advantage is robustness: information from sensor can be combined in the neural networks; the drawback of the method is that the performance of the neural network depends highly on how it is trained. There is no well defined method for this; it is more a matter of empirical rules and experimentation [146].

The experiments that deal with the trajectory tracking problem are conducted on a Khepera mobile robot. The original prediction method is implement and test on a real mobile robot in indoor environment for solving obstacle detection and trajectory tracking problems [5,141], and the next section give more details with solutions about those kinds of problems.

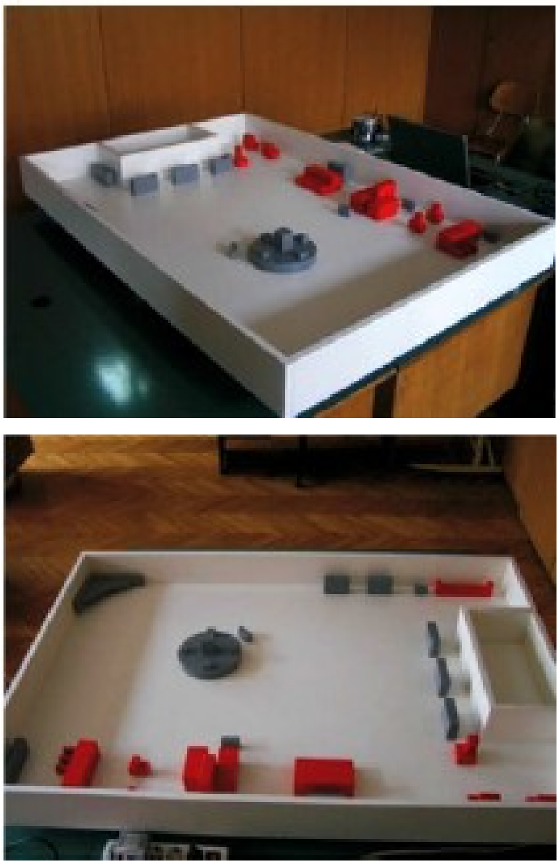


Figure 29 : Laboratory model of manufacturing environment

5.3.3.1 Khepera miniature mobile robot setup

In order to verify the proposed approach in a fair manner, several experiments in real time are conducted on a nonholonomic mobile robot in laboratory model of manufacturing environment. The robotic setup consists of *KheperaII* mobile robot and six integrated infrared sensors, the miniature robot is a mobile robot moving on two wheels. The data manipulation and robot control are carried out using Intel™ i5-2320 3GHz processor desktop computer with 4GBs RAM on Windows 7. Technical description of a mobile robot used in experiments is given in Table 3.

Table 3 : Robot configuration specifications

Specifications Robot Elements	Technical Description
Processor:	Motorola 68331, 25MHz
RAM:	512 Kbytes
Flash:	512 Kbytes
Motion:	2 DC servo motors with incremental encoders
Speed:	Max. 0.5 m/s; Min. 0.02 m/s
Power:	Power adapter or rechargeable NiMH batteries
Communication:	Standard serial port, up to 115kbps
Size:	Diameter: 0.07 m; Height: 0.03 m
Payload:	Approx. 250 g

Figure 30 shows the connection between robot and computer, which is converted by the connect module to line available on the robot. The line connects the robot with the interface module, and is also responsible for the robot power supply [146, 154].

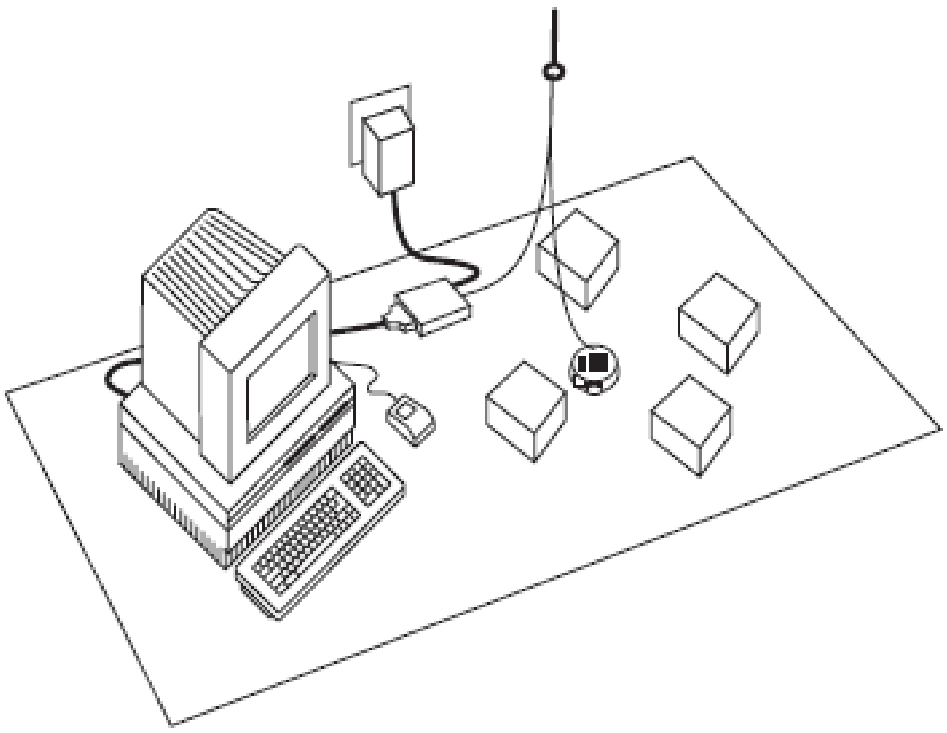


Figure 30 : Robot – computer connection configuration [154]

5.4 Mobile robot obstacle detection: problem and solution

As mentioned before, the mobile robot has a cylinder-like shape as shown in Figure 28. First experiment refers to an obstacle detection problem [155]. Six infrared sensors mounted directly on the front end of the robot are used to detect obstacle in two characteristic positions - on the left side and on the right side of the robot as shown in Figure 31. The NN training data are gathered by placing the platform in several positions for each case.

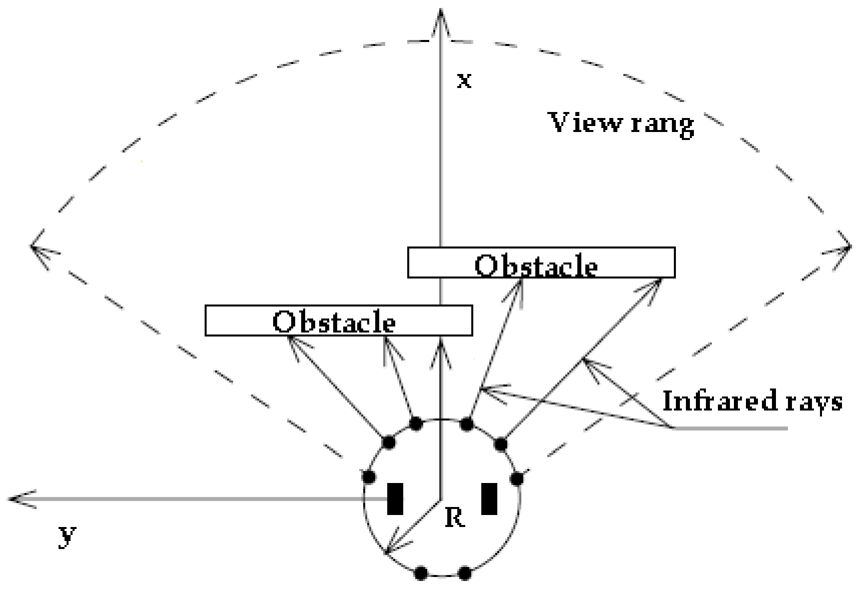


Figure 31 : Obstacle detection with IR sensors

In the Figure 32 one can notice that the robot is placed closer and further away from the object in different positions for both cases. This is done in order to show robustness of our approach regarding various combinations of obtained sensor measurements. Additionally, the failed data is gathered in two ways:

i. Manually, by blocking the chosen infrared sensor(s), and

ii. In control commands, by replacing correct values with the failed ones.

All these information are implemented in the training and testing set for NN divided in ratio 7: 3 (respectively), as in the previous case with the recorded forces and torques.

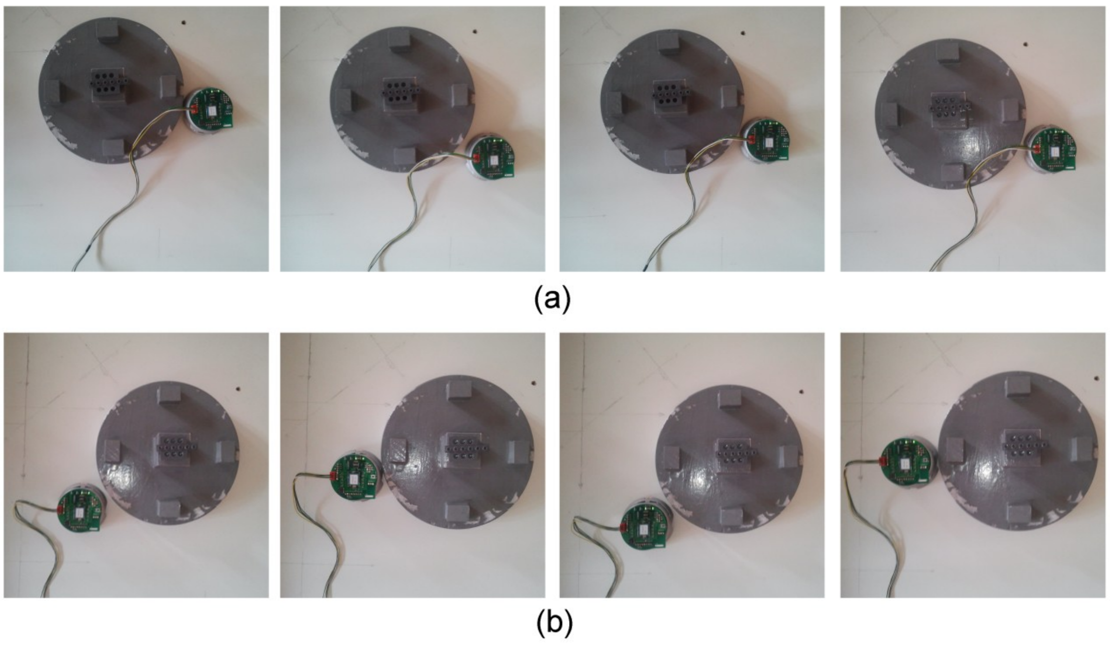


Figure 32 : Robot positions in the detection experiment: (a) obstacle on the left side, (b) obstacle on the right side [5]

5.4.1 Robot Obstacle detection problem based failure prediction

Since the earlier experiments showed that BR algorithm (Table 2) is the most suitable for this prediction problem, this one is used in all real world experiments. Also, we tested the overall most successful architecture. the network number 24 in table 1 (10-8-5-4). The activation functions are the same as earlier - sigmoid in hidden and linear in output layer. The NN input represents six infrared sensor measurements with values from 0 (obstacle is far) to 1020 (obstacle is near), and the output is a value 1 if the failure is recognized, and 0 otherwise. These values are scaled in accordance with equation (4.11). Similar to the previous cases, the failure prediction problem is formulated as classification problem [13, 17] using sensor information. An example of correct and incorrect sensor measurements for both cases are represented in Table 4, in which the „X" mark denotes failed sensor measurement. The first two measured values correspond to the sensor positions on the left side of the robot, while the last two values correspond to the sensors placed on the right side of the platform. In accordance to the aforementioned, one can conclude that the larger sensor values indicate that the object is closer to the robot (Table 4).

Table 4 : Example of correct and failed sensor information

	Correct infrared values	Failed (incorrect) infrared values
Obstacle on the left side	$IR = \begin{bmatrix}420 & 704 & 92 & 68 & 56 & 50\end{bmatrix}^T$	$IR = \begin{bmatrix}X & 554 & 52 & 64 & 56 & 95\end{bmatrix}^T$
Obstacle on the right side	$IR = \begin{bmatrix}72 & 68 & 48 & 100 & 984 & 1020\end{bmatrix}^T$	$IR = \begin{bmatrix}55 & X & 63 & 105 & X & 921\end{bmatrix}^T$

The incorrect sensor values are included in the training set in random manner, meaning that the number and the index location of the failed measurement are randomly generated [5,141]. We tested the BR NN algorithm and selected architecture several times for each detection problem with up to three failed values included in the input vector.

The result showed that in over 96 percent of all tested cases NN successfully recognized the failed sensor measurement. In addition to this, incorrect measurements are replaced with the expected values, so that the object location (left or right of the robot) is successfully recognized.

5.5 Mobile robot trajectory tracking: problem and solution

Usually, the main tasks that we consider for mobile robots in the absence of obstacles in an environment are trajectory tracking and point-to point motion [156]:

- Trajectory tracking is the case where a reference point of mobile robot should follow a trajectory in the Cartesian space starting from a given initial configuration.
- Point-to-point motion is the case where the mobile robot should reach a given goal configuration starting from a given initial configuration.

In this work we investigated NN behavior in the domain of failure prediction in the case of tracking two types of trajectories [5]. One can consider the case in which the robot wheels command unit is not working properly, i.e. the situation in which several control commands have unwanted values regarding tracking the particular trajectory. If the wheel command in every control iteration is not as expected (calculated), the mobile robot could make a significant error or even completely mist rack the desired trajectory. To prevent this from occurring, a safety-unit must be installed within the control system [5]. Using the developed NN-based approach, the mobile robot can detect the failed control value and replace it with the desired one, so as to enable successful accomplishment of trajectory tracking.

5.5.1 *M-shaped* trajectory tracking based failure prediction

Tracking of *M-shaped* trajectory is shown in Table 5 (1 unit corresponds to the wheel motion of 1/12 mm). It is noticeable that each control value must be as closer to the desired one, in order to successfully track this kind of trajectory. Obviously, in the case of employed incorrect wheel commands (second row in table 5), the tracking problem is not successfully solved.

Table 5 : Correct and failed wheel comment for M-shaped trajectory

Trajectory type	Correct wheel commands	Failed (incorrect) wheel commands
M-shaped trajectory	$Left\_wheel = \begin{bmatrix} -150 & 200 & 150 \\ 200 & -150 & 200 \\ 150 & 200 & -150 \end{bmatrix}$ $Right\_wheel = \begin{bmatrix} 150 & 200 & -150 \\ 200 & 150 & 200 \\ -150 & 200 & 150 \end{bmatrix}$	$Left\_wheel = \begin{bmatrix} X & 200 & 150 \\ 200 & -150 & 200 \\ 150 & X & -150 \end{bmatrix}$ $Right\_wheel = \begin{bmatrix} 150 & X & -150 \\ X & 150 & 200 \\ X & 200 & 150 \end{bmatrix}$

5.5.1.1 Tracking of M-shaped problem and solution

An implemented NN here is the same as in all experiments: BR algorithm with 10-8-4-2 architecture, and with sigmoid and linear activation functions. Input to the network is scaled right and left wheel commands, while the output represents successful or incorrect failure prediction [5, 141]. We tested trajectory several times, and the results of one test *M-shaped* is presented in Figure 33 (the red line denote the robot orientation in every control iteration). The experiments confirmed the usefulness of the proposed approach: in more than 99 percent of the cases, the network and the result show that the mobile robot is able to track *M*-shaped trajectory and that the developed intelligent approach successfully predicted the failures in control values.

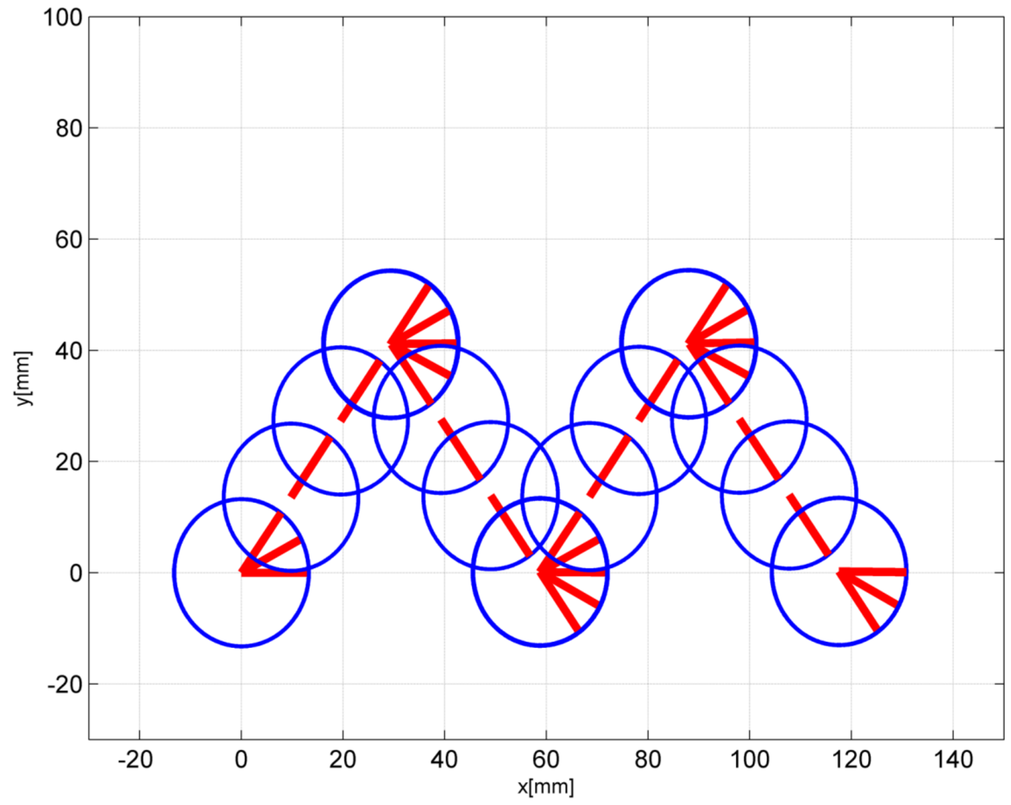


Figure 33 *M*-shaped trajectory tracking experiment [141]

In addition, in order to successfully track chosen trajectory, failed values are replaced with the desired information. The results shows that the mobile robot is able to track each trajectory, and that robot poses do not significantly differ from the wanted ones in every time instant. Screenshots from real world experiment are given in Figure 34 [141].

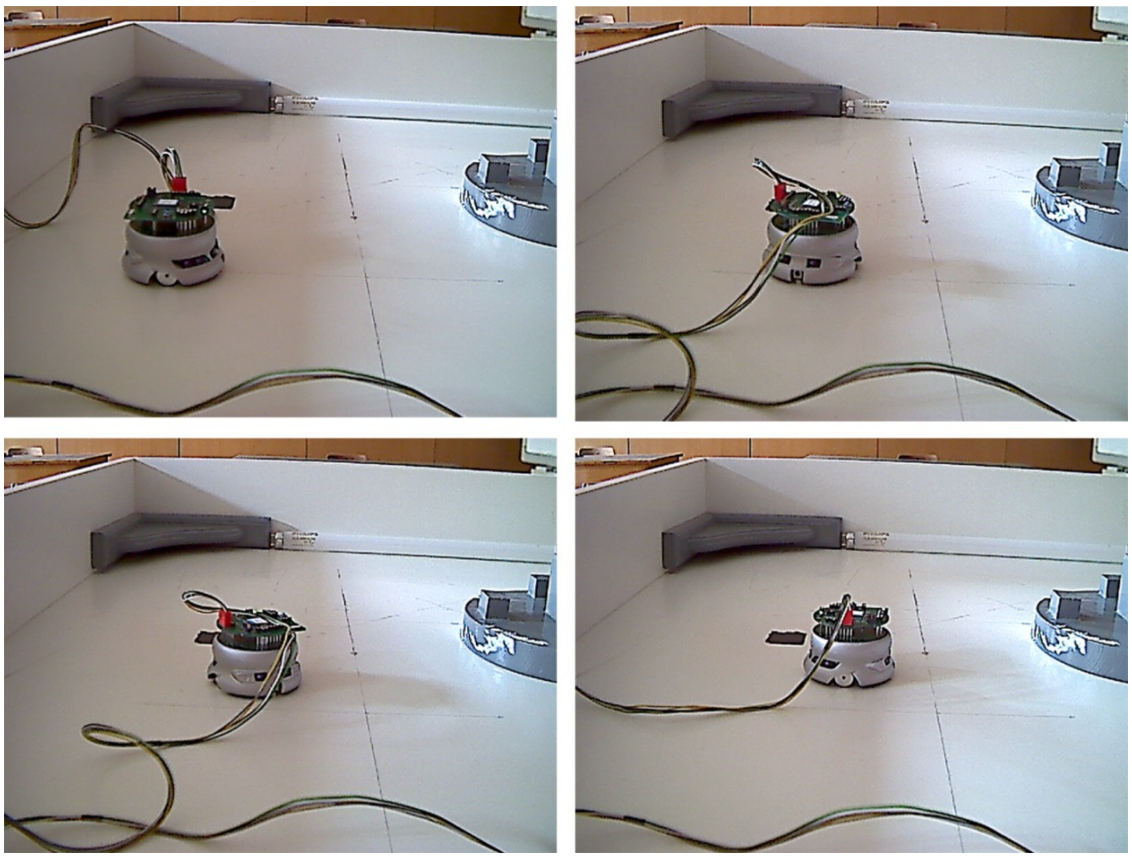


Figure 34 : Real world experiment in a manufacturing environment using KheperaII mobile robot [141]

5.5.2 *Labyrinth-type* trajectory tracking based failure prediction

In this case studied trajectory is more complicated *labyrinth-type* trajectory, with good and bad wheel signals given in Table 6. In this case, the tracking error is even more evident: the tracked trajectory is completely different unless every control value matches the desired information. As in the previous experiment, failed data is incorporated in random manner (number of failures and index locations) in the training set.

Table 6 : Correct and failed wheel comment of Labyrinth-type trajectory

Trajectory type	Correct wheel commands	Failed (incorrect) wheel commands
Labyrinth-type trajectory	$Left\_wheel = \begin{bmatrix} 200 & -124 & 200 \\ -124 & 150 & -124 \\ 150 & -124 & 100 \\ -124 & 100 & -124 \\ 20 & 20 & 10 \end{bmatrix}$ $Right\_wheel = \begin{bmatrix} 200 & 124 & 200 \\ 124 & 150 & 124 \\ 150 & 124 & 100 \\ 124 & 100 & 124 \\ 20 & 20 & 10 \end{bmatrix}$	$Left\_wheel = \begin{bmatrix} 200 & X & 200 \\ -124 & 150 & -124 \\ X & -124 & 100 \\ X & X & -124 \\ 20 & 20 & 10 \end{bmatrix}$ $Right\_wheel = \begin{bmatrix} 200 & X & 200 \\ 124 & 150 & 124 \\ 150 & 124 & X \\ 124 & 100 & 124 \\ X & 20 & 10 \end{bmatrix}$

5.5.2.1 Tracking of Labyrinth- type problem and solution

As stated, the other type of trajectory is *Labyrinth-type* trajectory which is more complicated task. In experiments, the implemented NN BR algorithm is the same as in the previous case. Trajectory tracking task is tested several times, and one of the test is shown in Figure 35. The experiment confirmed the robustness of the proposed approach: in more than 99 percent of the cases, the network successfully predicted the failure prediction [5]. In addition, in order to successfully track chosen trajectory, failed values are replaced with the desired information. The results shows that the mobile robot is able to track each trajectory, and that robot poses do not significantly differ from the wanted ones in every time instant [5].

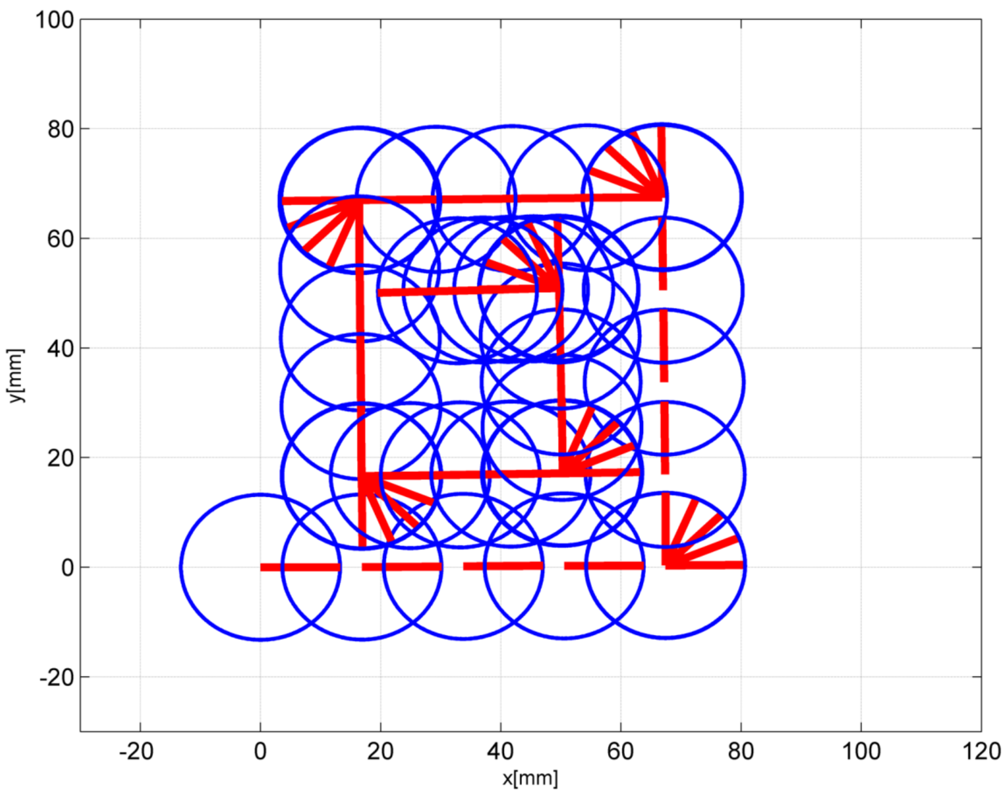


Figure 35 : Result of Labyrinth-type of trajectory tracking experiments

Finally, Figure 36 denotes robot poses in characteristic iterations during tracking of labyrinth-type trajectory (see also Figure 35). Starting from an arbitrarily pose in indoor environment, Figure 36 shows robot poses at the beginning and at the end of each straight-line motion (for example, Figure 36(a) and Figure 36(b), and also Figure 36(d) and Figure 36(e)), and the pose during the rotation (for example, Figure 36(c) and Figure 36(f)). It is obvious that, using the NN-based prediction method, mobile robot successfully track the complex trajectory. The minimal errors in the final robot pose evidence the usefulness and the robustness of the proposed approach described in this dissertation.

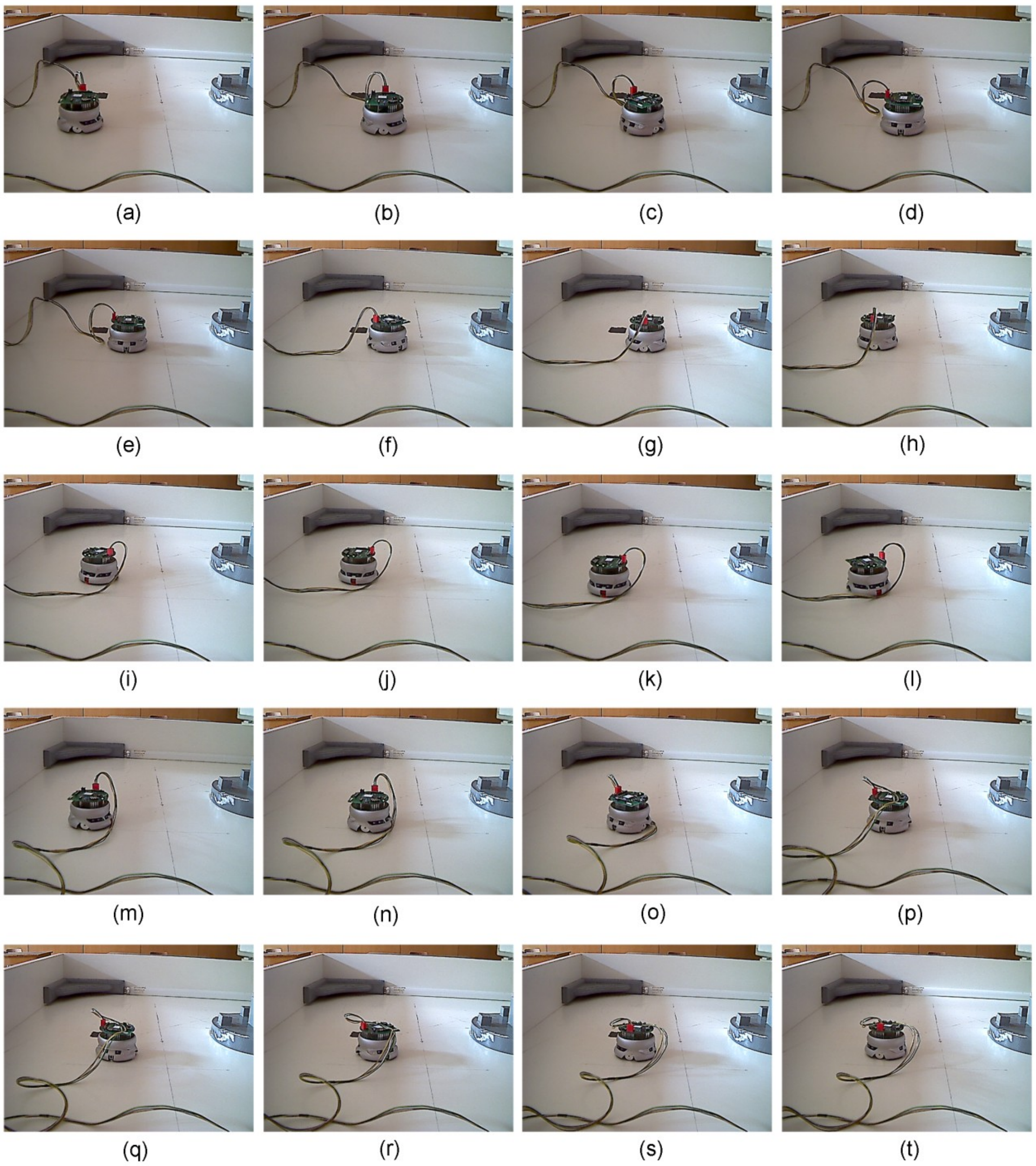


Figure 36 : Mobile robot tracking the labyrinth-type trajectory: (a)-(t) Robot poses in characteristic control iterations [5]

6. Experimental study: prediction of robot failures using neural networks

6.1 Experimental setup

In Chapter 5, two different problems are investigated under challenging conditions: object detection and trajectory tracking. First hypothesis tell us that it is possible to develop a machine learning base control subsystem for prediction of failures. Likewise, the other hypothesis referred to the situation in which it is possible to develop and implement NN prediction unit in robotic system which enables undisturbed execution of trajectory tracking and obstacle detection tasks. Three described experiments are used to validate these hypotheses; in two experiments a trajectory tracking task is studied, while one experiment relates to the obstacle detection task. These experiments focus on the core topic of this dissertation: verifying prediction ability of robots based on neural networks. Results show that NN with BR algorithm give the overall best results in aforementioned robotic tasks.

In this chapter, we employ Matlab, a high level matrix oriented programming language, and specially designed BPnet sofware in order to test several learning algorithms for analysis of failure data in four situations related to robot grasp position. The results should prove a remaining hypothesis: that is possible to develop a more precise prediction system based on Soft computing methods - neural networks. In Chapter 6 we focus on satisfying this hypothesis, using different learning algorithms and different NN architectures so as to find optimal solution for the robotic prediction unit.

6.1.1 Algorithm implementation (setup)

The verification of NN prediction performance is conducted using Intel® Core™2 Duo 2.1 GHz processor laptop computer with 2.96 GBs RAM on Windows XP platform. The Matlab 2009a (v. 7.8.0.347) is employed for algorithm implementation and testing. In order to find optimal NN, all architecture and algorithms are tested several times, the dataset is corrupted with erroneous values in random manner (these values are 0 to 3% in the entire set). Likewise, in order to test every NN structure in an appropriate way, we utilize 6 NN architectures separately for 1,2,3 and 4 layer networks (the total number of architectures is 24, as showed in Table 1. As stated before, the prediction in this work is treated as the classification problem, as used in many studies [3, 49]. Note that the acronyms listed in Table 2 are used to denote corresponding learning algorithm.

6.2 Experimental results

This dissertation delivers a novel approach using multilayer feedforward neural networks and Elman neural networks as a solution for the problem of failure prediction in robotic system, and also presents performance comparison of different NN learning algorithms and architectures. The tests are performed in Matlab, and they include all the main algorithms that proved to have best performance over classification, pattern recognition or nonlinear function approximation. We used criteria of MSE in Matlab and MAE in BPnet software to rank the performances of prediction algorithms. The adopted methodology and all results are given in the next sections within this chapter.

6.2.1 Prediction Performance Using LM and BR algorithm

Results of MSE on test data (30% of the dataset) for LM and BR algorithm are depicted in Figure 37(a) and Figure 37 (b), respectively. The NN architectures in the figures represent network number given in Table 7 and correspond to MSE. It is obvious that the MSE for LM has the decreasing trend when number of neurons and layers increases. In other words, the larger number of neurons and layers has positive influence on the training process. The best MSE result for LM algorithm is recorded for network number 23 in Table 7, and is 0.0023. In the case of BR algorithm the similar conclusion can be obtained. Overall, the NNs with 4 hidden layers show the best performances. Particularly, the network number 22 has the smallest test MSE of 0.0036.

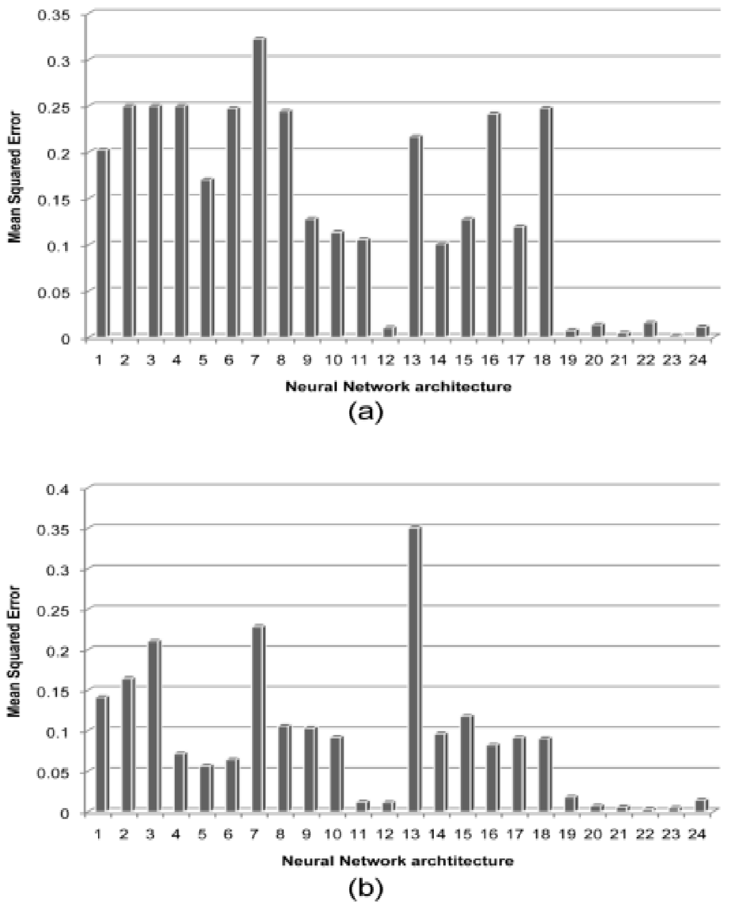


Figure 37 : NN testing results: (a) LM algorithm, (b) BR algorithm

Table 7 : MSE of LM & BR algorithms for NN architectures

No	Algorithm / Neural Network	LM	BR
1	[1]	0.2026	0.1416
2	[2]	0.2500	0.1646
3	[3]	0.2500	0.2108
4	[5]	0.2500	0.0717
5	[8]	0.1707	0.0567
6	[10]	0.2475	0.0646
7	[1 1]	0.3227	0.2286
8	[2 2]	0.2446	0.1060
9	[3 2]	0.1282	0.1035
10	[5 2]	0.1137	0.0921
11	[8 4]	0.1059	0.0124
12	[10 4]	0.0104	0.0118
13	[2 2 2]	0.2172	0.3509
14	[3 2 2]	0.1011	0.0963
15	[4 3 2]	0.1281	0.1182
16	[5 3 2]	0.2416	0.0825
17	[8 3 2]	0.1199	0.0917
18	[8 4 2]	0.2478	0.0906
19	[3 3 3 3]	0.0077	0.0185
20	[4 3 3 3]	0.0138	0.0075
21	[5 4 3 3]	0.0048	0.0062
22	[8 5 4 3]	0.0161	0.0036
23	[10 8 4 3]	0.0023	0.0050
24	[10 8 5 4]	0.0114	0.0146

The regression plot is used to validate the network performance. This regression plots display the network outputs with respect to targets for training, validation, and test sets. For a perfect fit, the data should fall along a 45 degree line, where the network outputs are equal to the targets and value of R (correlation coefficient) is equal to 1 indicates perfect correlation. The validation and regression performance of training NN with LM algorithm and with architecture [10 8 4 3] reported smallest. As it can be seen in Figure 38, the mean squared error of the validation and test start to decrease after epoch 15 and at epoch 38 the validation returns less MSE. The regression plots in Figure 39 show the results of the network outputs for the training patterns compared to the actual targets.

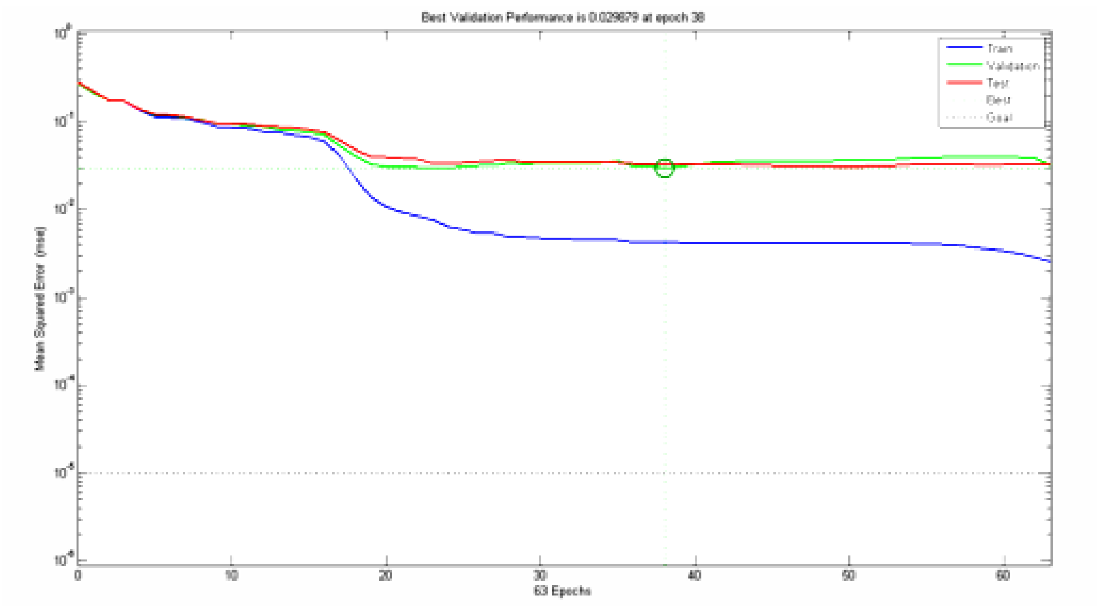


Figure 38 : Mean-square error performance for trainlm NN [10 8 4 3]

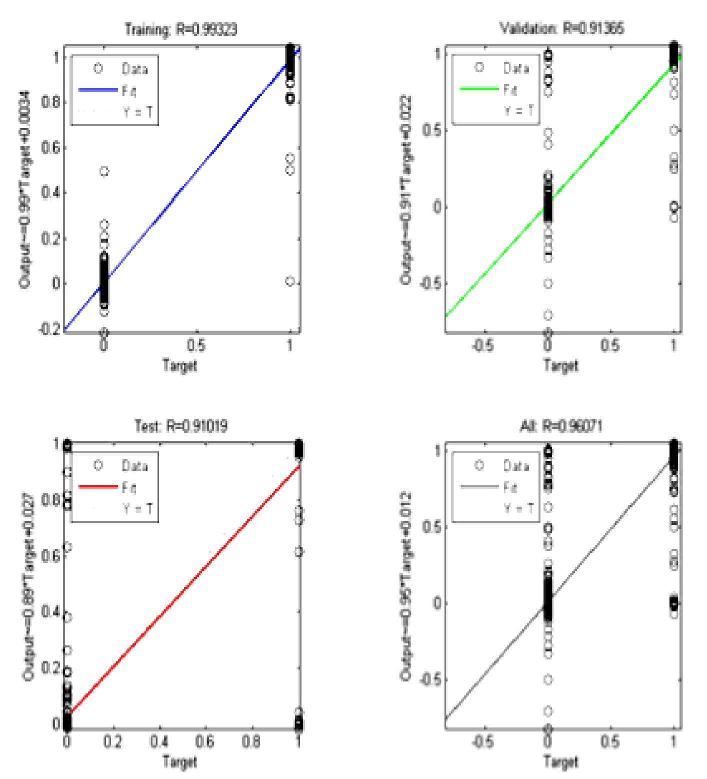


Figure 39 : Regression of the outputs vs. targets for the network [10 8 4 3]

The regression and performance plots of training results for BR algorithm and NN architecture [8 5 4 3] are shown in Figure 40 and Figure 41. Based on the performance plots we can see that the networks have obtained the best validation performance for training NN [8 5 4 3] at epochs 27. The plots show good results with 0.96473 values of all R.

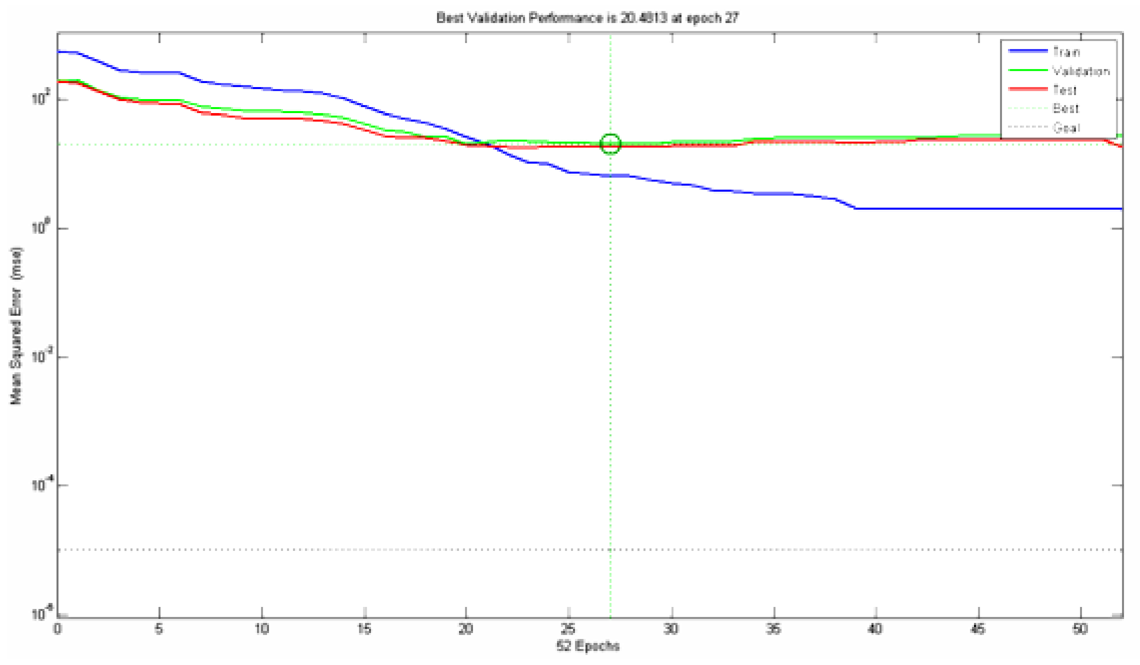


Figure 40 : Mean-square error performance for trainbr NN [8 5 4 3]

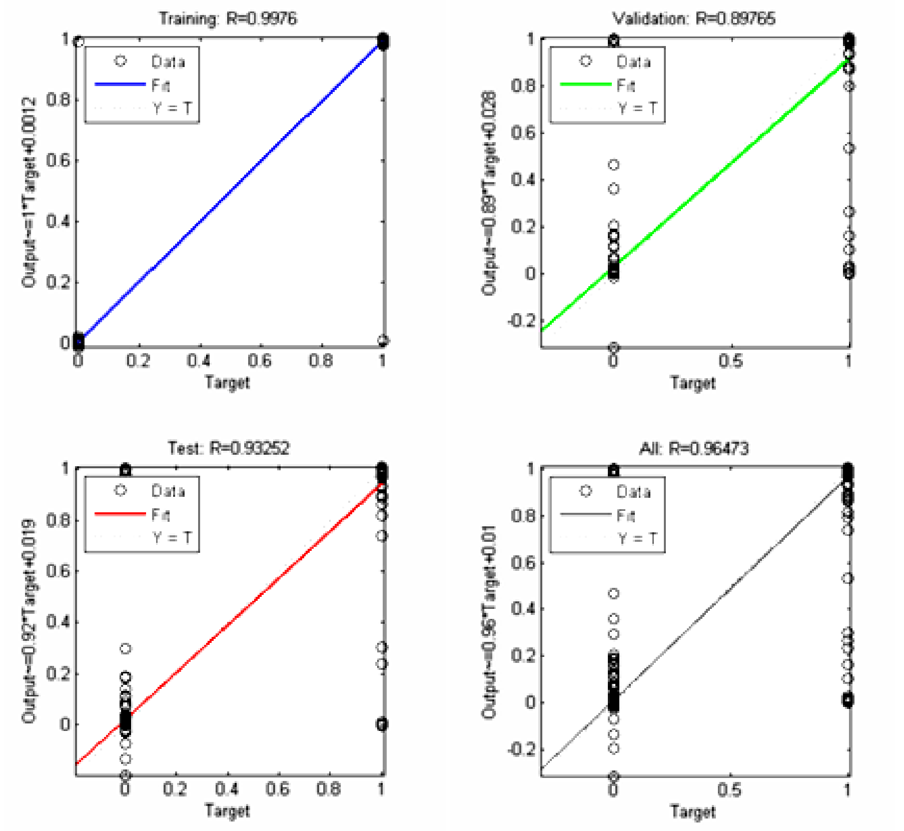
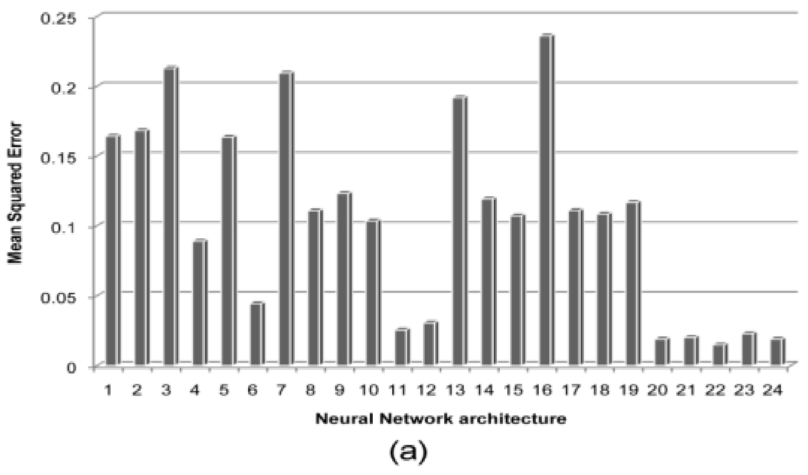


Figure 41 : Regression of the outputs vs. targets for the network [8 5 4 3]

6.2.2 Prediction performance using RP and SCG NN algorithms

Figure 42 and Table 8 indicate MSE results for RP and SCG algorithms. As in the previous case, the best results are for NNs with 4 hidden layers: for RP algorithm the smallest MSE is 0.0151 (for NN number 22, figure 42(a)), while the SCG has the best MSE of 0.005 (also NN 22, as shown in figure 42(b)). Likewise, the two NNs that give good performance are 8-4, and 10-4. Similarly to the result of LM and BR algorithm, the networks under numbers 11 (8-4 NN) and 12 (10-4 NN) show smallest errors among 2 layer networks. In other words, these NNs also show results that are promising for prediction purposes.



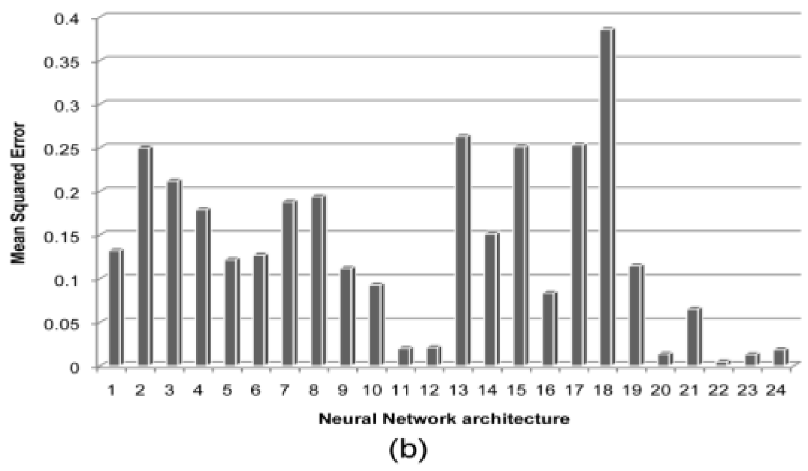


Figure 42 : NN testing results: (a) RP algorithm, (b) SCG algorithm

Table 8 : MSE of RP & SCG algorithms for NN architectures

No	Algorithm / Neural Network	RP	SCG
1	[1]	0.1644	0.1324
2	[2]	0.1683	0.2500
3	[3]	0.2131	0.2121
4	[5]	0.0891	0.1791
5	[8]	0.1637	0.1221
6	[10]	0.0445	0.1272
7	[1 1]	0.2096	0.1883
8	[2 2]	0.111	0.1942
9	[3 2]	0.1233	0.1121
10	[5 2]	0.1037	0.0930
11	[8 4]	0.0258	0.0202
12	[10 4]	0.0309	0.0214
13	[2 2 2]	0.1920	0.2632
14	[3 2 2]	0.1194	0.1513
15	[4 3 2]	0.1074	0.2512
16	[5 3 2]	0.2361	0.0838
17	[8 3 2]	0.1112	0.2532
18	[8 4 2]	0.1086	0.3858
19	[3 3 3 3]	0.1169	0.1148
20	[4 3 3 3]	0.0191	0.0138
21	[5 4 3 3]	0.0204	0.0654
22	[8 5 4 3]	0.0151	0.0050
23	[10 8 4 3]	0.0232	0.0134
24	[10 8 5 4]	0.0191	0.0186

The validation and regression performance of training NN with RP algorithm, which gives the minimum MSE for training NN with architecture [8 5 4 3] are given in Figure 43 and Figure 44. The MSE of the best validation performance is 0.0026096. The regression plots in Figure 44 show the results of the networks outputs for the training patterns compared to the targets.

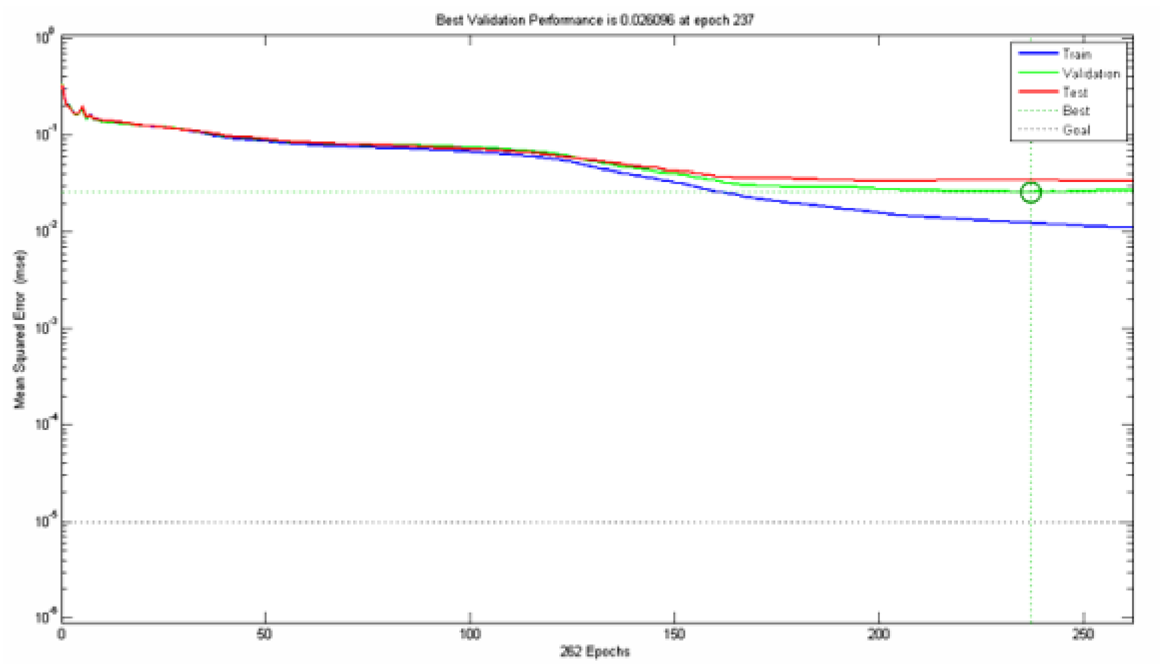


Figure 43 : Mean-square error performance for trainrp ANN [8 5 4 3]

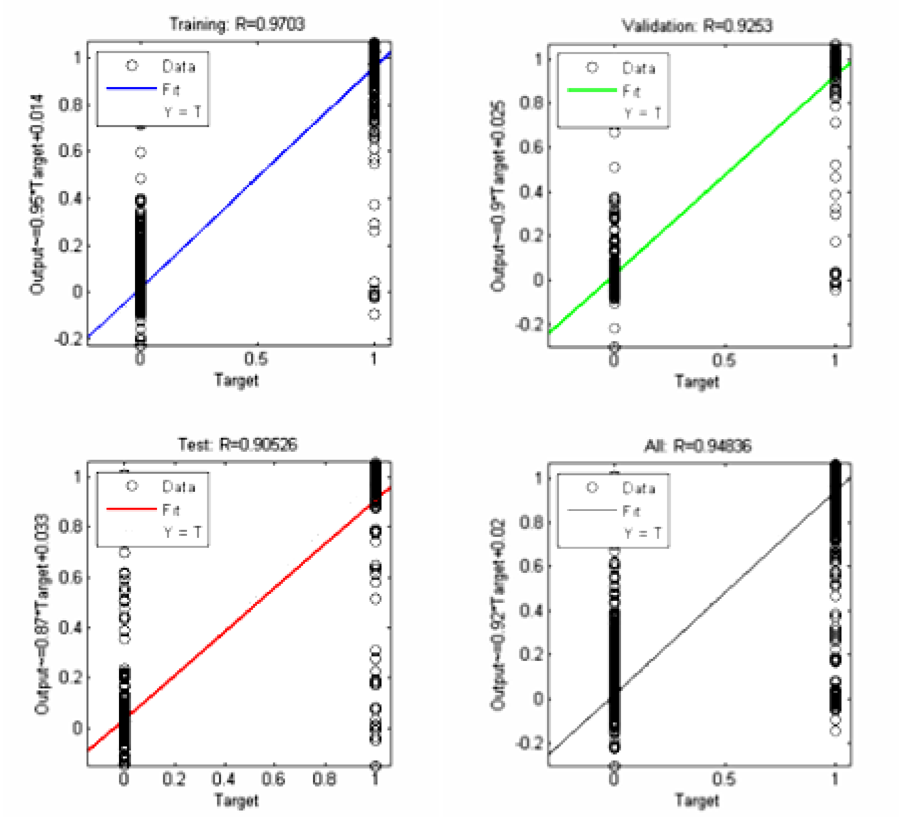


Figure 44 : Regression of the outputs vs. targets for the network [8 5 4 3]

For the SCG, Figure 45 shows the training performance plot of the neural network [8 5 4 3]. It can be seen that the network did not achieve the desired Mean Square Error (MSE) goal by the end of the training process. Same as in previous case, Figure 46 shows the regression plots of the networks outputs for the training patterns compared to the actual targets.

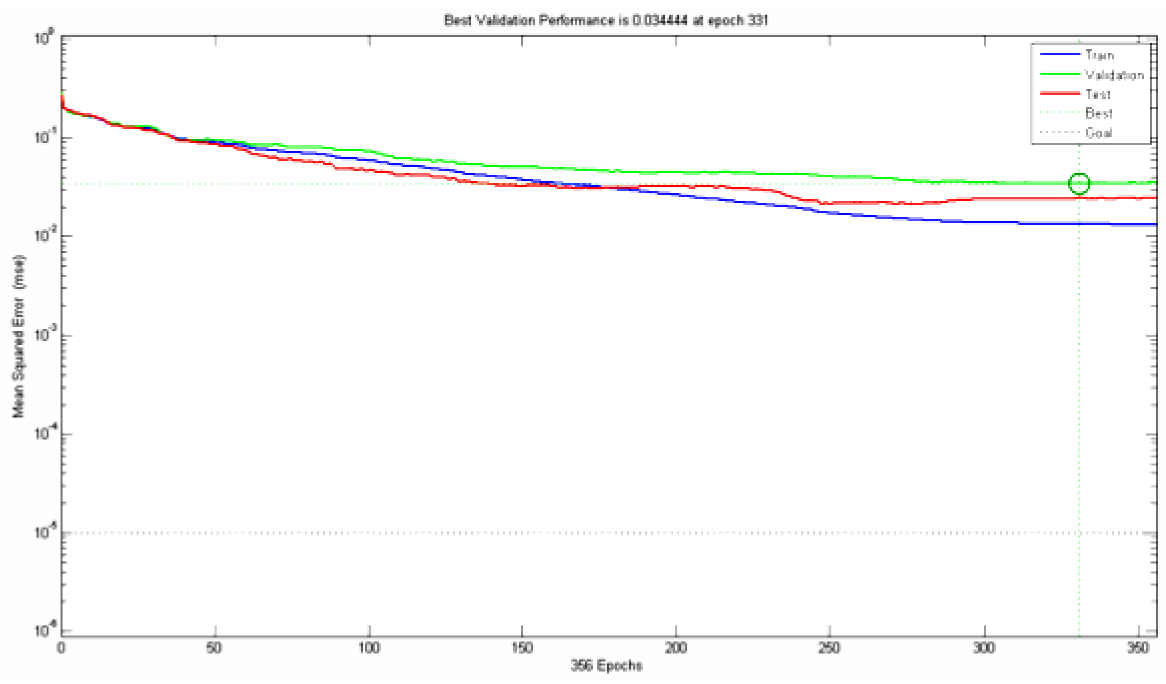


Figure 45 : Mean-square error performance for trainscg ANN [8 5 4 3]

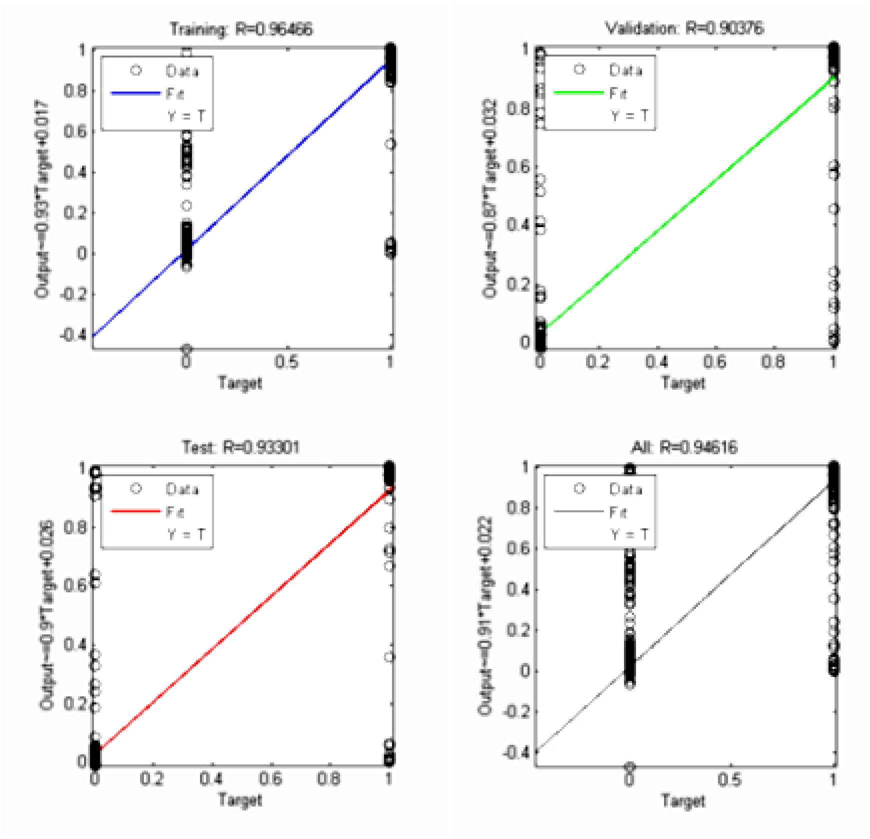


Figure 46 : Regression of the outputs vs. targets for NN [8 5 4 3]

6.2.3 Prediction performance using BFG and GDX algorithms

The MSE test results for BFG and GDX algorithms are presented in Figure 47(a) and Figure 47(b), respectively. In BFG case, the optimal network is the one with the 4 hidden layers (NN number 24 in Table 9), while the smallest MSE for GDX algorithm is NN with 2 hidden layers (NN number 11 in Table 9). Unlike previous cases, the NNs with the BFG algorithm do not exhibit overall best results with 4 hidden layers. In Figure 47(b), the NNs with 2 hidden layers show best GDX algorithm performance. However, the MSE results presented in this section only indicate the optimal NN outcome, since the prediction is determined based on the largest value in the network output.

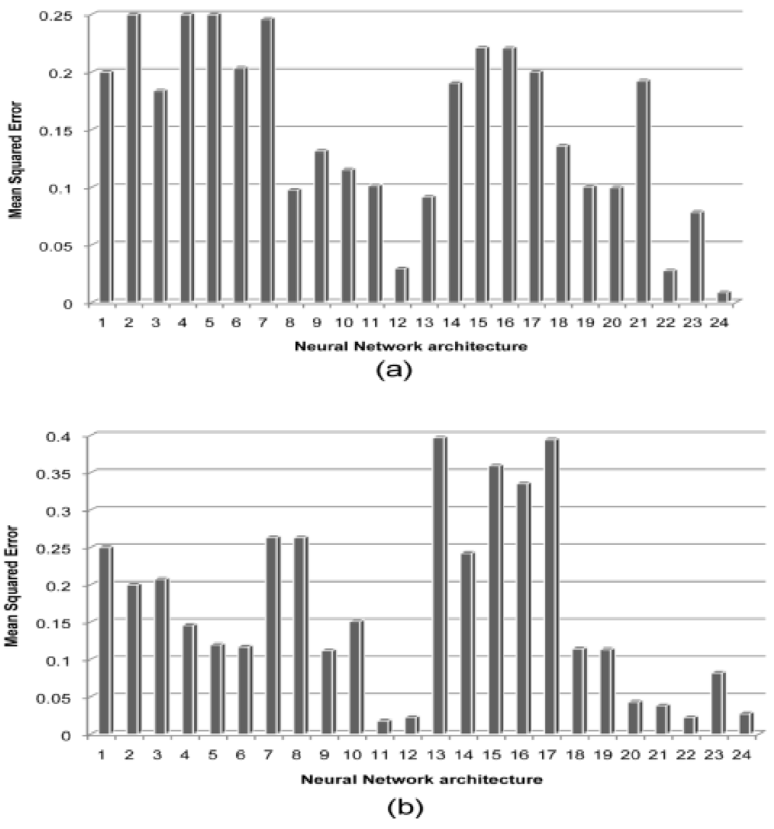


Figure 47 : NN testing results: (a) BFG algorithm, (b) GDX algorithm

Figures 48 and Figure 49 show the validation and regression performance of training NN using BFG algorithm and architecture [10 8 5 4], which gave the smallest results of MSE. In Table 9, the best validation performance is 0.02073 at epoch 217 with all R values equal to 0.53803.

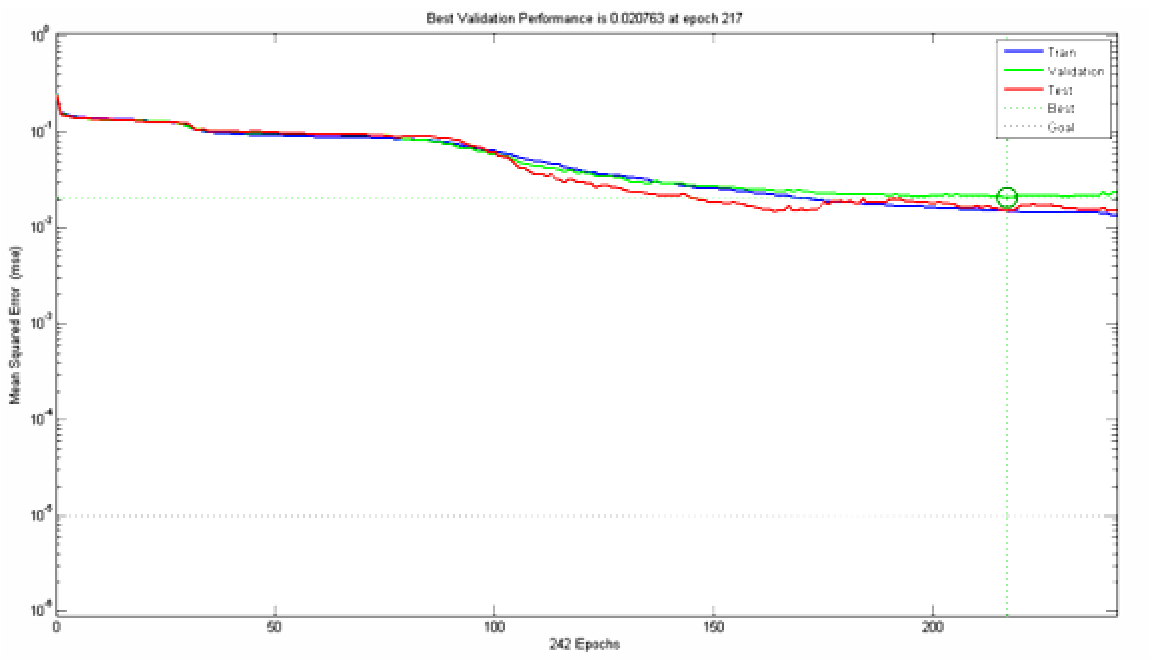


Figure 48 : Mean-square error performance for trainbfg NN [10 8 5 4]

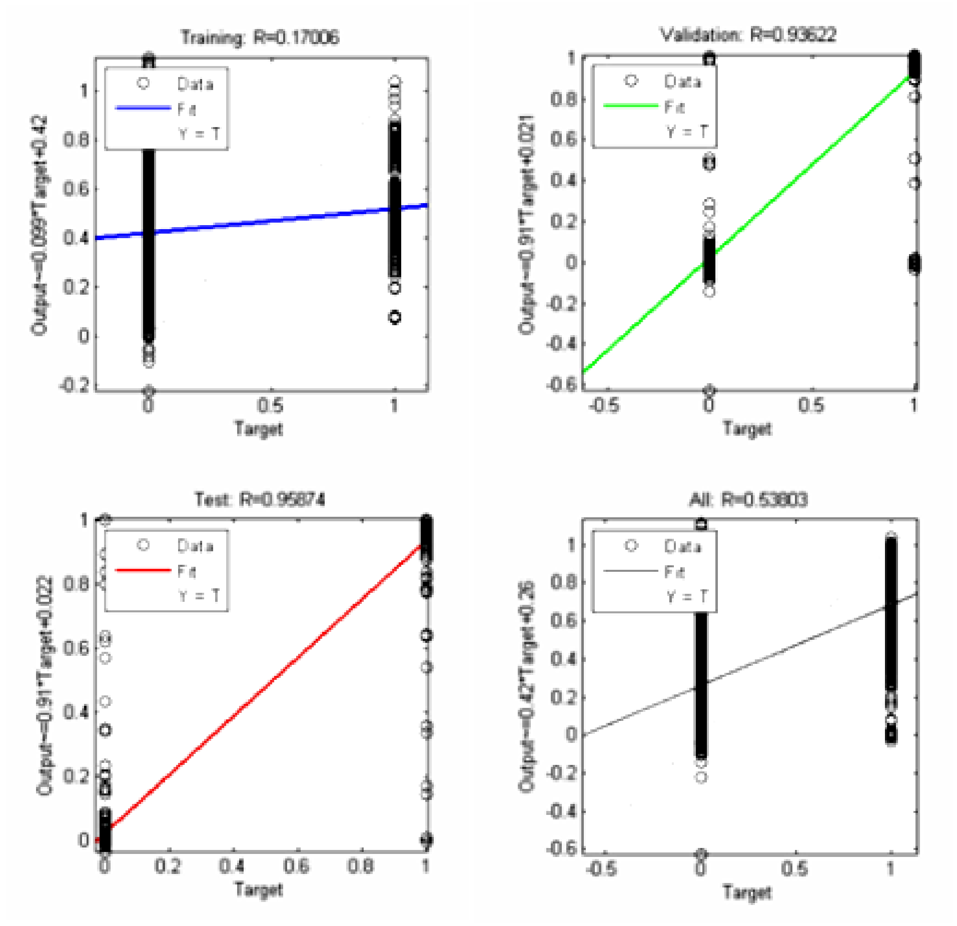


Figure 49 : Regression of the outputs vs. targets for the network [10 8 5 4]

Table 9 : MSE of GDX & BFG algorithms for NN architectures

No	Algorithm / Neural Network	GDX	BFG
1	[1]	0.2500	0.200
2	[2]	0.2000	0.2500
3	[3]	0.2076	0.1839
4	[5]	0.1460	0.2500
5	[8]	0.1197	0.2500
6	[10]	0.1172	0.2037
7	[1 1]	0.2634	0.2462
8	[2 2]	0.2633	0.0979
9	[3 2]	0.1120	0.1315
10	[5 2]	0.1511	0.1154
11	[8 4]	0.0178	0.1014
12	[10 4]	0.0226	0.0297
13	[2 2 2]	0.3967	0.0917
14	[3 2 2]	0.2425	0.1906
15	[4 3 2]	0.3594	0.2212
16	[5 3 2]	0.3354	0.2208
17	[8 3 2]	0.3944	0.2002
18	[8 4 2]	0.1145	0.1360
19	[3 3 3 3]	0.1135	0.1007
20	[4 3 3 3]	0.0433	0.0998
21	[5 4 3 3]	0.0384	0.1924
22	[8 5 4 3]	0.0225	0.0276
23	[10 8 4 3]	0.0822	0.0785
24	[10 8 5 4]	0.0276	0.0089

In Figure 50 the plot shows the mean squared error during training of the network with GDX algorithm, starting at a large value and decreasing to a smaller value. In other words, it shows that the learning process is correct. No significant change in learning has occurred during the process, and the iteration 1000 gives the best performance. In Figure 51, regression number is shown. The R-value between the outputs and targets is a measure of how well the variation in the output is explained by the targets. In this case, total response of R is more than 0.939, and is not significantly different from 1. Therefore, it can be concluded that the NN performance is overall satisfying.

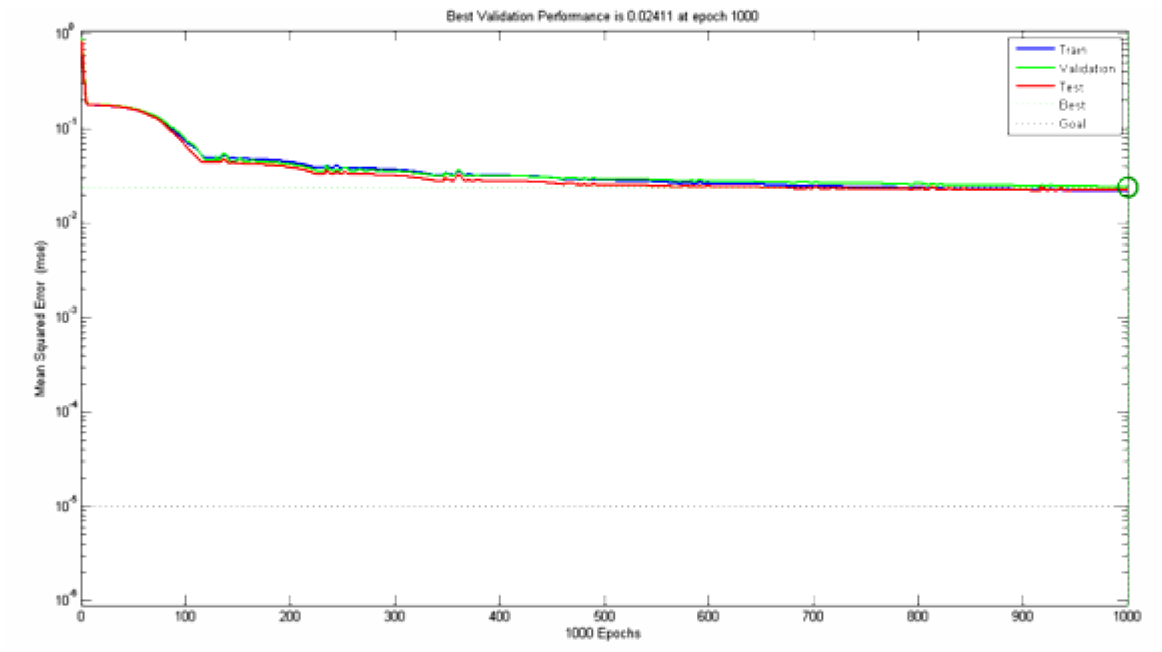


Figure 50 : Mean-square error performance for traingdx NN [1 1]

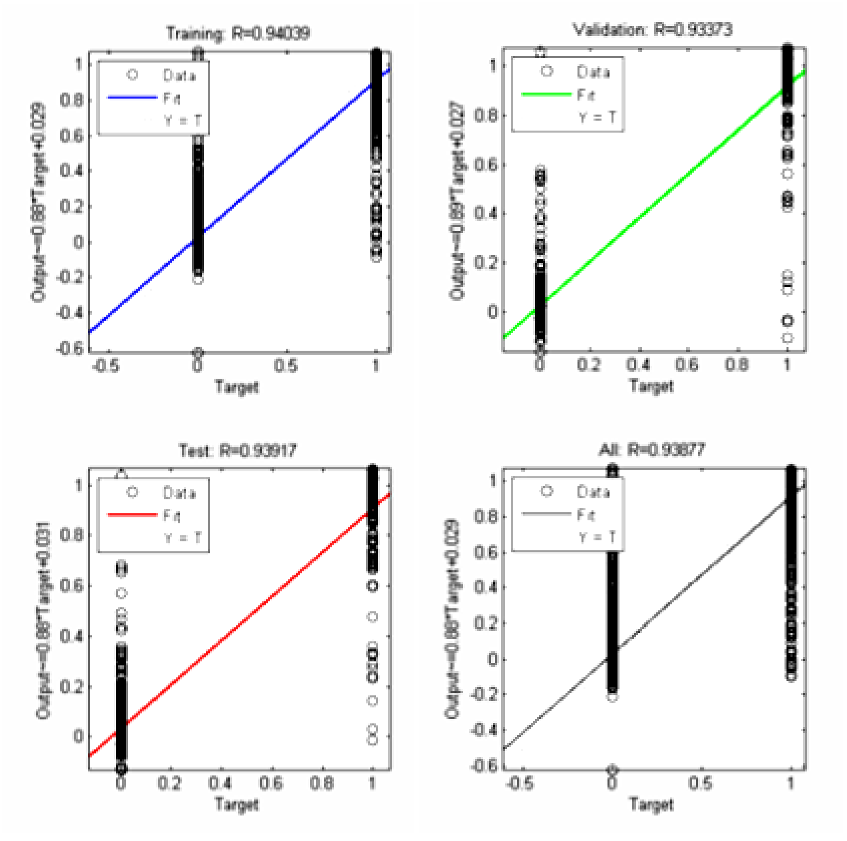


Figure 51 : Regression of the outputs vs. targets for the network [1 1]

6.2.4 Overall performance of NNs in prediction task

After the testing phase, the prediction rate was calculated for each learning class individually. Additionally, the average rate for all algorithms and all NN architectures is determined. Since the NN inputs are randomly generated from a predefined base, the prediction performance over the entire testing dataset is presented. The results obtained in Matlab environment are shown in Table 10.

Looking at the Table 10 one can notice several important conclusions (best results are given in bold font):

Firstly, some results reported in Figures 37 and Figure 47 do not correspond to the results given in Table 10. For example, the second largest MSE for BR algorithm (Figure 37(b)) is found for network number 7 (architecture [1 1] in Table 7), while the corresponding prediction rate in Table 10 is 56.0606 which is fourth lowest result for that algorithm. This is because we evaluate prediction based on the largest network output, i.e. we assume that the NN predicted correctly if the node that gives the largest output corresponds to the neuron with the target value 1. Nevertheless, this is not significant, since the NNs with the smallest MSE show the best prediction rate for each learning algorithm (Table 10).

Secondly, despite the aforementioned, in most cases obtained MSE corresponds to the algorithm prediction rate. The evident MSE increasing or decreasing trend reported is also found in the prediction table.

Thirdly, the results confirmed that the NN can successfully predict robot execution failures and showed that artificial neural networks are a powerful tool for failure prediction rates [157]. The highest prediction rate of 95.4545 was found for the network number 24 [10 8 5 4], which is better than the results obtained with the base-level and meta-level classifier reported in [43]. In spite of added erroneous data, the NN BR method outperforms the Naïve Bayes, Decision Trees and Support Vector Machine based algorithms [43]. These results evidence the applicability and the effectiveness of the NN in the case of failure prediction related task.

Table 10 : NN prediction rate in Matlab (in percentage)

NN architecture	LM	BR	RP	SCG	BFG	GDX	Average rate (architecture)
1	42.9293	46.2121	44.4444	51.2626	19.9495	23.2323	38.005
2	27.5253	40.1515	50	37.6263	24.2424	52.7677	38.7189
3	35.6061	37.8788	16.1616	45.9596	38.1313	16.6667	31.734
5	23.9899	72.7273	71.4646	37.8788	24.4949	52.5253	47.1801
8	42.1717	78.5354	37.1212	56.5657	21.2121	63.1313	49.7896
10	23.7374	72.2222	83.0808	63.1313	31.5657	57.0707	55.1347
1-1	51.2626	56.0606	53.2828	53.7879	52.2727	57.5758	54.0404
2-2	57.5758	69.4444	71.2121	54.0404	73.7374	60.3535	64.3939
3-2	67.9293	76.5152	73.7374	66.9192	70.2020	87.1212	73.7374
5-2	70.7071	73.4848	72.9798	69.4444	69.4444	69.4444	70.9175
8-4	92.4242	92.9293	92.6768	93.1818	72.2222	92.1717	89.2677
10-4	91.9192	91.9192	91.4141	93.1818	91.9192	94.1919	92.4242
2-2-2	55.3030	35.8586	56.0606	55.3030	73.9899	36.1111	52.1044
3-2-2	89.8990	74.7475	69.1919	68.9394	53.7879	55.3030	68.6448
4-3-2	73.2323	71.7172	73.2323	54.7980	58.8384	37.1212	61.4899
5-3-2	56.0606	75	56.3131	77.0202	56.0606	33.8384	59.0488
8-3-2	71.7172	79.0404	70.2020	56.8182	49.2424	33.8384	60.1431
8-4-2	56.0606	77.2727	76.7677	42.4242	71.4646	70.2020	65.6986
3-3-3-3	89.6465	89.8990	69.1919	72.2222	67.6768	69.4444	76.3468
4-3-3-3	90.6566	92.1717	91.1616	91.1616	71.4646	85.6061	87.037

5-4-3-3	94.6970	91.1616	91.6667	78.2828	53.7879	86.6162	82.702
8-5-4-3	93.6869	94.4444	92.6768	92.6768	93.4343	89.3939	92.7189
10-8-4-3	94.4444	94.4444	91.1616	91.6667	72.9798	75.2525	86.6582
10-8-5-4	94.1919	95.4545	93.6869	92.1717	93.9394	89.3939	93.1397
Average rate (algorithm)	68.9509	74.1372	70.3704	66.5194	58.5858	62.0156	/

6.2.6 Feedforward & Elman neural network comparison results

The testing results in terms of Mean Squared Error (MSE) for Elman NN (ELM) and feedforward NN with Bayesian Regularization algorithm in are given in Figure 37(b). In Figure 52 and Figure 53, the Elman NN with smallest MSE was reported for [10] architecture (is equal 0.0539), and the validation and training plots are shown in Figure 53. The BR algorithm based NN show overall better results than ELM. Smallest MSE was reported for [10-8-4-3] architecture (see Table 7).

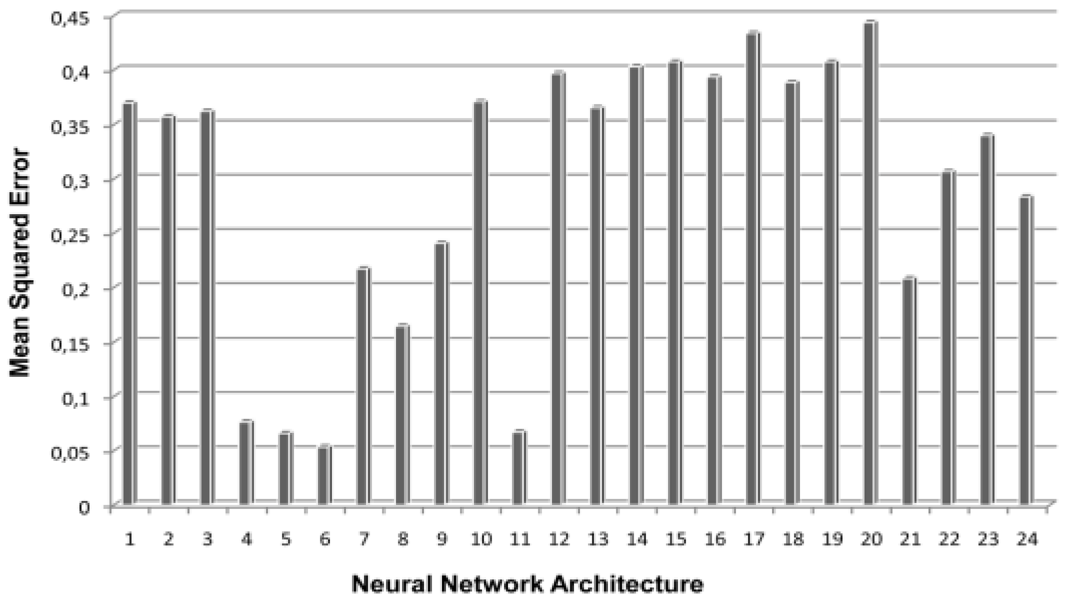


Figure 52 : Results of Elman NN [10] testing

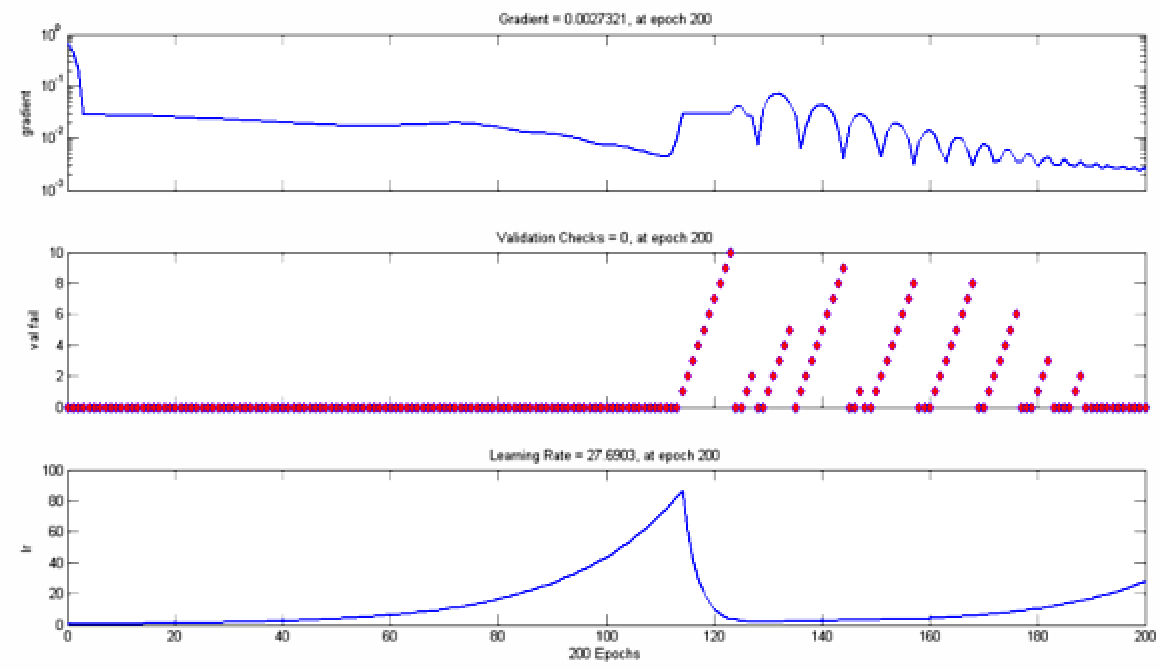
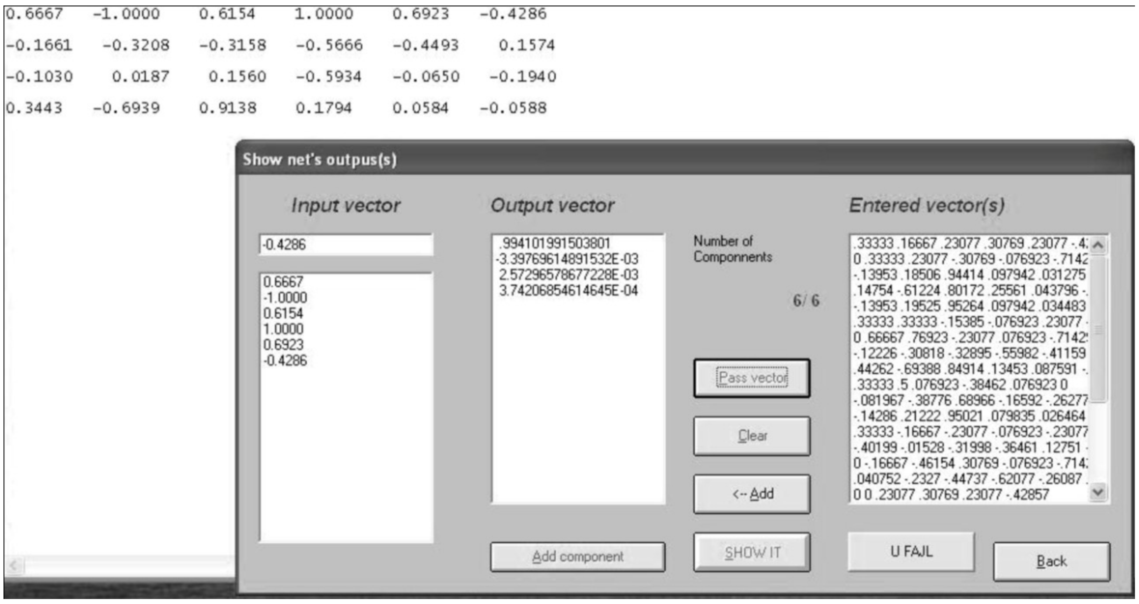


Figure 53 : Elman training states for NN $[10]_1$ of the smallest MSE

6.3 Experimental results in BPnet software

Results of BPnet engagement are given in Figure 55 and in Figure 56. The same network architectures listed in Table 11 are considered in this case. MAE results for each architecture are shown in figure 55. Note that the error is similar for most of the tested NN. The best result is obtained for the same NN architecture as in Matlab software – MAE is given in table for 10-8-5-4 network is 0.009961966. As for the LM case, the worst result shows NN with architecture 1-1 (Table 10). In order to validate the prediction ability, network with smallest MAE is tested next. Figure 56 presents testing input and output values. It can be observed that the trained NN successfully predicts the "normal" case from the scaled input forces and torques.

In other words, the generated output vector in Figures 33 and 35 in chapter 4 corresponds to the randomly selected sensor measurements from the testing dataset. Generally, the BP algorithm performs well – overall, the prediction rate for all networks is 70.8333 percent. These results indicate that the BP, as well as other tested algorithms, can successfully be applied for robot execution failure prediction. Furthermore, as in the previous cases, the NN shows robustness regarding implemented erroneous values of forces and torques in the training/testing dataset.



```
0.6667   -1.0000    0.6154    1.0000    0.6923   -0.4286
-0.1661   -0.3208   -0.3158   -0.5666   -0.4493    0.1574
-0.1030    0.0187    0.1560   -0.5934   -0.0650   -0.1940
0.3443   -0.6939    0.9138    0.1794    0.0584   -0.0588
```

Input vector: -0.4286

```
0.6667
-1.0000
0.6154
1.0000
0.6923
-0.4286
```

Output vector:

```
.994101991503801
-3.39769614891532E-03
2.57296578677228E-03
3.74206854614645E-04
```

Number of Componnents 6/ 6

Entered vector(s):

```
.33333 .16667 .23077 .30769 .23077 -.4
0 .33333 .23077 -.30769 -.076923 -.7142
-.13953 .18506 .94414 .097942 .031275
.14754 -.61224 .80172 .25561 .043796 -
-.13953 .19525 .95264 .097942 .034483
.33333 .33333 -.15385 -.076923 .23077 -
0 .66667 .76923 -.23077 .076923 -.7142
-.12226 -.30818 -.32895 -.55982 -.41159
.44262 -.69388 .84914 .13453 .087591 -
.33333 .5 .076923 -.38462 .076923 0
-.081967 -.38776 .68966 -.16592 -.26277
-.14286 .21222 .95021 .079835 .026464
.33333 -.16667 -.23077 -.076923 -.23077
-.40199 -.01528 -.31998 -.36461 .12751 -
0 -.16667 -.46154 .30769 -.076923 -.714
.040752 -.2327 -.44737 -.62077 -.26087
0 0 .23077 .30769 .23077 -.42857
```

Figure 54 : NN testing in BPnet software

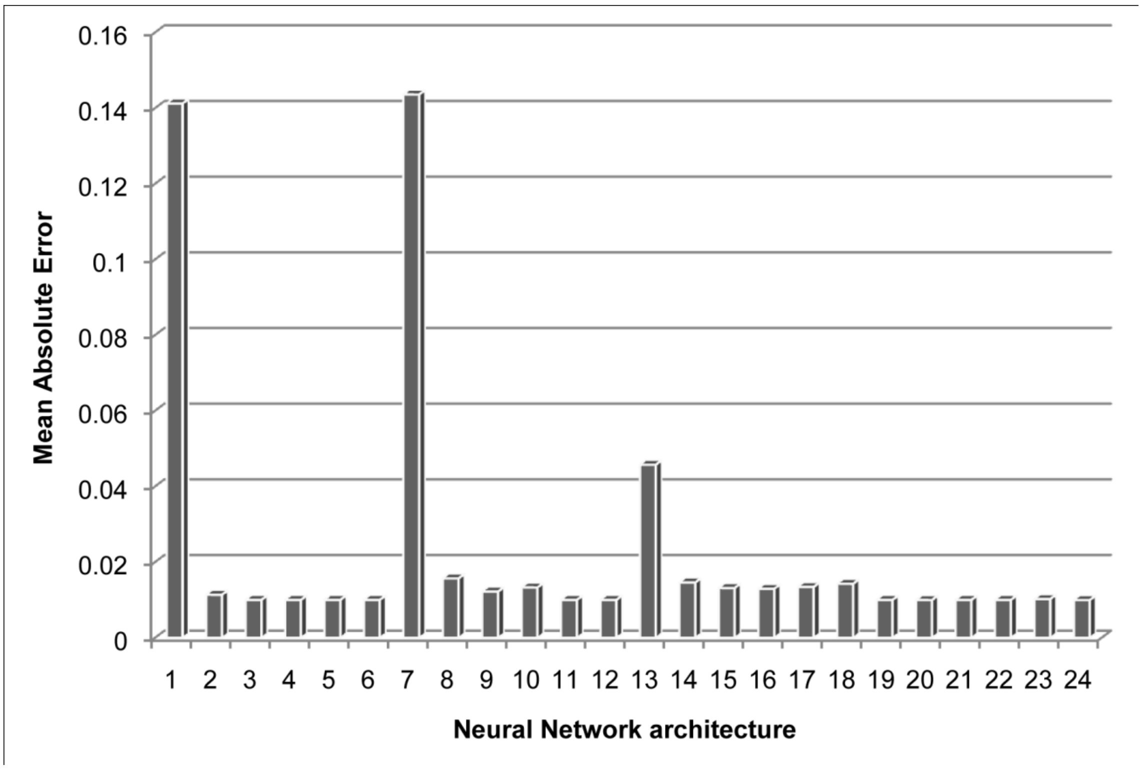


Figure 55 : BPnet software training results

Table 11: MAE of Gradient Descent Backpropagation algorithm for NN architectures in BPnet software

No	Algorithm / Neural Network	Gradient Descent Backpropagation
1	[1]	0.141225721
2	[2]	0.011362442
3	[3]	0.0099993
4	[5]	0.009999755
5	[8]	0.009999557
6	[10]	0.009999956
7	[1 1]	0.14359879
8	[2 2]	0.015681501
9	[3 2]	0.012203045
10	[5 2]	0.013296583
11	[8 4]	0.009999593
12	[10 4]	0.009999142
13	[2 2 2]	0.045724337
14	[3 2 2]	0.014606998
15	[4 3 2]	0.013146853
16	[5 3 2]	0.012939744
17	[8 3 2]	0.01342879
18	[8 4 2]	0.014260617
19	[3 3 3 3]	0.009999993
20	[4 3 3 3]	0.009972804
21	[5 4 3 3]	0.00999136
22	[8 5 4 3]	0.009998468
23	[10 8 4 3]	0.010228907
24	[10 8 5 4]	0.009961966

7. Conclusions and future work

7.1 Conclusions

This PhD dissertation presents several novel approaches for predicting irregular work of different robotic systems. All proposed intelligent methods are based on the latest machine learning algorithms. After considering all the facts listed throughout the thesis, the following conclusions can be drawn:

- This is the first study that develops approach based on Neural Networks (NNs) for prediction of failures and faults in robots. The treated problem is important if we want to achieve new generation robots working along with humans in factory plants. One can consider a robotic system working in a structured or unstructured environment exposed to severe conditions such as: increased working hours, changeable working demands, possibility of collision with known/unknown objects, and/or presence of human workers near the robot workspace. In these cases it is crucial to ensure maximum safety and smallest deviation from the nominal operating mode by recognizing irregularities in robot behavior. Therefore, the prediction of robot failures is important, since this can provide a continuous and undisturbed work using a backup emergency control commands.
- The first failure problem includes real forces and torques recorded during execution of a specific task. These are used to train NNs in order to predict one of four possible working cases (normal, collision, front collision and obstruction). The erroneous data is also implemented in the NN training set in order to fully investigate robustness of the proposed approach.

- In order to fairly investigate proposed prediction approach, two software environments, Matlab and BPnet, are utilized in the experiments described in this dissertation.
- Various different learning algorithms and architectures are employed in order to obtain. Two types of neural networks are used: feedforward and recurrent (Elman) NNs. In total, 7 different learning algorithms and 24 NN architectures are implemented in order to find optimal solution for the problem of robot execution failures prediction. Each multilayer feedforward NN with different learning algorithm and architecture that consists of 1, 2, 3, or 4 hidden layers is evaluated several times, and the same NN architectures are trained using Elman recurrent NN. Experimental results indicate that Bayesian Regularization algorithm is the best choice for the prediction problem with prediction rate of 95.4545 percent, despite having the erroneous or otherwise incomplete sensor measurements invoked in the dataset. The experimental results show that the NN outperforms the Naïve Bayes, Decision Trees and Support Vector Machine based algorithms [43] employed for the prediction of robot execution failures. These results prove assumed hypothesis *that Soft Computing technique (NN) can be used for increasing the reliability and success rate of prediction unit in industrial robotic system*.
- Two additional failure prediction problems are treated in this dissertation. Several experiments in real time are conducted on an nonholonomic mobile robot *Khepera II* in a laboratory model of manufacturing environment.
- First real world failure problem refers to the robot obstacle detection in indoor environment. Six infrared sensors mounted on the mobile robot are used to obtain information of the obstacle located left and

right from the platform. Randomly generated failed sensor data is integrated into the training set so as to test the NN performance in this task. The result show that in over 96 percent of all tested cases NN recognized failed value, meaning that the obstacle location is successfully determined after the failed information is replaced with the expected one.

- Second real world problem refers to the failure prediction in a mobile robot trajectory tracking problem. This problem is important if we want to secure safe mobile robot navigation in technological environment. Consider that mobile robot wheels command unit is not working properly all the time, and that in certain control iterations it gives unexplainable large/small commands for tracking the specific trajectory. In this case, NNs can predict these irregularities, with the aim to invoke a nominal control value in the command dataset. In this manner, the bad wheel command is replaced with the desired (calculated) value, and the robot motion is continued without difficulties.
- Two independent trajectories are employed so as to objectively test the proposed intelligent approach. The tracking of the *M-shaped* and *Labyrinth-type* trajectories showed as a fairly easy task for the developed prediction method. In more than 99 percent of the cases, the network predicted the wheel command failure, which is next replaced with the desired value in order to successfully track chosen trajectory. The experiments show that a mobile robot can track desired trajectories with a minimal error in every control iteration, which evidence the robustness and the applicability of the proposed approach.

- Finally, it can be concluded that the real world obstacle detection and trajectory tracking experiments prove remain two research hypotheses. The experiments show that it is possible to develop *failure prediction unit that enable corrections of robot behavior online*, and also that *the control system based on the neural networks and the empirical (i.e. gathered) sensor information from the environment can be employed for the obstacle detection and trajectory tracking in a laboratory model of a manufacturing environment*.

7.2 Recommendations for future work

The developed intelligent approach for failure prediction proves robustness and usefulness in real time control of robotic systems. Other research studies also acknowledge presented approach; for example, in [158] novel method use kernel based Extreme Learning Machines coupled with particle swarm optimization in order to optimize the parameters of kernel functions of neural networks for improving the prediction accuracy of robot execution failures. This is therefore a research area that needs additional investigation. Future work on improving presented approach includes two main directions:

1. Failure execution dataset should be expanded. In other words, using data available in [91], the robotic failures need to include different failure situation (apart from an approaching behavior studied in this dissertation).
2. Other Computational Intelligence techniques should be used for the failure prediction, e.g. [124, 159]. For example, it would be interesting to see the prediction results using swarm intelligence optimized Support Vector Machines.

References

[1] Miljković, Z and Aleksendrić, D. Artificial neural networks – Solved examples with theoretical background (in Serbian). Belgrade: University of Belgrade – Faculty of Mechanical Engineering, 2009.

[2] Lolasa, S and Olatunbosun, OA. Prediction of vehicle reliability performance using artificial neural networks. Expert Syst Appl, Vol. 34 No. 4, pp. 2360–2369, 2008.

[3] Miljković, Z, Gerasimović, M, Stanojević, Lj, et al. Using artificial neural networks to predict professional movements of graduates. Croat J Educ, Vol. 13, No. 3, pp. 117–141, 2011.

[4] Miljković, Z, Vuković, N, Mitić, M, et al. New hybrid vision-based control approach for automated guided vehicles. Int J Adv Manuf Technol, Vol. 66, Issue 1–4, pp. 231–249, 2013.

[5] Diryag, A, Mitić, M and Miljković, Z. Neural Networks for Prediction of Robot Failures. Proc IMechE, Part C: J Mechanical Engineering Science Vol. 228, No. 8, pp. 1444-1458, 2014. (Available online_first published on October 10, 2013 as DOI: 10.1177/0954406213507704), http://pic.sagepub.com/content/228/8/1444.

[6] Miljković, Z, Mitić, M, Lazarević, M, et al. Neural network reinforcement learning for visual control of robot manipulators. Expert Syst Appl, Vol. 40, No. 5, pp. 1721–1736, 2013.

[7] Al-Assadi, HMAA, Mat Isa, AA, Hasan, AT, et al. Development of a real-time-position prediction algorithm for under-actuated robot manipulator by using of artificial neural network. Proc IMechE, Part C: J Mechanical Engineering Science, Vol. 225, No. 8, pp. 1991–1998, 2011.

[8] Althoefer, K, Lara, B, Zweiri, YH, et al. Automated failure classification for assembly with self-tapping threaded fastenings using artificial neural networks. Proc IMechE, Part C: J Mechanical Engineering Science, Vol. 222, No. 6, pp. 1081–1095, 2008.

[9] Bevilacqua, M, Braglia, M, Frosolini, M, et al. Failure rate prediction with artificial neural networks. J Qual Maint Eng, Vol. 11, No. 3, pp. 279 – 294, 2005.

[10] Van, M, Kang, HJ and Ro, YS. A robust fault detection and isolation scheme for robot manipulators based on neural networks. Lecture notes in computer science (Springer-Verlag, Berlin), Vol. 6838, pp. 25–32, 2011.

[11] Kreuziger, J. Application of machine learning to robotics-an analysis. In In Proceedings of the Second International Conference on Automation, Robotics, and Computer Vision (ICARCV'92), 1992.

[12] Sato, T. and Meister S. A model enhanced intelligent and skillful teleoperational robot system. In Proc. Robotics Research - The Fourth Int. Symposium, pp. 155-162, 1987.

[13] Asada, H. and Izumi, H. Automatic program generation from teaching data for the hybrid control of robots. IEEE Transactions on Robotics and Automation, Vol. 5, No. 2, pp. 166-173, 1989.

[14] Christiansen, AD et al. Learning reliable manipulation strategies without initial physical models. In Proc. IEEE Robotics and Automation, Cincinnati, pp. 1224-1230, 1990.

[15] Vaaler, EG and Seering, WP. A machine learning algorithm for automated assembly. In Proc. IEEE Robotics and Automation, Sacramento, pp. 2231-2237, 1991.

[16] Dufay, B and Latombe J-C. An approach to automatic robot programming based on inductive learning. In M. Brady et al., eds., Proc. Robotics Research: The 1st Int. Symposium (The MIT Press), pp. 97-115, 1984.

[17] Chen, K. Smooth path tracking through symbolic computations. Techn. Report ISG 86-13, Dept. of CS, Univ. of Illinois at U.-C., 1986.

[18] Bartenstein, O and Inoue, H. Learning assisted robot programming. In Proc. Robotics Research - The 4th Int. Symposium, 1987.

[19] Sammut, C. and Hume, D. Observation and generalisation in a simulated robot world. In Proc. 4th Int. Workshop on ML, Irvine, pp. 267-273, 1987.

[20] Heise, R. Demonstration instead of programming: Focusing attention in robot task acquisition. Technical Report 89/360/22, Dept. of Computer Science, Univ. of Calgary, Canada, 1989.

[21] Langley, P, et al. An integrated cognitive architecture for autonomous agents. Technical Report 89-28, Univ. of California, Irvine, Dept. of Information and Computer Science, 1989.

[22] Tan, M. CSL: A cost-sensitive learning system for sensing and grasping objects. In Proc. IEEE Robotics and Automation, Cincinnati, pp. 858-863, 1990.

[23] Kadie, CM. Continuous conceptual set covering: Learning robot operators from examples. In Proc. 8th Int. Workshop on ML, Evanston, pp. 615-619, 1991.

[24] Findler, NV and Ihrig, LH. Analogical reasoning by intelligent robots. In A.K.C. Wong and A. Pugh, eds., Machine Intelligence and Knowledge Engineering for Robotic Applications, Vol. F33 of NATO ASI, pp. 269-282, 1987.

[25] Segre, AM. Machine Learning of Robot Assembly Plans. Kluwer Academic Publishers, 1988.

[26] Bennett, SW. Reducing real world failures of approximate explanation based rules. In Proc. 7th Int. Conf. on ML, Austin, pp. 226-234, 1990.

[27] Laird, JE. et al. Correcting and extending do-main knowledge using outside guidance. In Proc. 7th Int. Conf. on ML, Austin, pp. 235-243, 1990.

[28] Kaiser, M. and Kreuziger, J. Integration of symbolic and connectionist processing to ease robot programming and control. In ECAI'94 Workshop on Combining Symbolic and Connectionist Processing, 1994.

[29] Kreuziger, J. Application of machine learning to robotics - an analysis. In Proceedings of the Second International Conference on Automation, Robotics, and Computer Vision (ICARCV '92), Singapore, 1992.

[30] Hagan, MT., Demuth, HB, Beale, M. Neural Network Design. Brooks/Cole Publishing Company, USA. 1996.

[31] Webb, AR., Statistical Pattern Recognition. 2nd ed. West Sussex, England: John Wiley & Sons, Ltd. 2002.

[32] Kabari, LG and Bakpo, FS. Diagnosing skin diseases using an artificial neural network. In Adaptive Science & Technology, ICAST 2009. 2nd International Conference on, pp. 187-191, 2009.

[33] Haykin, S. Neural Networks: A Comprehensive Foundation. Prentice Hall, The 2nd edition, 1998.

[34] He, Q. Neural Network and its Application in IR. Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign Spring, 1998.

[35] Das, SK, Kumar, A, Das, B, and Burnwal, AP. On soft computing techniques in various areas. Computer Science & Information Technology, Vol. 59, 2013.

[36] L-X Wang, A. Course in Fuzzy Systems and Control. Upper Saddle River, NJ: Prentice- Hall, 1997.

[37] Gol, DE. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989.

[38] Roy, R, Furuhashi, T and Chawdhry, P. K. (eds.). Advances in Soft Computing: Engineering Design and Manufacture. London, UK, Springer, 1998.

[39] Zadeh, LA. Roles of Soft Computing and Fuzzy Logic in the Conception, Design and Deployment of Information/Intelligent Systems, Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, Kaynak O. et al (Eds.), pp. 1-9, 1998.

[40] Dasgupta, D. An artificial immune system as a multi-agent decision support system. in Proc. IEEE International Conference on Systems, Man, and Cybernetics, San Diego, CA, pp. 3816-3820, 1998.

[41] Siegwart, R, Nourbakhsh, I and Scaramuzza, D. Introduction to Autonomous Mobile Robots. MIT Press, 2nd edition, 2011.

[42] Gao, XZ. Soft Computing Methods for Control and Instrumentation. PhD thesis, 1999.

[43] Frank, H. An Overview on Soft Computing in Behavior BasedRobotics, 2002.

[44] Miljković, Z and Babić, B. Empirical control strategy for learning industrial robot. FME Trans, Vol. 35, No. 1, pp. 1–8, 2007.

[45] Rodrigo, S, Anthony, A, Rodney, G. Failure-Tolerant Path Planning for the PA-10 Robot Operating Amongst Obstacles. in Proc. IEEE Int Conf Robot Automat, New Orleans. LA, April 2004

[46] Koohi, T, Mirzaie, E and Tadaion, G. Failure Prediction Using Robot Execution Data. In: Proceedings of the 5th Symposium on advances in science & technology, Mashhad, Iran, 2011.

[47] Wikman, TS, Branicky, MS and Newman, WS. Reflexive collision avoidance: A generalized approach. in Proc. IEEE Int Conf Robot Automat, Atlanta, GA, pp. 31–36, 1993.

[48] Ting, Y, Tosunoglu, S and Tesar, D. A control structure for fault tolerant operation of robotic manipulators. in Proc. IEEE Int Conf Robot Automat, Atlanta, GA, pp. 684–690, 1993.

[49] Paredis, CJJ. and Khosla, PK. Fault tolerant task execution through global trajectory planning. Rel. Eng. Syst. Safety, Vol. 53, pp. 225–235, 1996.

[50] Park, J, Chung, W-K and Youm, Y. Failure recovery by exploiting kinematic redundancy. In 5th Int. Workshop Robot Human Commun., Tsukuba, Japan, pp. 298–305, 1996.

[51] Visinsky, ML, Walker, ID, and Cavallaro, JR. Fault detection and fault tolerance in robotics. In Proceedings of NASA Space Operations, Applications, Research Symposium, pp. 262-271, 1991.

[52] Horak, DT. Failure Detection in Dynamic Systems with Modeling Errors. AIAA Journal of Guidance, Control, and Dynamics, Vol. 11. No. 6, pp. 508–516, November-December 1988.

[53] Twala B. Robot execution failure prediction using incomplete data. In: Proceedings of the IEEE international conference on robotics and biomimetics, Guilin, China, pp. 1518–1523, 2009.

[54] Miljković, Z. Systems of artificial neural networks in production technologies (in Serbian). Belgrade: University of Belgrade – Faculty of Mechanical Engineering, 2003.

[55] Rumelhart, DE, Hinton, GE and Williams, RJ. Learning internal representations by error propagation in parallel distributed processing. In: Parallel distributed processing (ed. Rumelhart DE, McClelland JL), Cambridge, UK, pp. 318–362, 1986.

[56] Dixon, W, Walker, I, Dawson, D, et al. Fault detection for robot manipulators with parametric uncertainty: A prediction-error-based approach. IEEE Trans Robot Autom, Vol. 16, No. 6, pp. 689–699, 2000.

[57] Visinsky, M, Cavallaro, J and Walker, I. Robotic fault detection and fault tolerance: A survey. Reliab Eng Syst Safety, Vol. 46, No. 2, pp. 139–158, 1994.

[58] Sreevijayan, D, Tosunoglu, S and Tesar, D. Architectures for fault tolerant mechanical systems. In: Proceedings of the IEEE mediterranean electrotechnical conference, Antalya, Turkey, pp. 1029–1033, 1994.

[59] Vemuri, A and Polycarpou, MM. A dynamic fault tolerance framework for remote robots. IEEE Trans Robot Automat, Vol. 11, No. 4, pp. 477–490, 1995.

[60] Shin, JH and Lee, JJ. Fault detection and robust fault recovery control for robot manipulators with actuator failures. In: Proceedings of the IEEE international conference on robotics and automation, Detroit, USA, pp. 861–866, 1999.

[61] Brambilla, D, Capisani, LM, Ferrara A, et al. Fault detection for robot manipulators via second-order sliding modes. IEEE Trans Ind Electron Vol. 55, No. 11, pp. 3954–3963, 2008.

[62] Halder, B and Sarkar, N. Robust fault detection of a robotic manipulator. Int J Robot Res, Vol. 26, No. 3, pp. 273-285, 2007.

[63] Muradore, R and Fiorini, P. A PLS-based statistical approach for fault detection and isolation of robotic manipulators. IEEE Trans Ind Electron. Vol. 59, No. 8, pp. 3167–3175, 2012.

[64] Valavavins, KP, Jacobson, CA and Gold, BH. Integration Control and Failure Detection with Application to the Robot payload Variation Problem. Journal of Intelligent and Robotic Systems, Vol. 4, pp.145-173, 1991.

[65] Dixon, W, Walker, I and Dawson, D. Fault detection for wheeled mobile robots with parametric uncertainty. In: Proceedings of the IEEE/ASME international conference on advanced intelligent mechatronics, Como, Italy, pp. 1245–1250, 2001.

[66] Roumeliotis, SI, Sukhatme, GS and Bekey, GA. Sensor fault detection and identification in a mobile robot. In: Proceedings of the IEEE/RSJ international conference on intelligent robots and systems, Victoria, Canada, pp. 1383–1388, 1998.

[67] Hashimoto, M, Kawashima, H, Nakagami, T, et al. Sensor fault detection and identification in dead-reckoning system of mobile robot: interacting multiple model approach. In: Proceedings of the IEEE/RSJ international conference on intelligent robots and systems, Maui, USA, pp. 1321–1326, 2001.

[68] Ming, Y, Danwei, W and Qijun, C. Prediction of multiple failures for a mobile robot steering system. In: Proceedings of the IEEE international symposium on industrial electronics, Hangzhou, China, pp. 1240–1245, 2001.

[69] Coskun, N. and Yildrim, T. The effects of training algorithms in MLP network on image classification, in Proc. Int. Joint Conf. on Neural Networks, Vol. 2, pp. 1223-1226, 2003.

[70] Tu, J. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. Journal of Clinical Epidemiology, Vol. 49, No. 11, pp. 1225–1231, 1996.

[71] Zaamout, KM. Two novel ensemble approaches for improving classification of neural networks., Master Thesis, Lethbridge, Alberta: University of Lethbridge, Dept. of Mathematics and Computer Science, 2012.

[72] Cheng, B and Titterington, DM. Neural networks: A review from a statistical perspective. Statistical Science, Vol. 9, No. 1, pp. 2–30, 1994.

[73] Lu, B and Ito, M. Task decomposition and module combination based on class relations: a modular neural network for pattern classification. IEEE Transactions on Neural Networks, Vol. 10, No. 5, pp. 1244–1256, 1999.

[74] Hansen LK, and Salamon, P. Neural network ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 10, pp. 993–1001, 1990.

[75] Opitz, DW and Shavlik JW. Actively searching for an effective neural-network ensemble. Connection Science, Vol. 8, No. 3, pp. 337–353, 1996.

[76] Zhou, Z, Wu, J and Tang W. Ensembling neural networks: Many could be better than all. Artificial Intelligence, Vol. 137, pp. 239–263, 2002.

[77] Tu, J. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. Journal of Clinical Epidemiology, Vol. 49, No. 11, pp. 1225–1231, 1996.

[78] Othman, M, Selamat, MH, Muda, Z and Abdullah, LN. Modeling The Tower Of Hanoi Using Neural Network. Jurnal Teknologi, Vol. 21, No. 1, pp. 47-56, 1993.

[79] Akerkar, R. Introduction to artificial intelligence. PHI Learning Pvt. Ltd., 2005.

[80] Das, SK, Tripathi, S and Burnwal, AP. Some Relevance Fields of Soft Computing Methodology. International Journal of Research in Computer Applications and Robotics, Vol. 2, No. 1-6, 2014.

[81] Adriaans, P and Zantinge, D. Data mining. New York: Addison-Wesley, 1997.

[82] Wong, BK, Bonovich, TA and Selvi, Y. Neural network applications in business: A review and analysis of the literature (1988-95). Decis Support Syst, Vol. 19, No. 4, pp. 301–320, 1997.

[83] Vuković, N and Miljković, Z. A growing and pruning sequential learning algorithm of hyper basis function neural network for function approximation. Neural Networks, Vol. 46C, pp. 210–226, 2013. (DOI: 10.1016/j.neunet.2013.06.004).

[84] Liao, SH and Wen, CH. Artificial neural networks classification and clustering of methodologies and applications – literature analysis form 1995 to 2005. Expert Syst Appl, Vol. 32, No. 1, pp. 1–11, 2007.

[85] Schalkoff, RJ. Artificial neural networks. New York: McGraw-Hill, 1997.

[86] Russell, BS. A comparison of neural network and regression models for Navy retention modeling. MSc Thesis, Monterey, California. Naval Postgraduate School, 1993.

[87] Stanley, J. Introduction to Neural Networks, (Sierra Madre, CA: California Scientific Software), 1989.

[88] Beale, R and Jackson, T. Neural Computing. An Introduction, 2nd ed., (New York: Adam Hilger), 1991.

[89] Osmond, C. NeuralWare Neural Computing in Business. Industry and Government class notes, taken December 11-14, Pittsburgh PA, 1992.

[90] McClelland, J and Rumelhart, D. Parallel Distributed Processing, (Cambridge MA: MIT Bradford Press), 1986.

[91] Blake, CL and Merz, CJ. UCI Repository of Machine Learning databases, http://archive.ics.uci.edu/ml/

[92] Gopalakrishnan, K. Effect of training algorithms on neural networks aided pavement diagnosis. International Journal of Engineering, Science and Technology, Vol. 2, No. 2, pp. 83-92, 2010.

[93] Hagan, MT and Menhaj, MB. Training feedforward networks with the Marquardt algorithm, IEEE Transactions on Neural Networks, Vol. 5, No. 6, pp. 989-993, 1994.

[94] Pham, D., Sagiroglu, S. Training multilayered perceptrons for pattern recognition: a comparative study of four training algorithms. International Journal of Machine Tools and Manufacture, Vol. 41, pp. 419–430, 2001.

[95] Anastasiadis, AD, Magoulas, GD and Vrahatis, MN. New globally convergent training scheme based on the resilient propagation algorithm. Neurocomputing, Vol. 64, pp.253–270, 2005.

[96] http://www-rohan.sdsu.edu/doc/matlab/toolbox/nnet/backpr57.html

[97] Hager, WW and Zhang, H. A survey of nonlinear conjugate gradient methods. Pacific of Journal Optimization. Vol. 2, pp. 35–58, 2006.

[98] Moller, M. A scaled Conjugate Gradient Algorithm for Fast supervised learning Vol. 6, pp. 525-533, 1993.

[99] Haykin, S and Kosko, B (ed.). Intelligent Signal Processing, IEEE Press, 2001.

[100] http://www.heatonresearch.com/wiki/Elman_Neural_Network

[101] Linda, O. Applications of Computational Intelligence in Critical Infrastructures: Network Security, Robotics, and System Modeling Enhancements. MSc thesis, University of Idaho, 2009.

[102] Haykin, S. Neural Networks and Learning Machines – Third Edition, Prentice Hall, 2008.

[103] Witten, H. Frank, E. Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann Publishers, 2005.

[104] http://www.iitbhu.ac.in/faculty/min/rajesh-rai/NMEICT-Slope/lecture/c14/l1.html

[105] Adewole, AC. Investigation of methodologies for fault detection and diagnosis in electric power system protection. Master Thesis, Cape Peninsula University of Technology, 2012.

[106] Veelenturf, LPJ. Analysis and Application of Artificial Neural Networks. Prentice Hall, NJ, 1995.

[107] Jones, T. Artificial Intelligence: A Systems Approach. Hingham, Massachusetts: Infinity Science Press LLC, 2008.

[108] Demuth, H, Beale, M and Hagan, M. Neural Network Toolbox for Use with MATLAB, Users Guide Version 4, 2004.

[109] Khattab, AA. ANN Based Mechanistic Force Model for Face Milling Processes. Master Thesis, American University of Sharjah, 2011.

[110] Skapura, D. Building Neural Networks. ACM Press, Addison-Wesley Publishing Company, New York, 1996.

[111] El Kadi, H and Al-Assaf, Y. The Use of Neural Networks in the Prediction of the Fatigue Life of Different Composite Materials. 16th International Conference on Composite Materials, Kyoto, Japan, July 8-13, 2007.

[112] MATLAB software, www.mathworks.com

[113] Maleki, S, et al. Prediction of shear wave velocity using empirical correlations and artificial intelligence methods. NRIAG Journal of Astronomy and Geophysics, Vol. 3, No. 1, pp. 70-81, 2014.

[114] Mood, A, Graybill, F and Boes, D. Introduction to the Theory of Statistics (3rd ed.), McGraw-Hill, pp. 229, 1974.

[115] Visual Basic 5.0. Enterprise edition. Microsoft Corporation, 1997.

[116] Danišová, Nina. Application of intelligent manufacture system in the flexible assembly cell. Journal of engineering. Vol. 3, (ISSN 1584-2673), 2007.

[117] Rao, M, Wang, Q and Cha, J. Integrated distributed intelligent systems in manufacturing, Chapman &Hall, 1993.

[118] Miljković, Z, Vuković, N, Babić, B. Mobile Robot Localization in a Manufacturing Environment, Proceedings of the 3rd International Conference on Manufacturing Engineering (ICMEN 2008) and EUREKA Brokerage Event, pp. 485-494, Kallithea of Chalkidiki, Greece, 1-3 October, 2008.

[119] Groover, MP. Automation, Production Systems, and Computer-Integrated Manufacturing, 2nd Edition, Prentice Hall, 2001.

[120] Kopacek, P. Intelligent Manufacturing: Present State and Future Trends, Journal of Intelligent and Robotic Systems, Vol. 26, pp. 217-229, 1999.

[121] Mnif, F and Touati F. An adaptive control scheme for nonholonomic mobile robot with parametric uncertainty. International Journal of Advanced Robotic Systems, Vol. 2, No. 1, pp 59-63, 2005.

[122] Miljkovic, Z. Hierarchical Intelligent Robot Control Based on Artificial Neural Network System, Journal Mathematical Modelling and Scientific Computing (ISSN 1067-0688), Vol. 8, No. 1-2, pp. 331-336, Principia Scientia, Printed in USA, 1997.

[123] Miljkovic, Z, Lazarevic, I. Control Strategy for Learning Industrial Robot Based on Artificial Neural Network System, Proceedings of the International Conference on Systems, Signals, Control, Computers – SSCC'98, Vol. 3, pp. 124-128, Durban-South Africa, September 1998.

[124] Mitić, M, Vuković, N, Diryag, A, Miljković, Z. Learning Motion Trajectories and Visual Commands of a Nonholonomic Mobile Robot Using Metaheuristic Technique. Proceedings of the 5th International Conference on Manufacturing Engineering (ICMEN 2014), pp. 89-98, Thessaloniki, Greece, 1-3 October, 2014.

[125] Olumide, O and Dumitrache, I. Fuzzy Control Of Autonomous Mobile Robot, ISSN 1454-234x, U.P.B. Sci. Bull., Series C, Vol. 72, Iss. 3, 2010.

[126] Velagic, J, Osmic, N. and Lacevic, B. Neural network controller for mobile robot motion control. International journal of intelligent systems and technologies Vol. 3, No. 2, pp. 127-133, 2008.

[127] Horlink, K, Stinchombe, M and White, H. Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks. Neural Networks, Vol. 3, pp. 551-560, 1990.

[128] Bugeja, MK, Fabri, SG and Camilleri, L. Dual Adaptive Dynamic Control of Mobile Robots Using Neural Networks. IEEE Transactions on Systems, Man, and Cybernetics-Part B: cybernetics, Vol. 39, No. 1, pp. 129-141, 2009.

[129] Imen, M, Mansouri, M and Shoorehdeli, MA. Tracking Control of Mobile Robot Using ANFIS. Proceedings of the IEEE International Conference on Mechatronics and Automation, Beijing, Chain, 7-10 August, pp. 422-427, 2011.

[130] Dezfoulian, SH. A Generalized Neural Network Approach to Mobile Robot Navigation and Obstacle Avoidance. University of Windsor, Msc thesis, 2012.

[131] Miljković, Z, Mitić, M, Babić, B, Diryag, A. Q-Learning Algorithm for a Mobile Robot Obstacle Avoidance in an Unknown Environment Based on Artificial Neural Networks, Proceedings of the 4th International

Conference on Manufacturing Engineering (ICMEN 2011), pp. 431-440, Thessaloniki, Greece, 3-5 October, 2011.

[132] Silva, C, Crisostomo, M and Ribeiro, B. MONODA: a neural modular architecture for obstacle avoidance without knowledge of the environment. Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, Vol.6, pp. 334-339, 2000.

[133] Singh, MK and Parhi, DR. Path optimisation of a mobile robot using an artificial neural network controller. International Journal of Systems Science, Vol. 42, pp. 107-120, 2011.

[134] Parhi DR and Singh, MK. Real-time navigational control of mobile robots using an artificial neural network. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, Vol. 223, pp. 1713-1725, 2009.

[135] Mohareri, O. Mobile robot trajectory tracking using neural networks. Master thesis, American University of Sharjah, 2009.

[136] Fierro, R and Lewis, FL. Control of a Nonholonomic Mobile Robot Using Neural Networks. in IEEE Transactions on Neural Networks, Vol. 9, pp. 589-600, 1998.

[137] Ye, J. Tracking control for nonholonomic mobile robots: Integrating the analog neural network into the backstepping technique. Neurocomputing, Vol. 71, pp. 3373-3378, 2007.

[138] Dong, F and Xu, WL. Adaptive Tracking Control of Uncertain Nonholonomic Dynamic System. IEEE Transactions on Automatic Control, Vol. 46, pp. 450-454, 2001.

[139] Pourboghrat, F and Karlsson, MP. Adaptive control of dynamic mobile robots with nonholonomic constraints. Computers & Electrical Engineering Vol. 28, No. 4, pp. 241-253, 2002.

[140] Liu, Z-y et al. Trajectory tracking control of wheeled mobile robots based on the artificial potential field. In Natural Computation, 2008. ICNC'08. Fourth International Conference on, Vol. 7, pp. 382-387. IEEE, 2008.

[141] Mitić, M, Miljković, Z, Vuković, N, Babić, B, Diryag, A. Prediction of Robot Execution Failures Using Neural Networks. Proceedings of the 35th International Conference on Production Engineering, pp. 335-339, Kopaonik, Serbia, 2013.

[142] Shimal, AF. Neural Controller for Nonholonomic Mobile Robot System Based on Position and Orientation Predictor. Journal IJCCCE, Vol. 11, No. 1, 2011.

[143] Mansournia P, et al. RoboCupRescue 2013 - Robot Leagua team Chardomalu Rescue robot (IRAN), Report, 2013.

[144] Jensfelt, P. Localization using laser scanning and minimalistic environmental models. Licentiate thesis, Automat. Contr. Dept., Royal Inst. Technol., Stockholm, Sweden, 1999.

[145] Murphy, R.R. Introduction to AI Robotics, MIT Press, Cambridge, Massachusetts, 2000.

[146] Fabien, L. Localisation through supervised learning. Master thesis, Umea University, Sweden, 2005.

[147] de Wit, CC, Siciliano, B. and Bastin, G (Eds). Theory of Robot Control, pp. 259-357, Springer, 1996.

[148] Martins-Filho, LS, Machado, RF, Rocha, R and Vale, VS. Commanding mobile robots with chaos. In ABCM Symposium Series in Mechatronics, Vol. 1, pp. 40-46, 2004.

[149] Durrant-Whyte, H. Sensor models and multisensor integration,"The International Journal of Robotics Research, Vol. 7, pp. 97-113, 1988.

[150] Crowley, J. Navigation for an intelligent mobile robot. IEEE Journal of Robotics and automation, Vol. 1, pp. 31-41, 1985.

[151] Borenstein, J and Feng, L. Gyrodometry: A new method for combining data from gyros and odometry in mobile robots. In: Proceedings of the International Conference on Robotics and Automation, (Minneapolis, Minnesota), pp. 423-428, IEEE, 1996.

[152] Zou et al. Neural Networks for Mobile Robot Navigation: A Survey. Science and Technology, pp. 1218-1226, 2006.

[153] Thrun, SB. Exploration and model building in mobile robot domains," Neural Networks, IEEE International Conference on, pp. 175-180, 1993.

[154] KheperaII User Manual. K-team Company, www.k-team.com

[155] Kube, CR. A minimal infrared obstacle detection scheme. The Robotics Practitioner:The Journal for Robot Builders, Vol. 2, No.2, pp. 15–20, 1996.

[156] Faress, KN., El Hagry, MT and El Kosy, A. Trajectory tracking control for a wheeled mobile robot using fuzzy logic controller. WSEAS Transactions on Systems, Vol. 4, No. 7, pp. 1017-1021, 2005.

[157] Ming, Y, Danwei, W and Qijun, C. Prediction of multiple failures for a mobile robot steering system. In: Proceedings of the IEEE international symposium on industrial electronics, Hangzhou, China, 2012, pp.1240–1245.

[158] Bin, L, Xuewen, R and Yiblin L. An Improved Kernel Based Extreme Learning Machine for Robot Execution Failures. The Scientific World Journal, Article ID 906546, 2014, http://dx.doi.org/10.1155/2014/906546

[159] Mitic, M, Vukovic, N, Petrovic, M, Petronijevic, J, Diryag, A, Miljkovic Z. Bioinspired metaheuristic algorithms for global optimization, 5th International Conference on Information Society and Technology, Kopaonik, Serbia, March 8-11, 2015.

Biography

Ali Karkara A. DIRYAG was born on 10th April 1969 in Libya, Libyan nationality. He finished his secondary school in Sirte, Libya in 1987. In 1994, he received his B.Sc. degree from Bright Star University of Technology, Braga, Libya, department of Mechanical Engineering, branch of Production Engineering. In 2004, Ali Diryag received M.Sc. degree with distinction grade from Beijing University of Aeronautics and Astronautics, China, School of solid Mechanics. Since March 2009, he has been Ph.D. candidate at the University of Belgrade, Faculty of Mechanical Engineering. In the period 1994-1995, he worked as an Engineer in the company for markets of oil and petroleum products, "Braga", Sirte, Libya, and from 1995 to 1996 he worked as Mechanical engineer in a company for service and maintenance administrative centers, Sirte, Libya. From 1996 to July 1997 Ali also worked as the head of the primary and secondary levels of training, the Ministry of Education and vocational training, Libya. From 1998 to 2000, he worked as the Director of the Training of the Ministry of Education and Vocational Training, Libya. From 2005 to 2009, he worked as an Assistant lecturer at Atthadi University, Sirte, Libya. He taught subjects such as: engineering drawing, engineering mechanics “static”, and engineering mechanics “dynamic”. He has engaged in his Ph.D. research and worked under the supervision of Professor Zoran Miljkovic in the field of production engineering, robotics and machine learning.

So far, Ali Diryag passed all subjects exams and has published papers in four international conferences and one international journal on SCI list with Impact Factor.

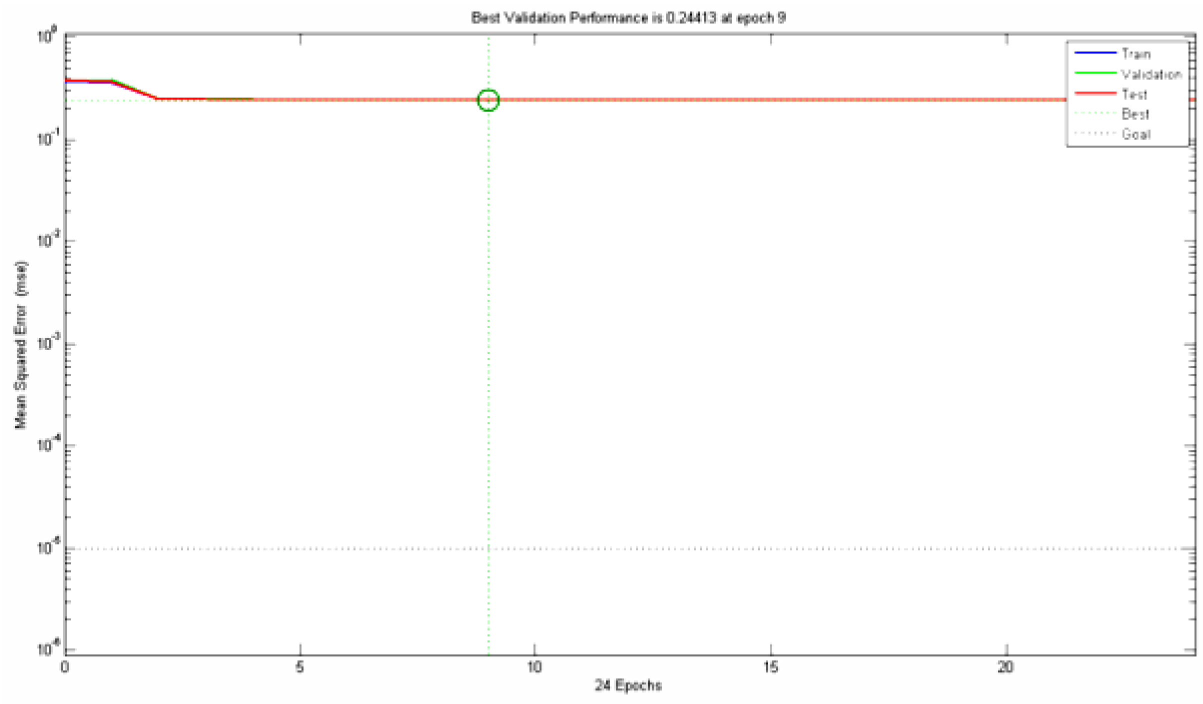
Appendix

The Appendix presents additional results in validation and performance plots of various neural networks with different numbers of units in hidden layers. Likewise, the results are given for all algorithms given in Table 2.

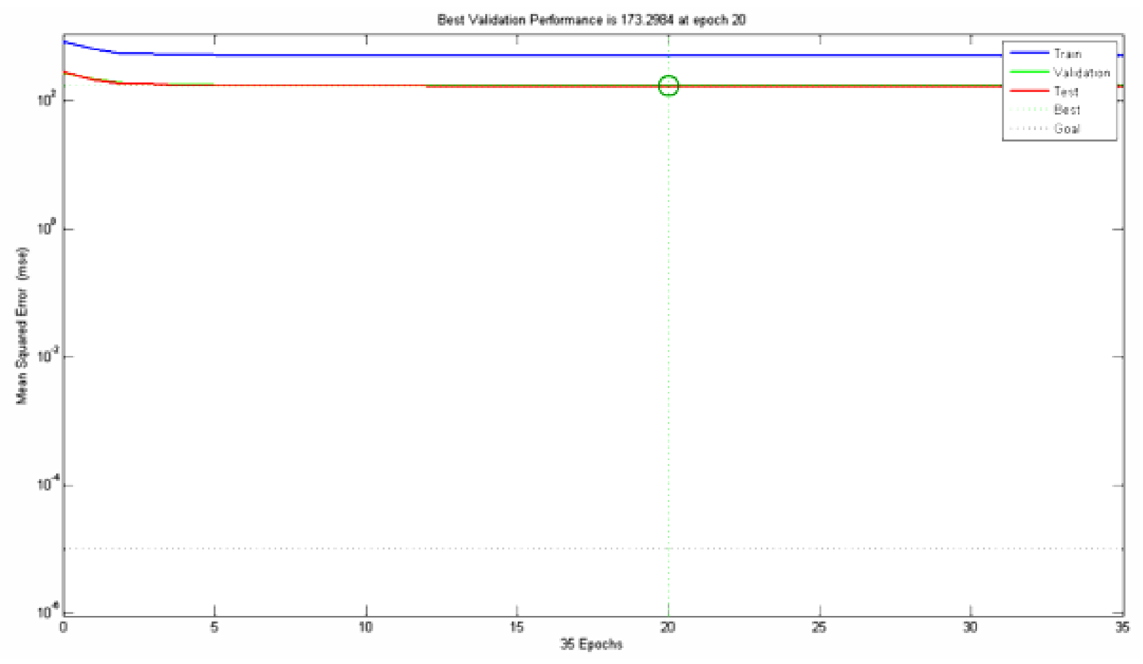
<u>One hidden layer</u>

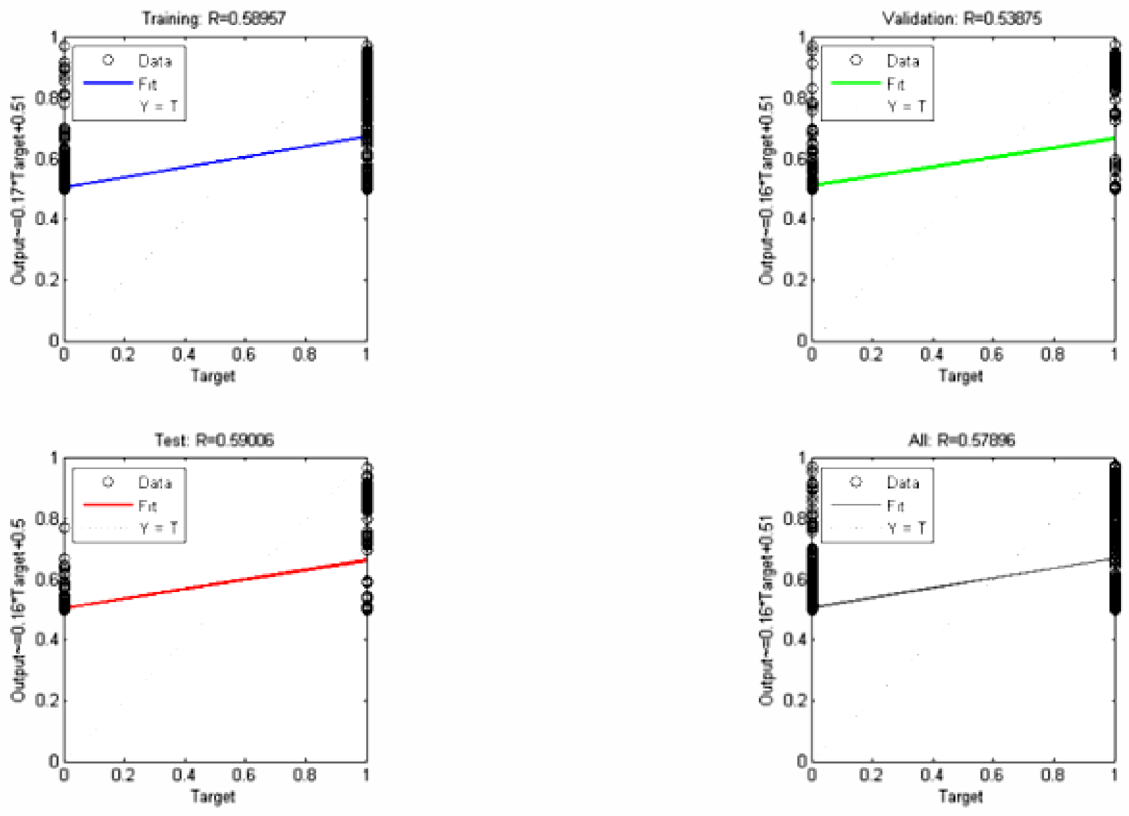
1) Performance plots for architecture [1]

   a) l Levenberg–Marquardt

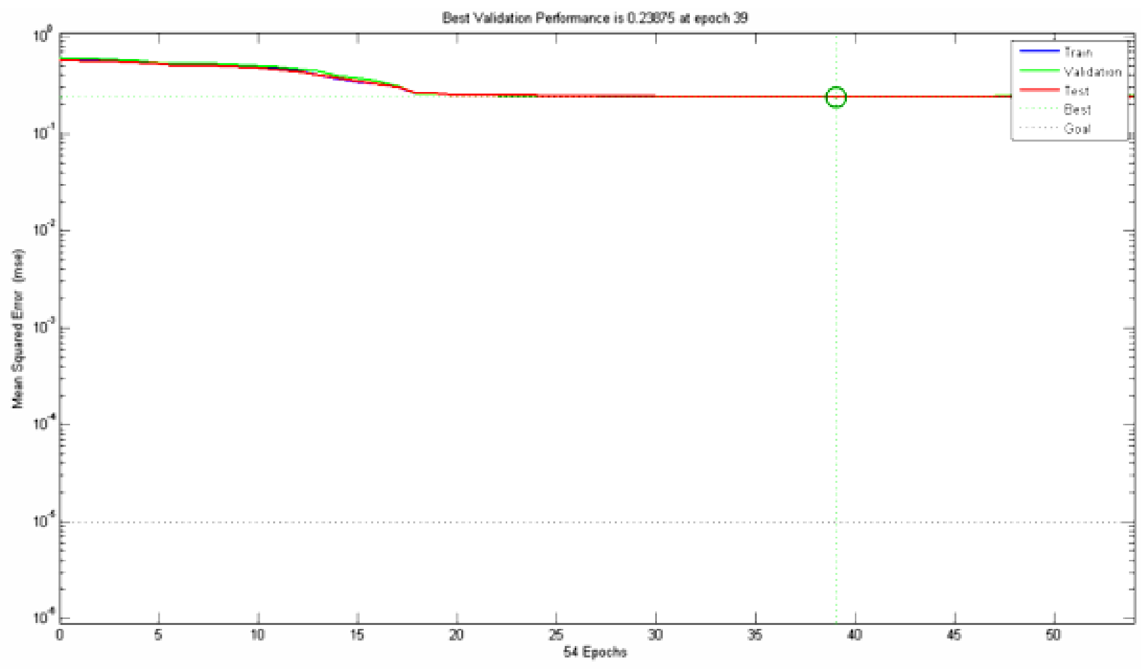


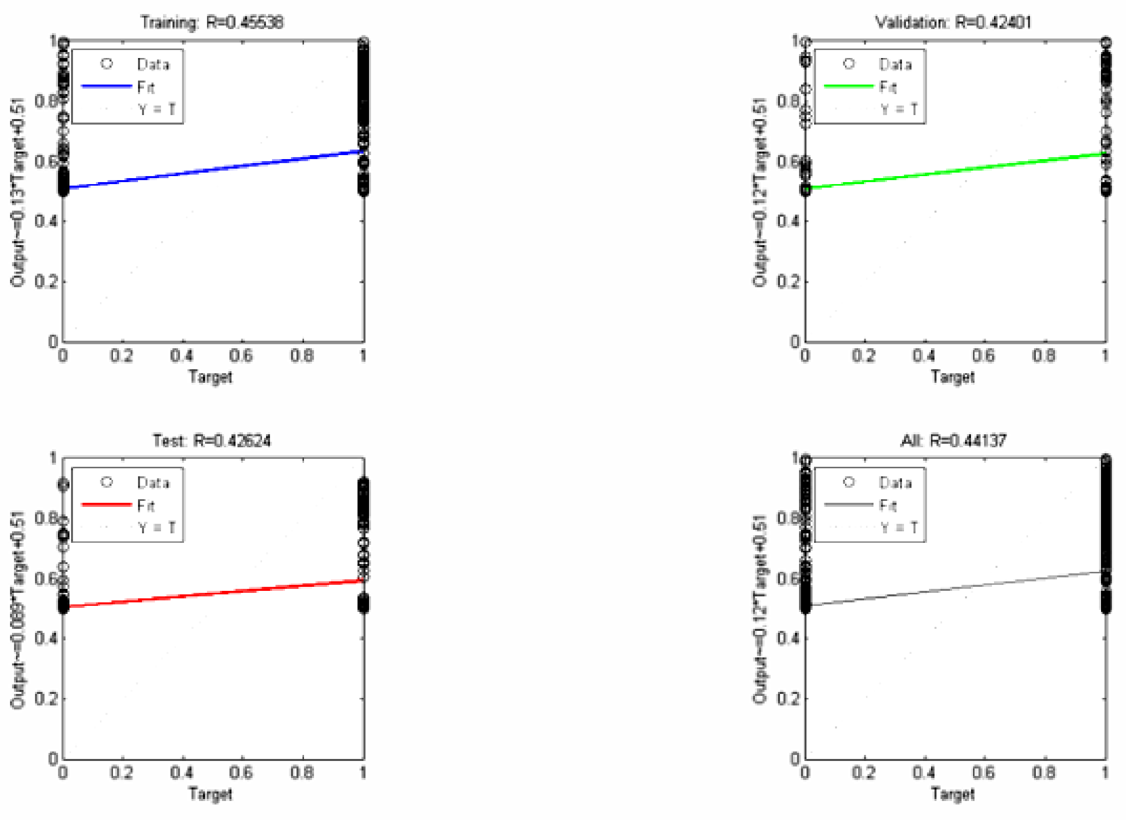
b) Bayesian Regularisation



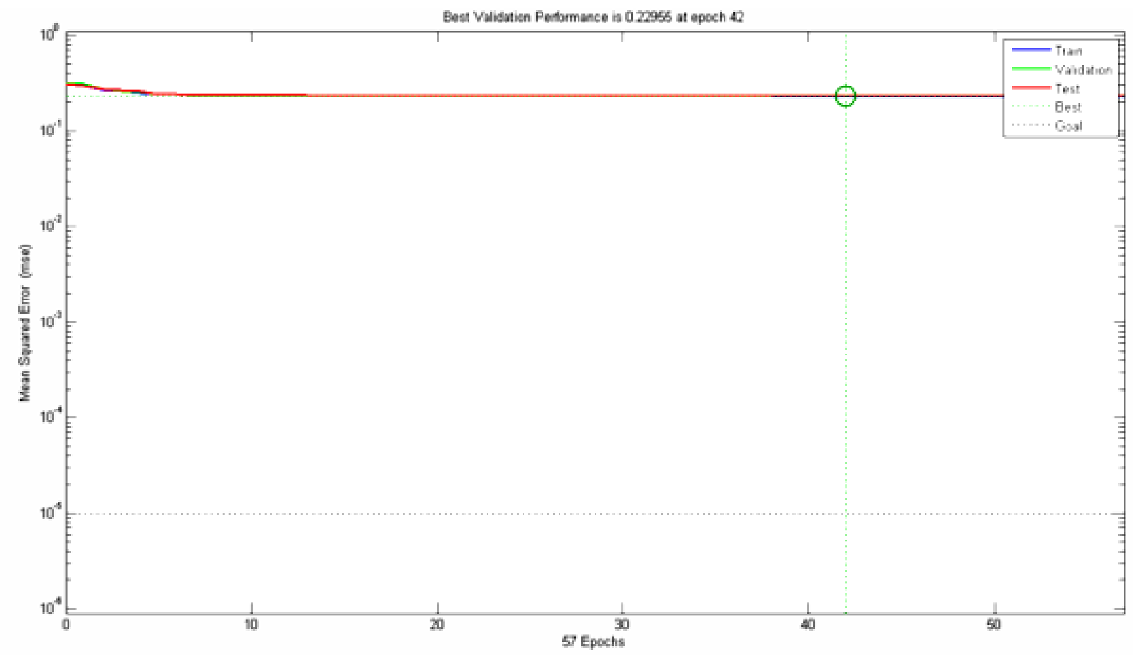


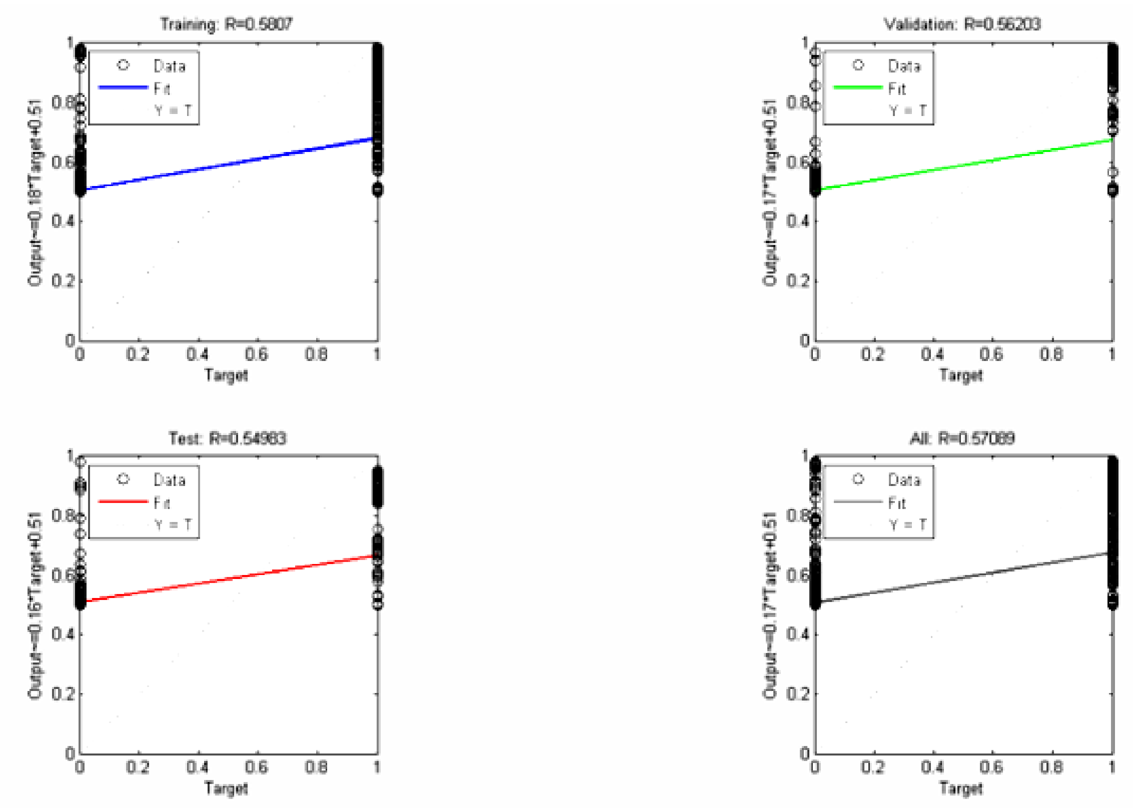
c) Resilient Backpropagation



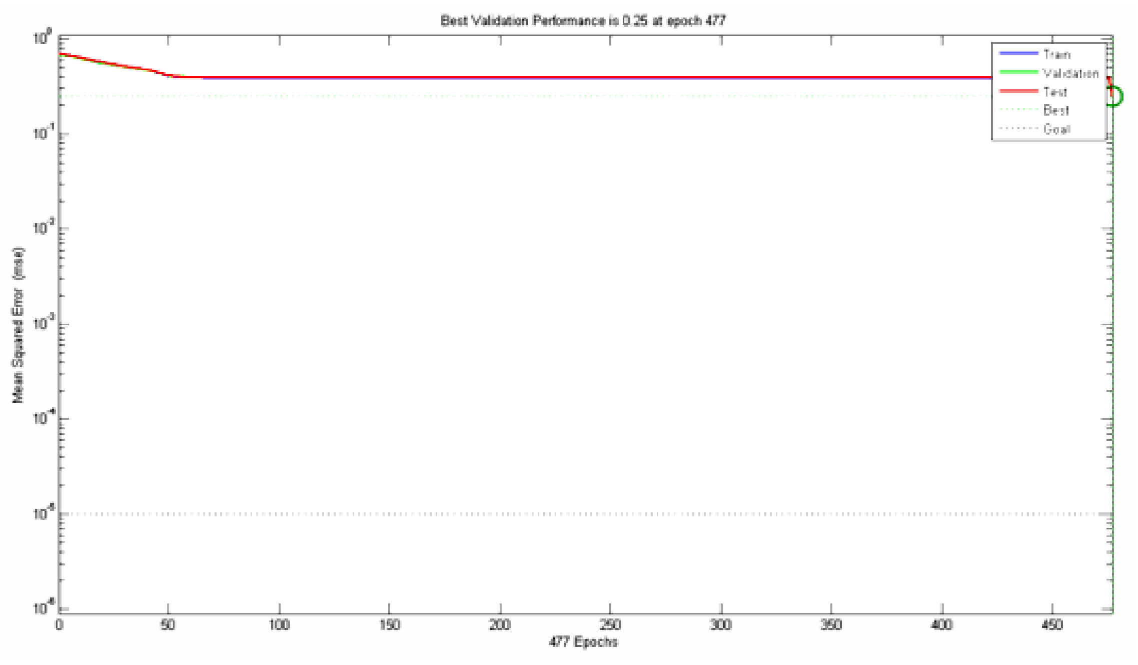


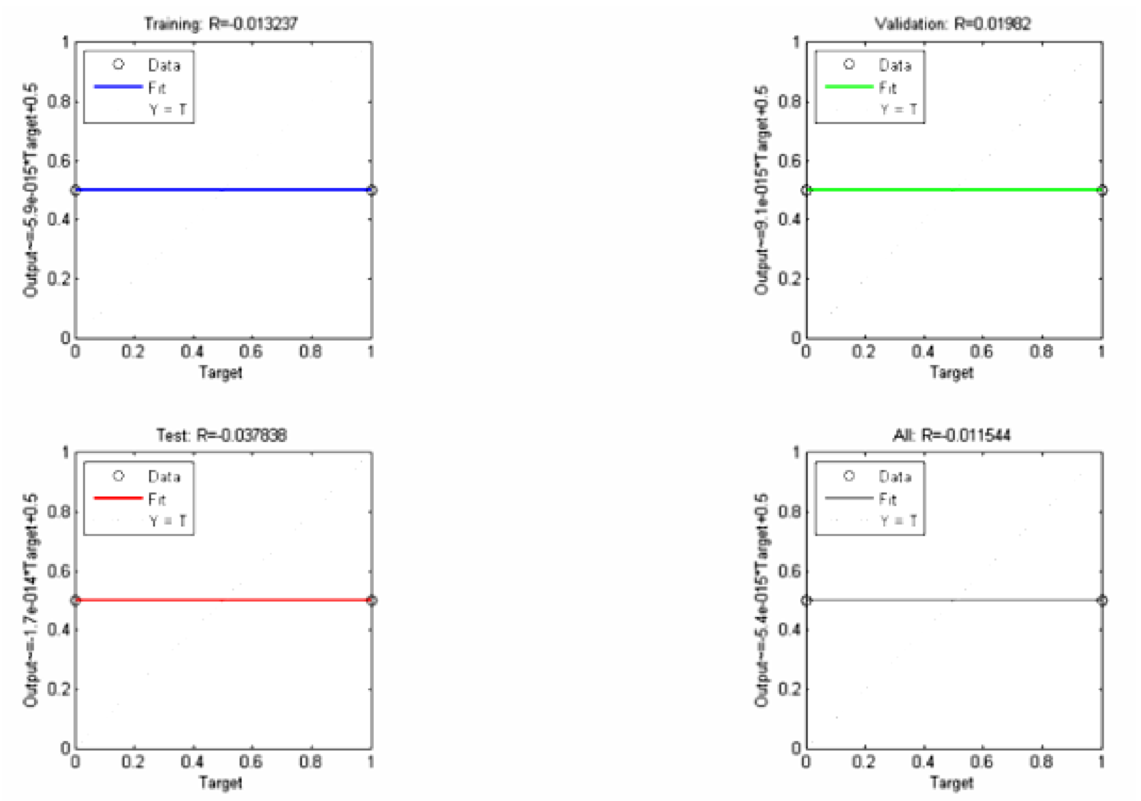
d) Scaled Conjugate Gradient



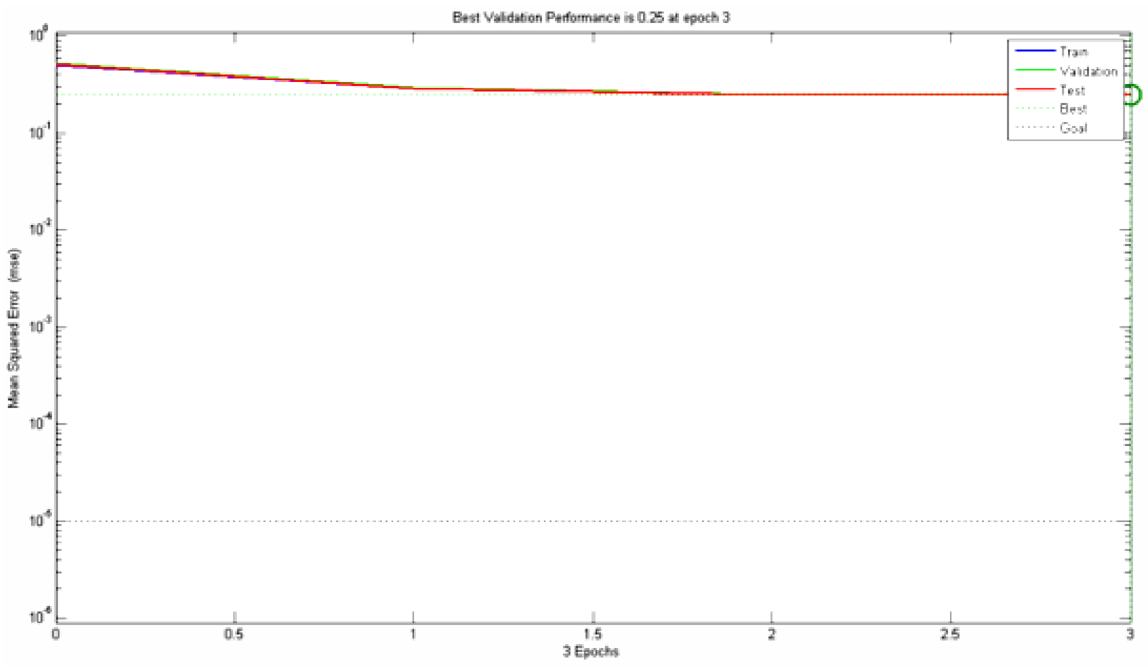


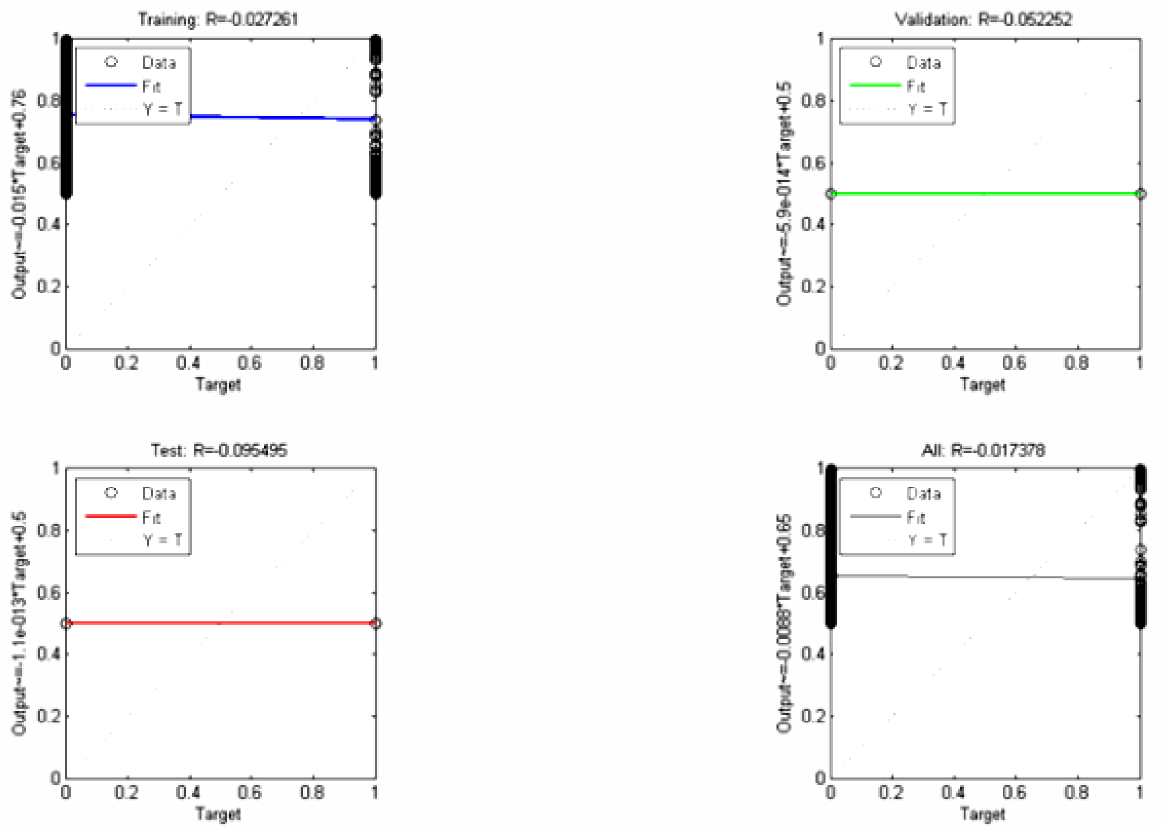
e) Variable Learning Rate Backpropagation





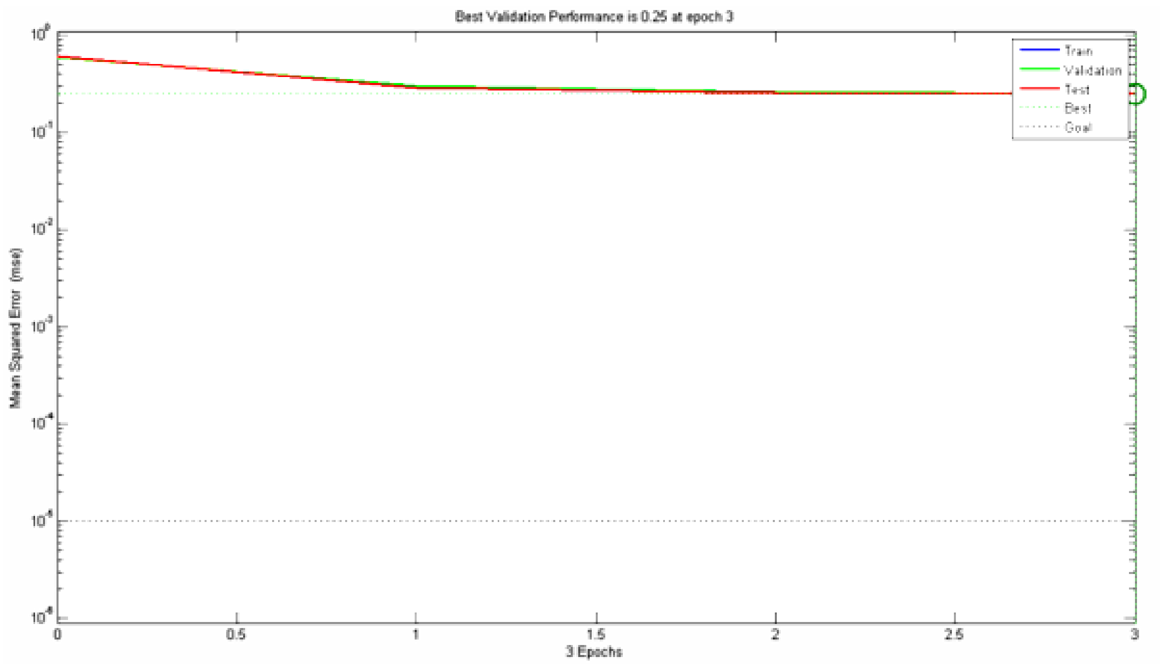
f) BFGS quasi–Newton Backpropagation

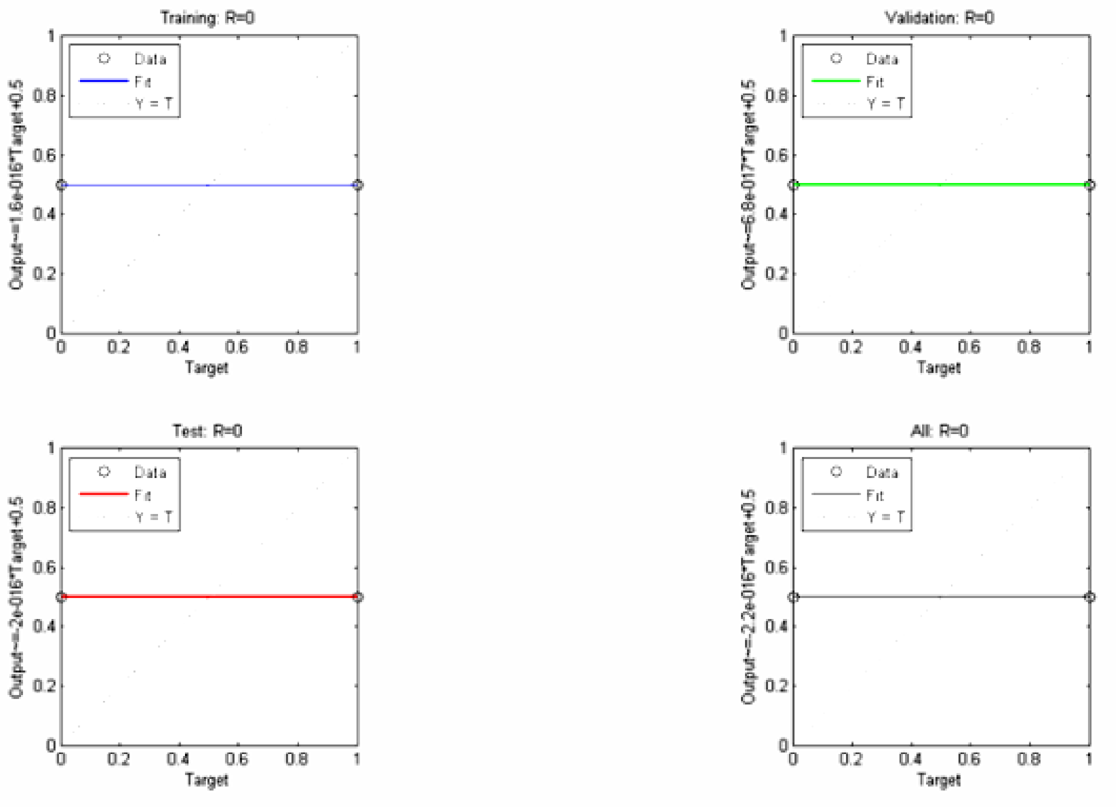




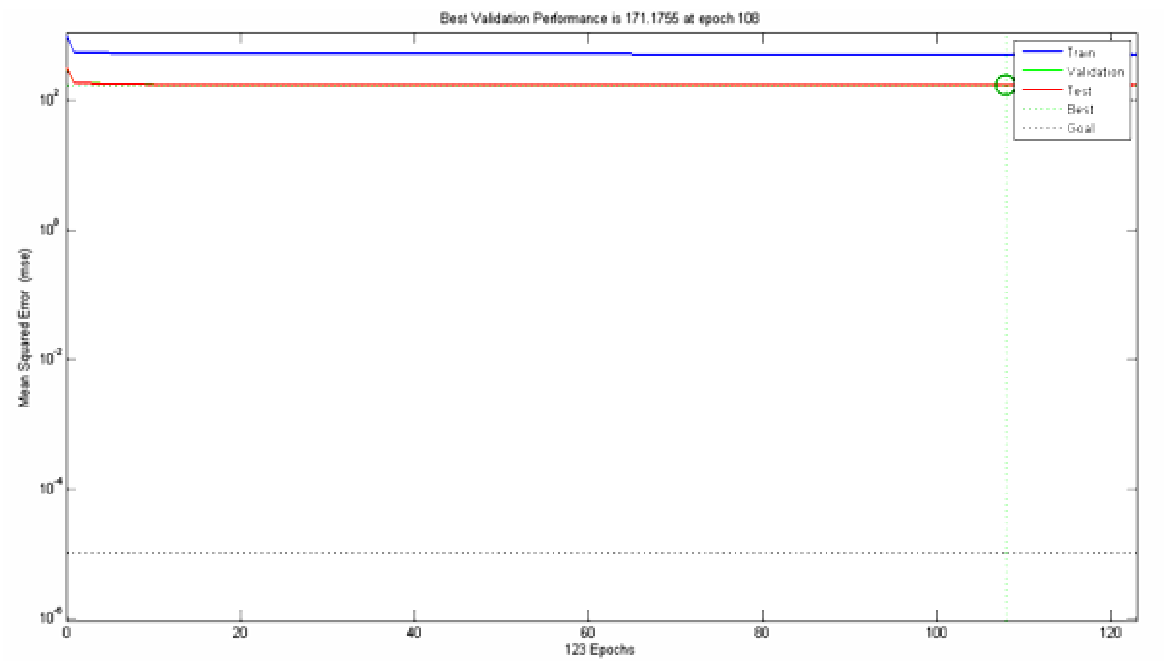
2) [2]

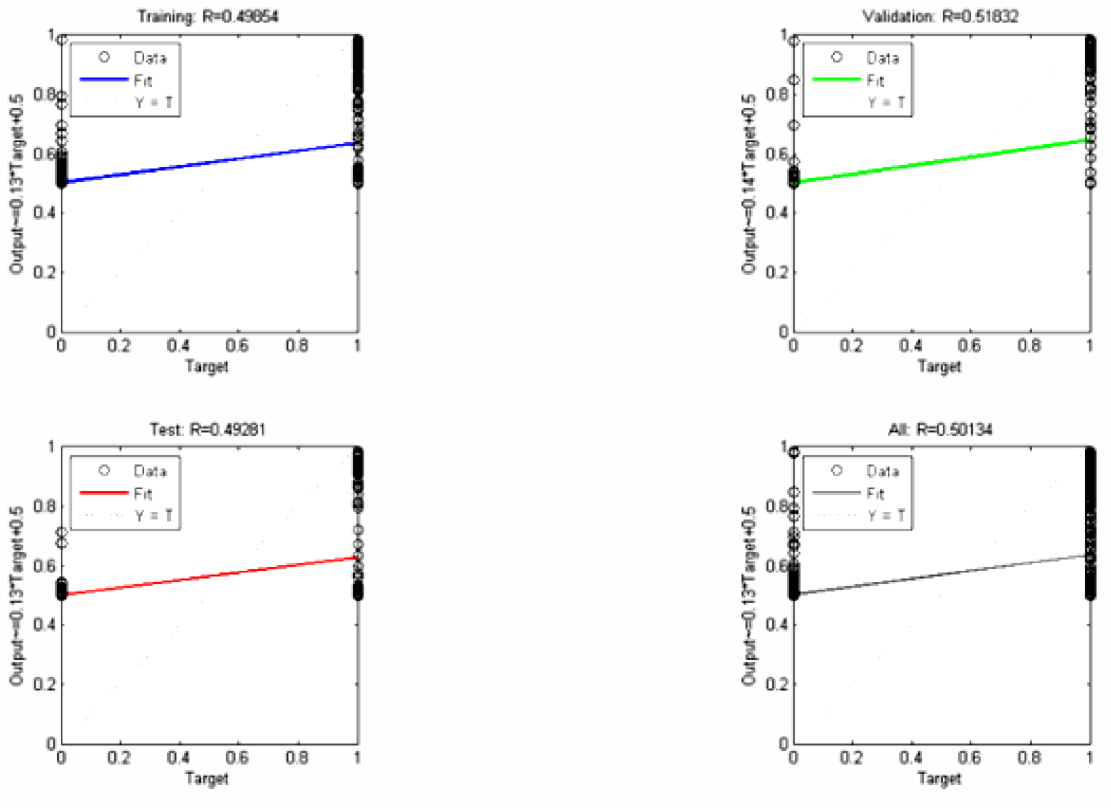
a) Levenberg–Marquardt



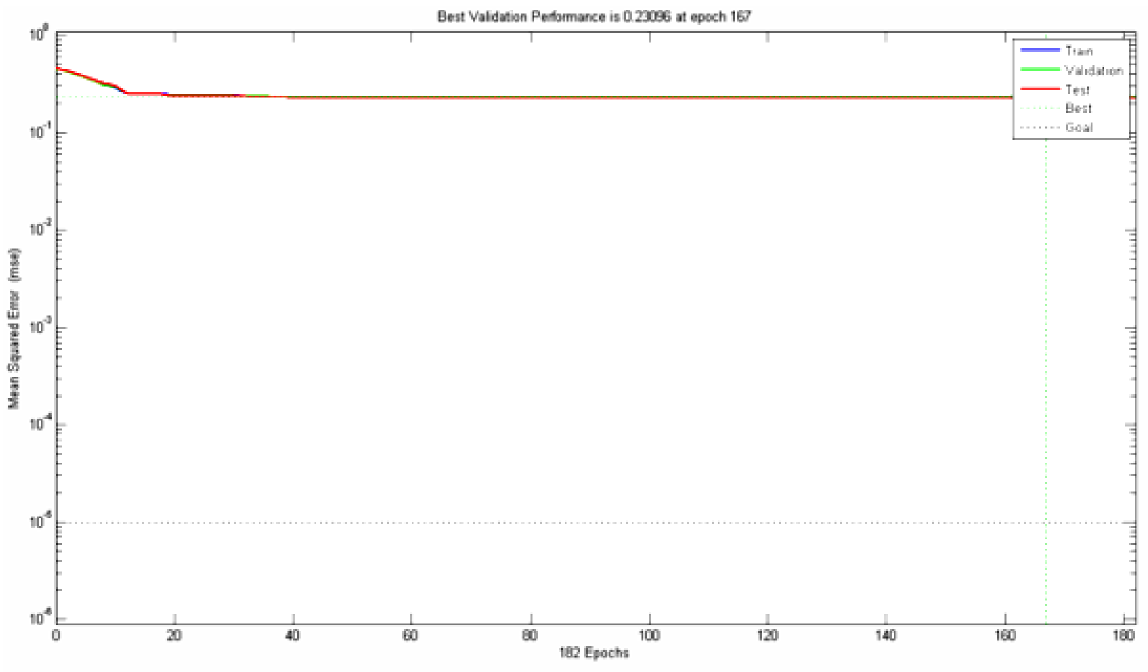


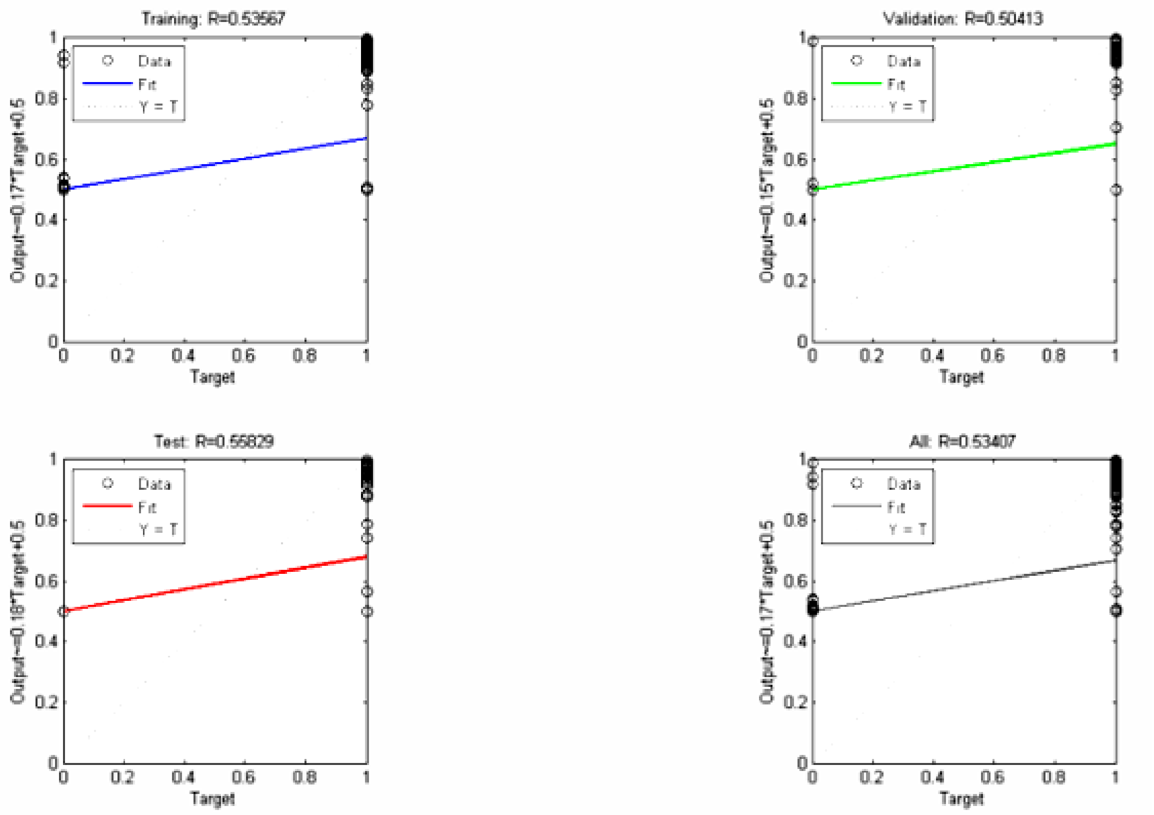
b) Bayesian Regularisation



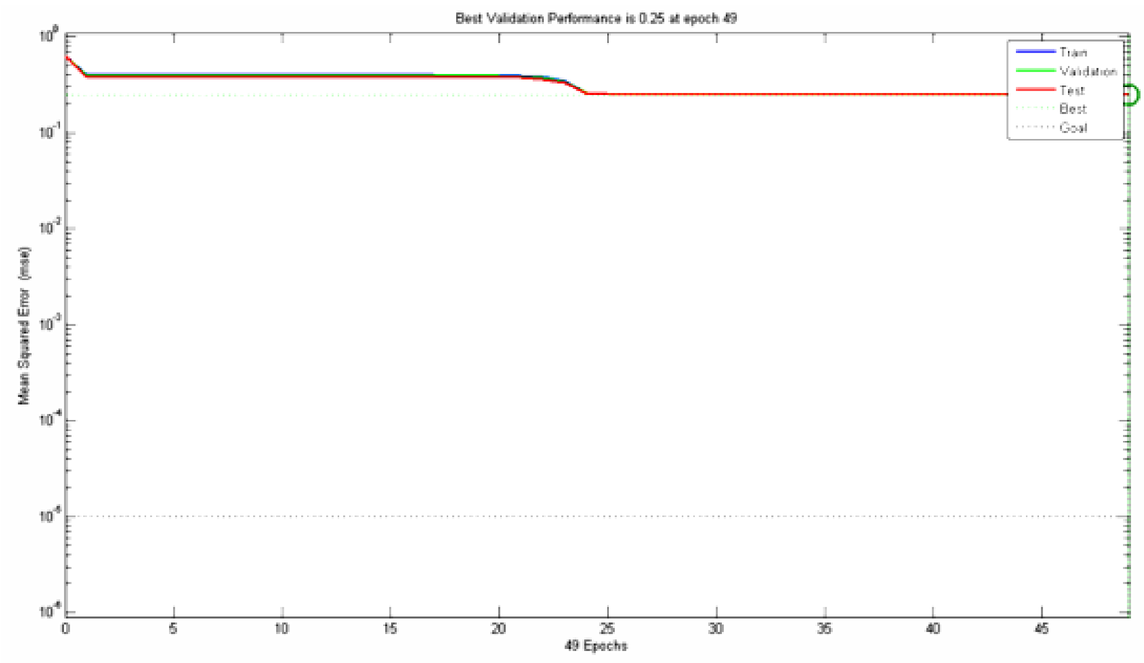


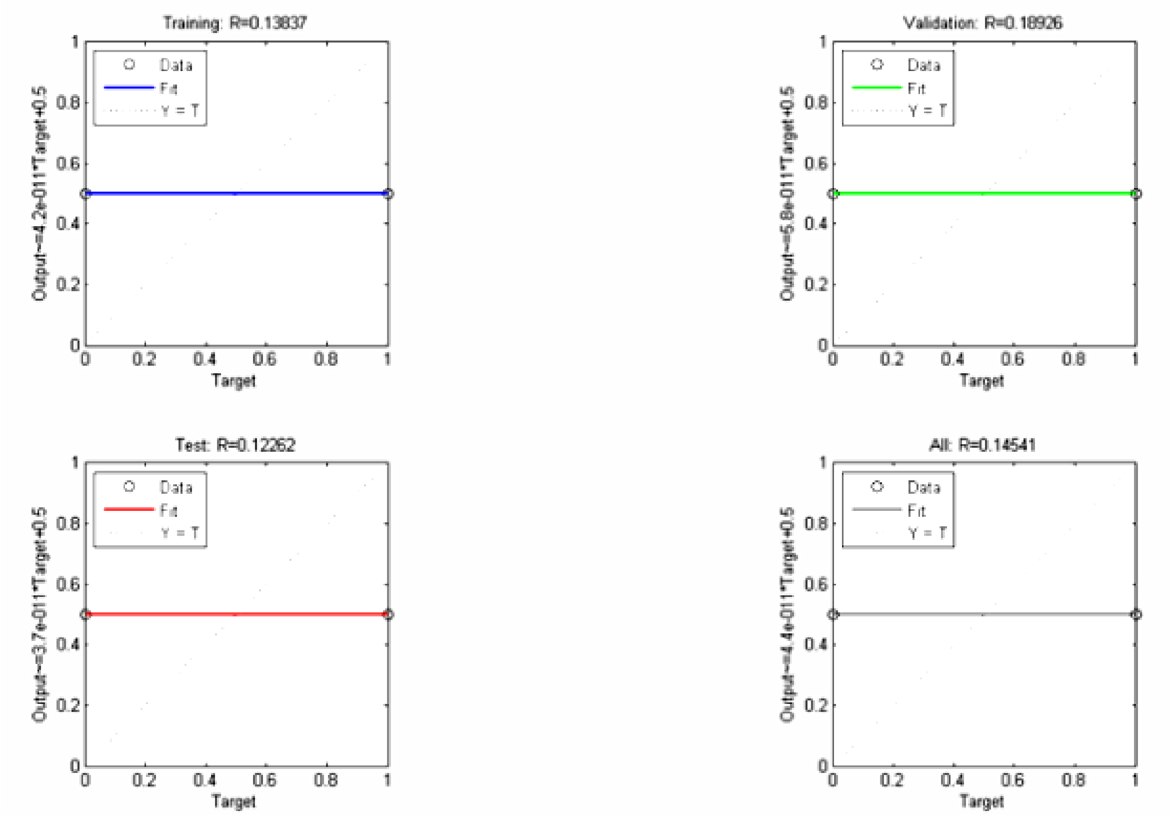
c) Resilient Backpropagation



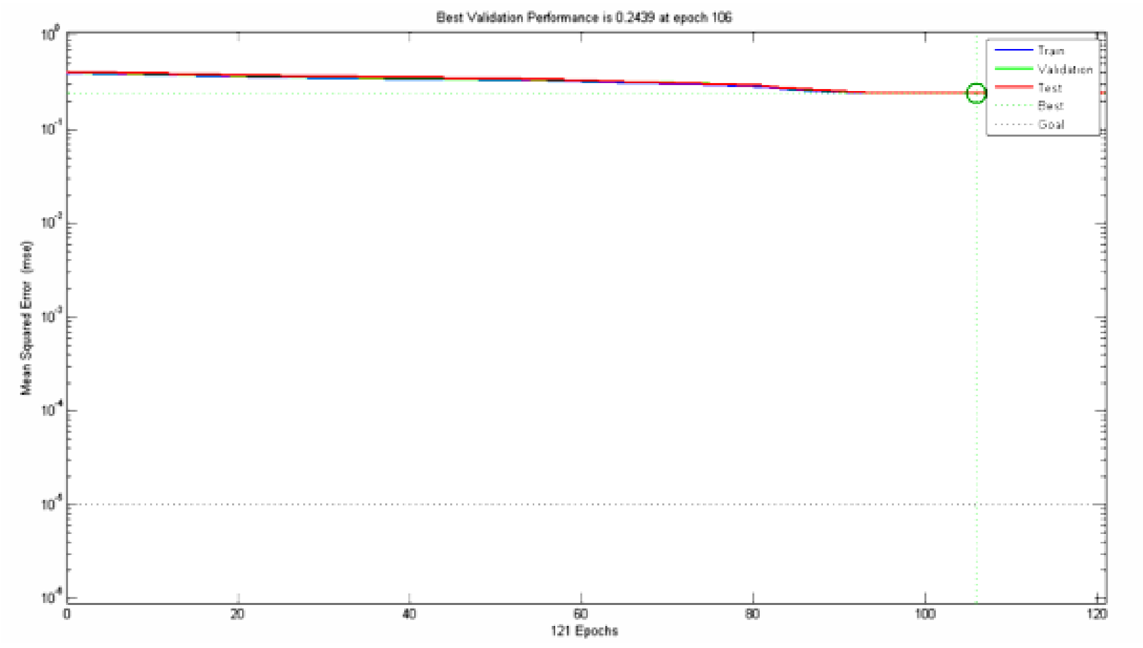


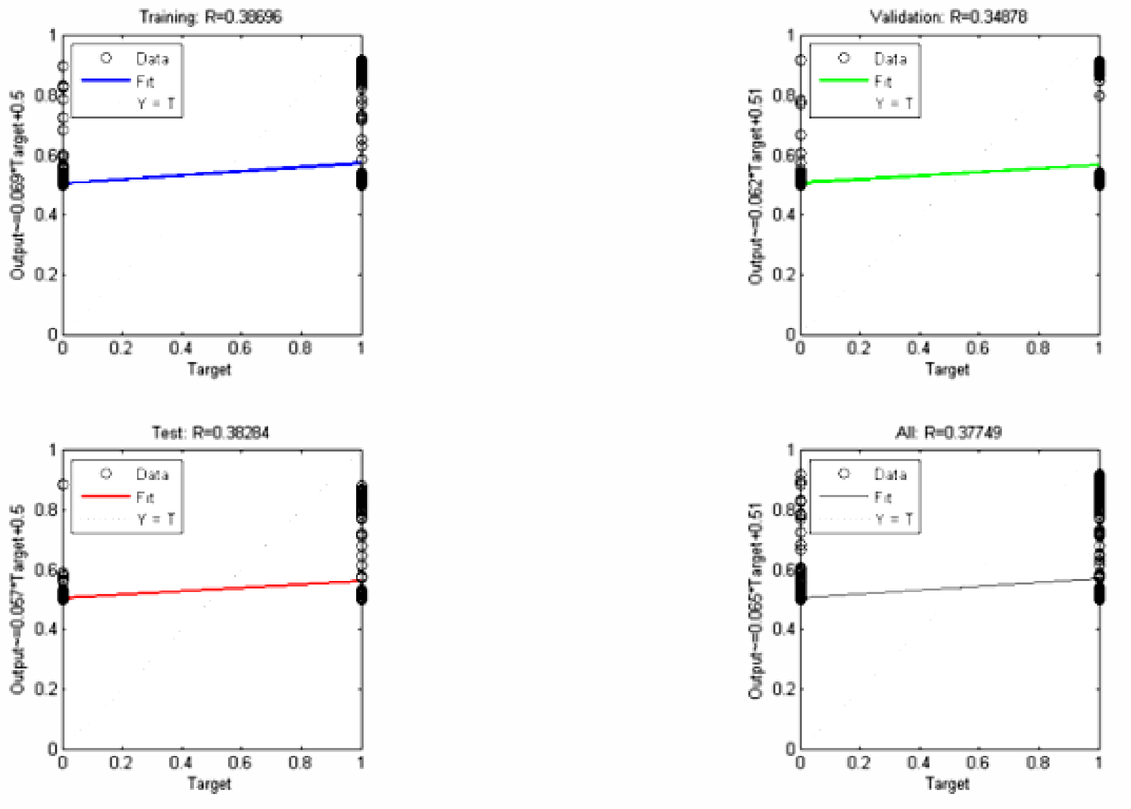
d) Scaled Conjugate Gradient



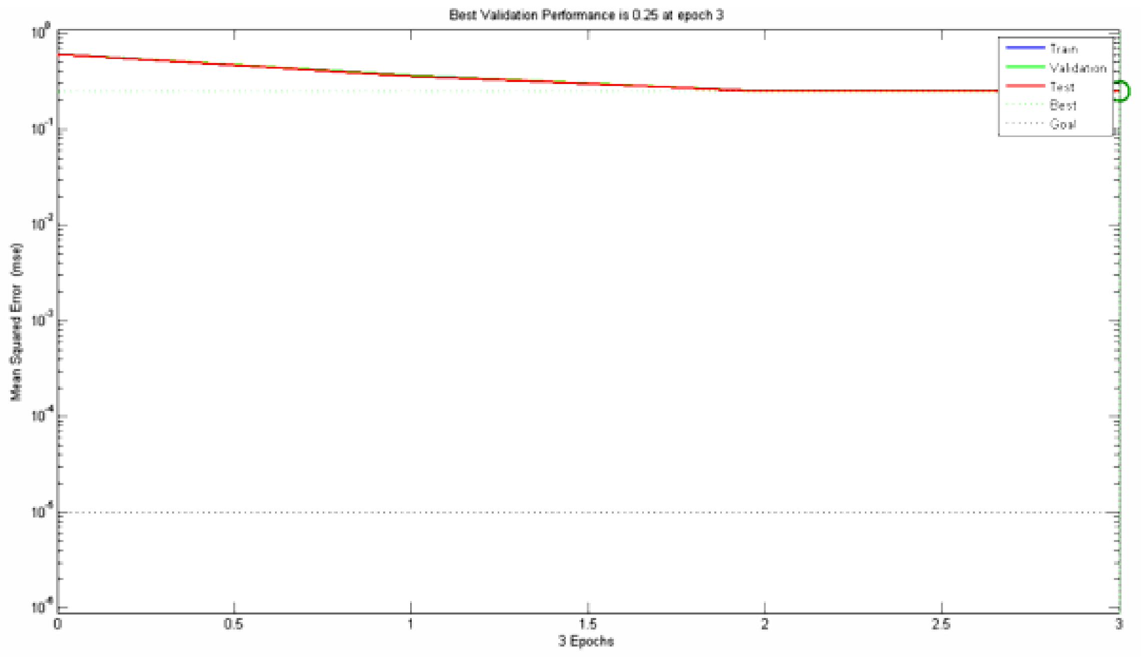


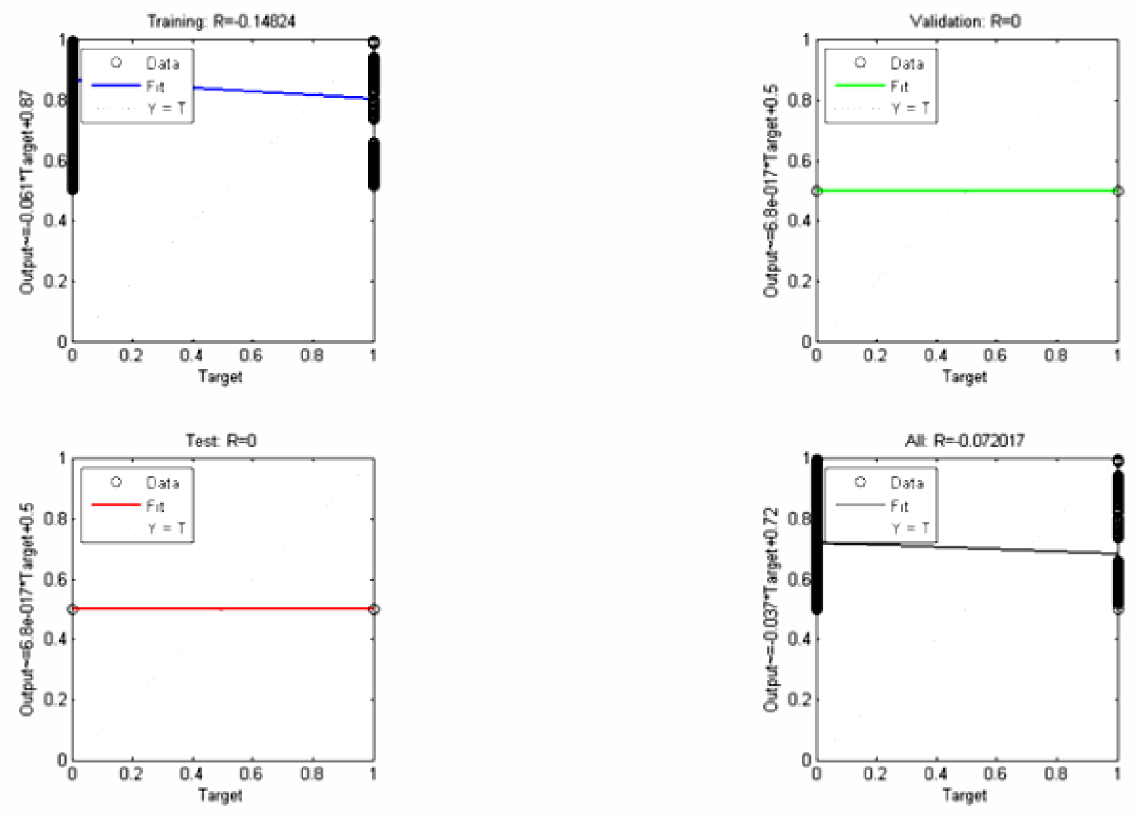
e) Variable Learning Rate Backpropagation





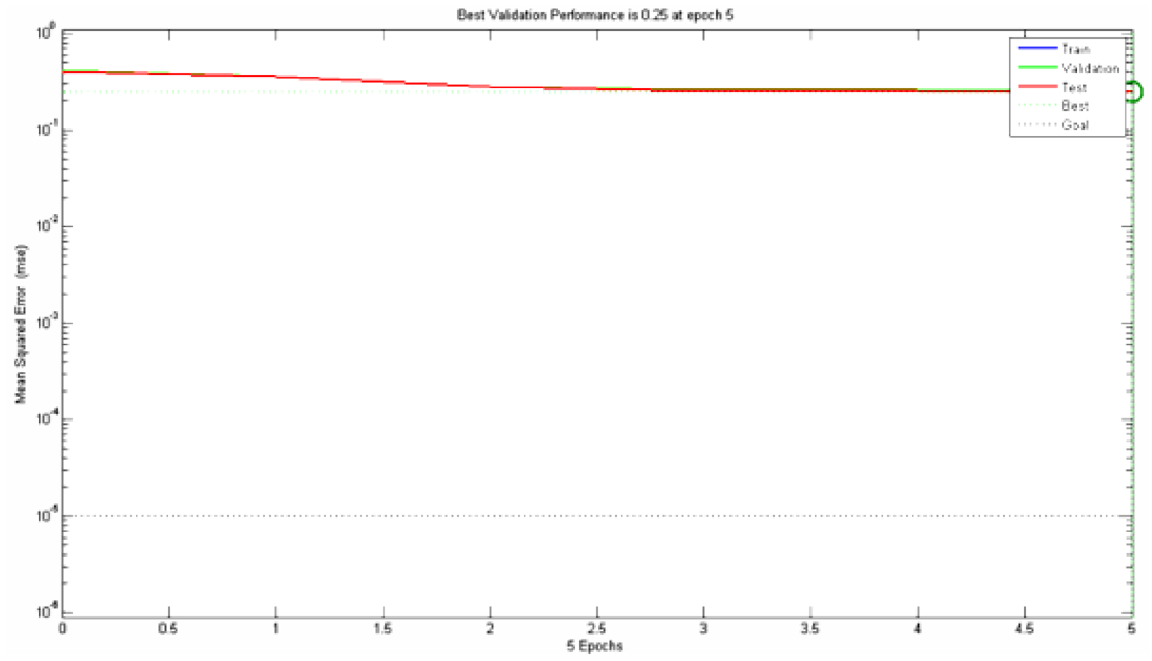
f) BFGS quasi–Newton Backpropagation

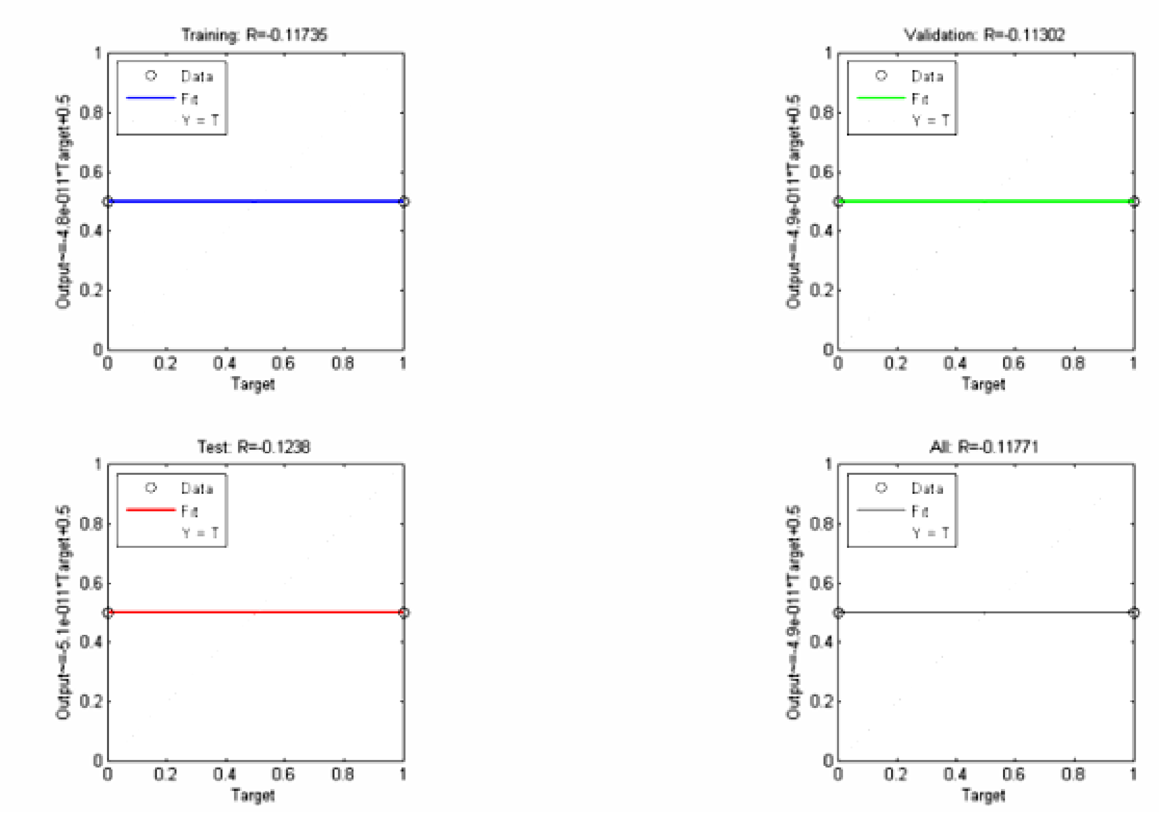




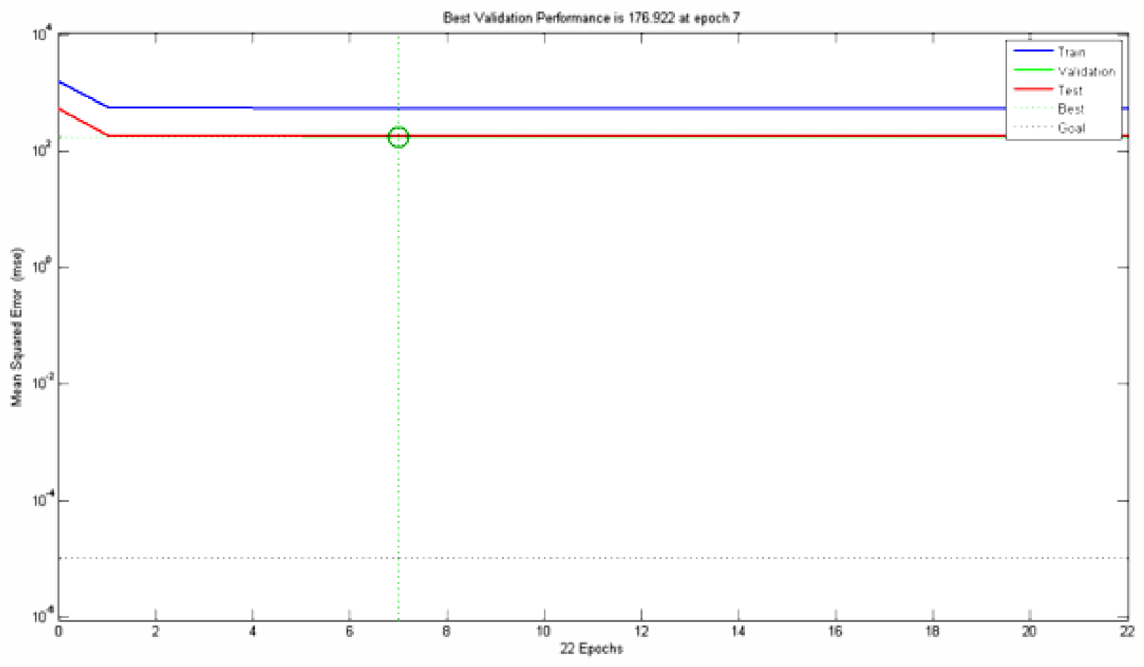
3) [3]

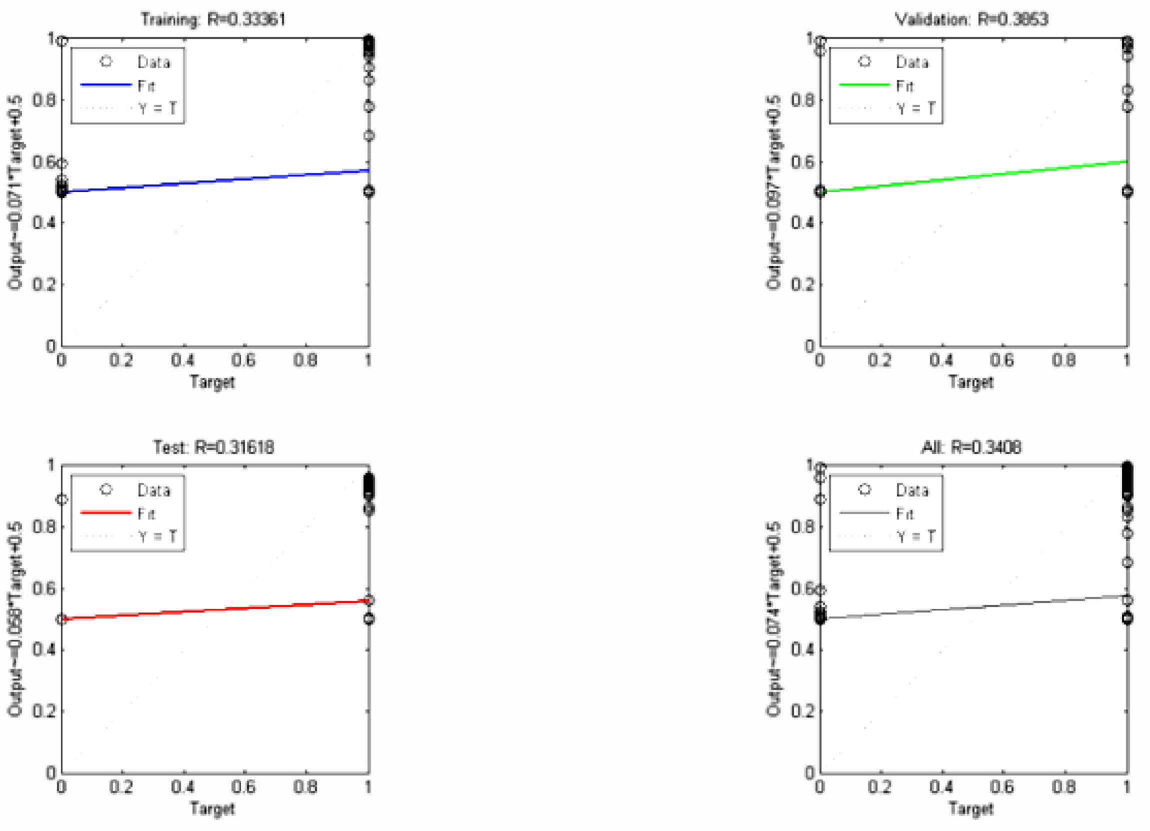
a) l Levenberg–Marquardt



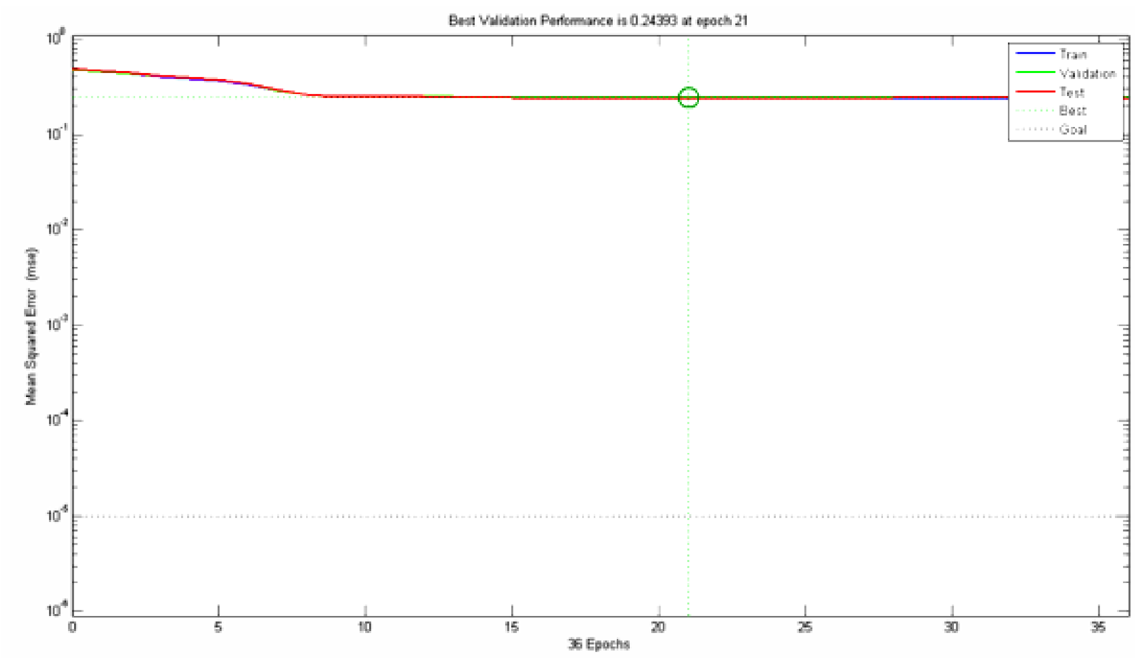


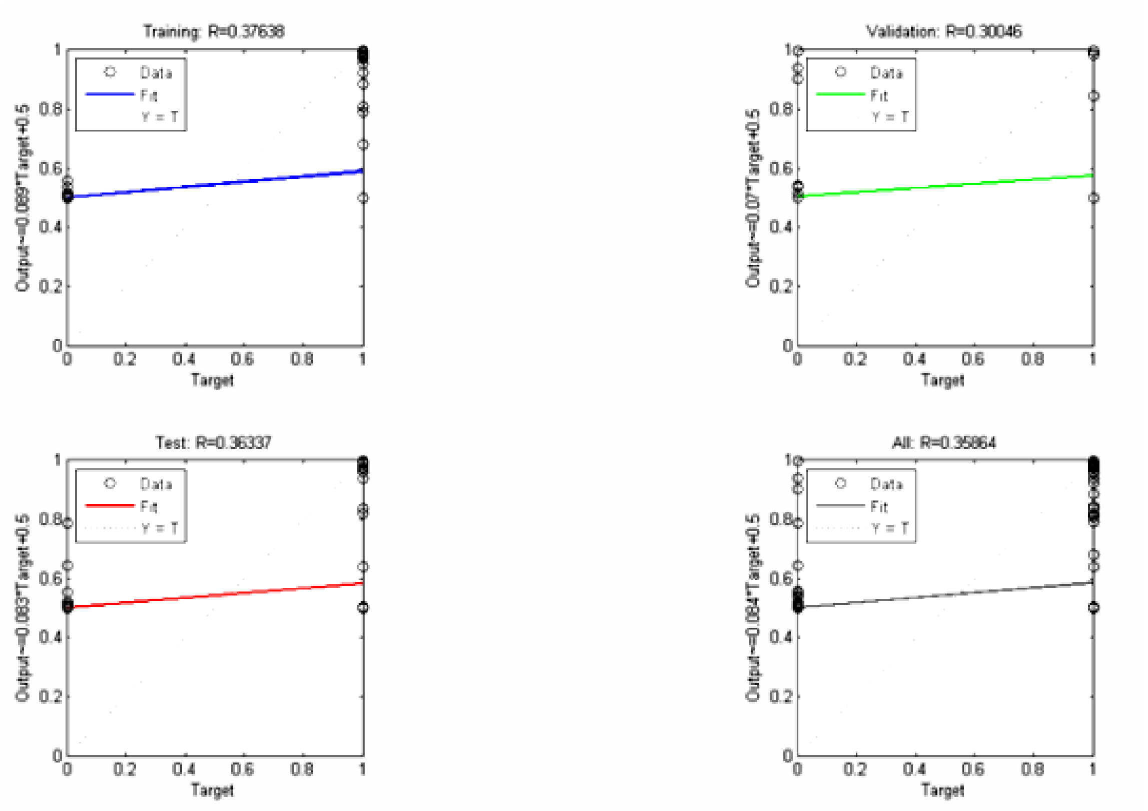
b) Bayesian Regularisation



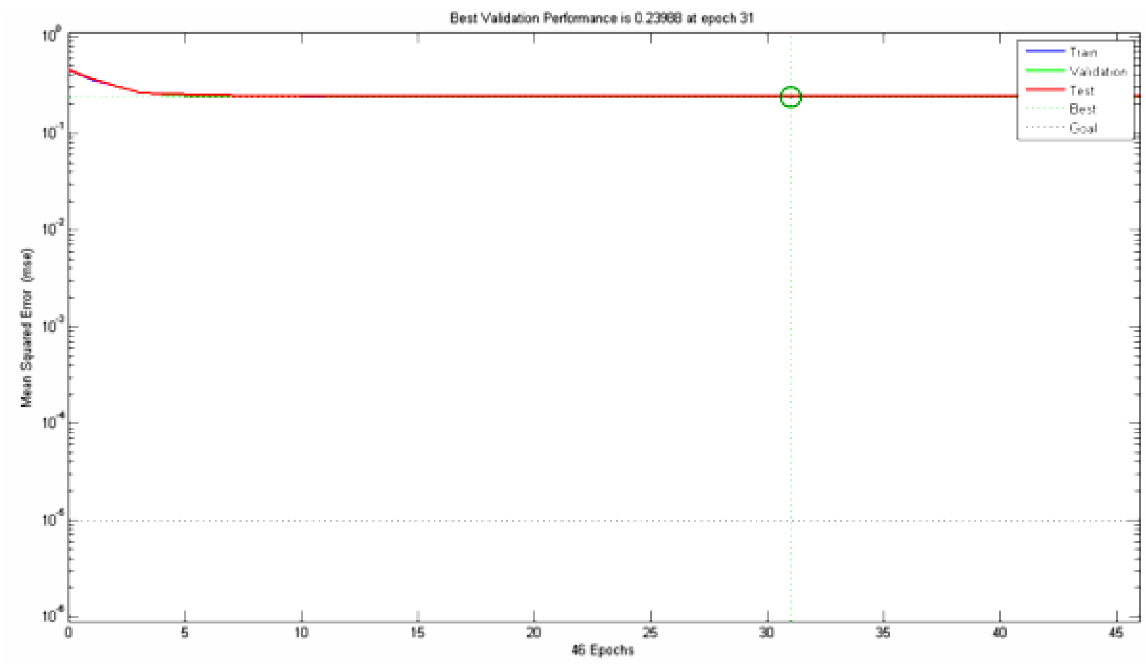


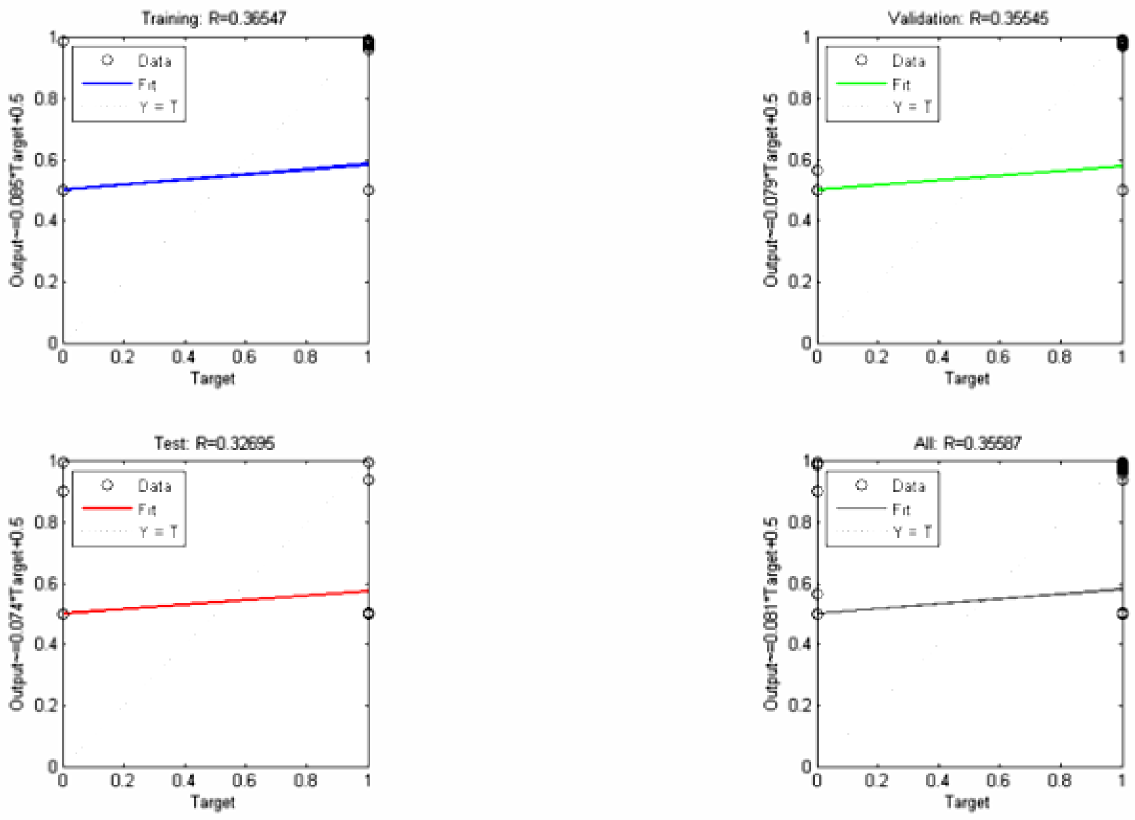
c) Resilient Backpropagation



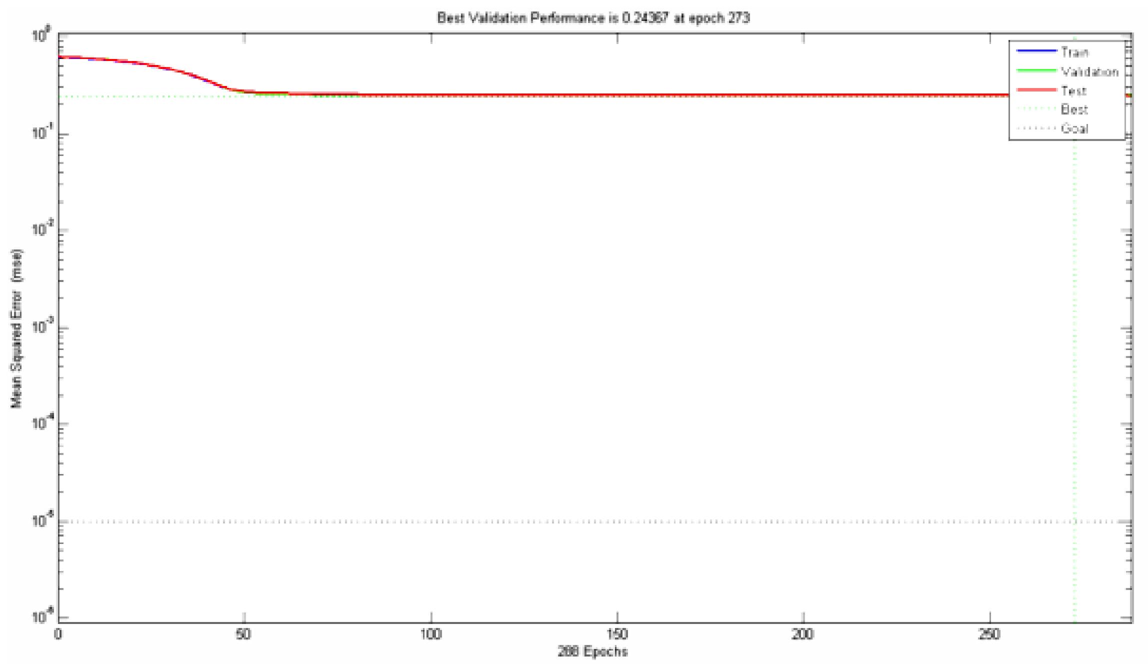


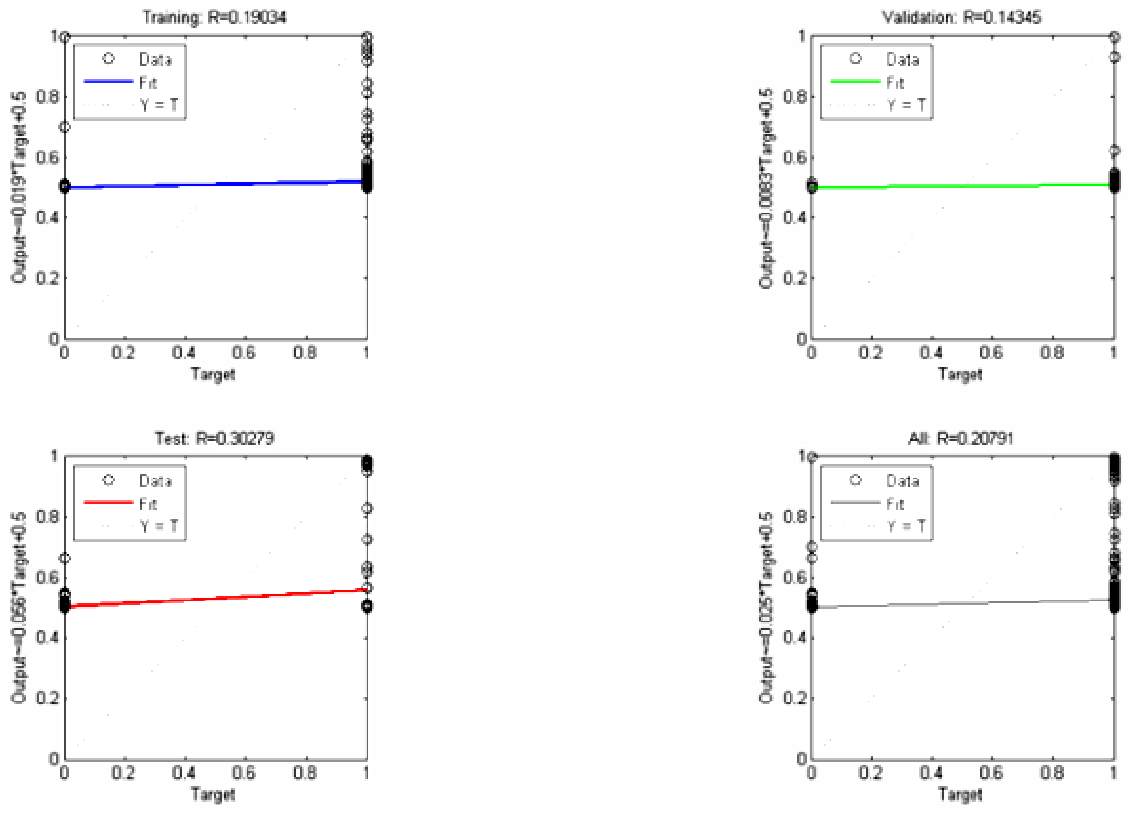
d) Scaled Conjugate Gradient



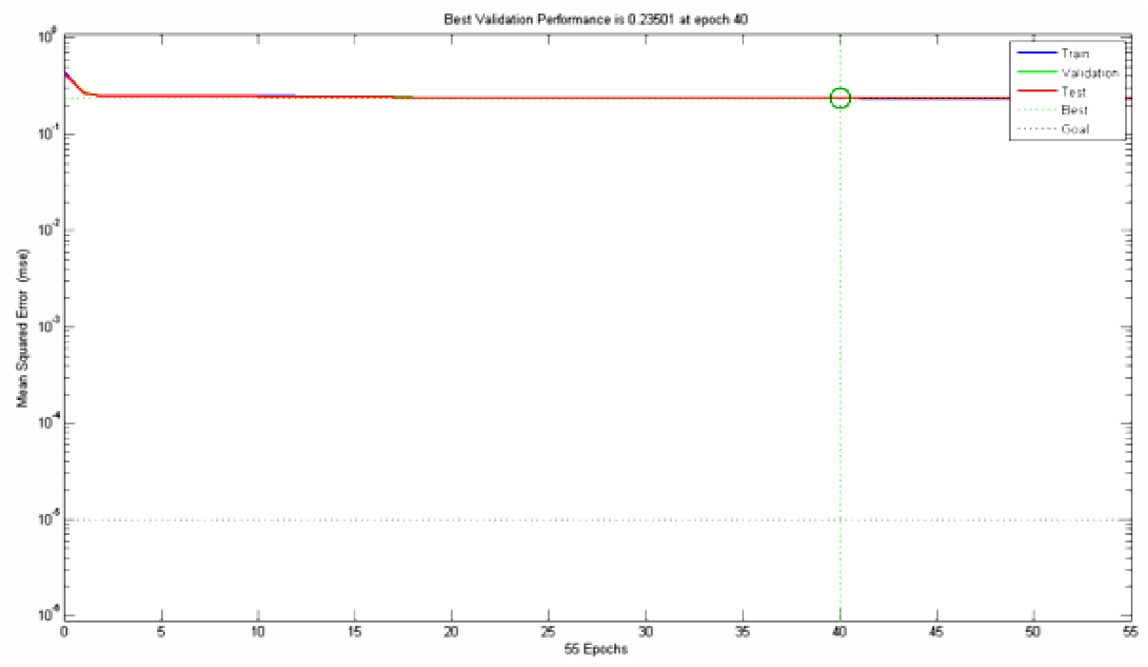


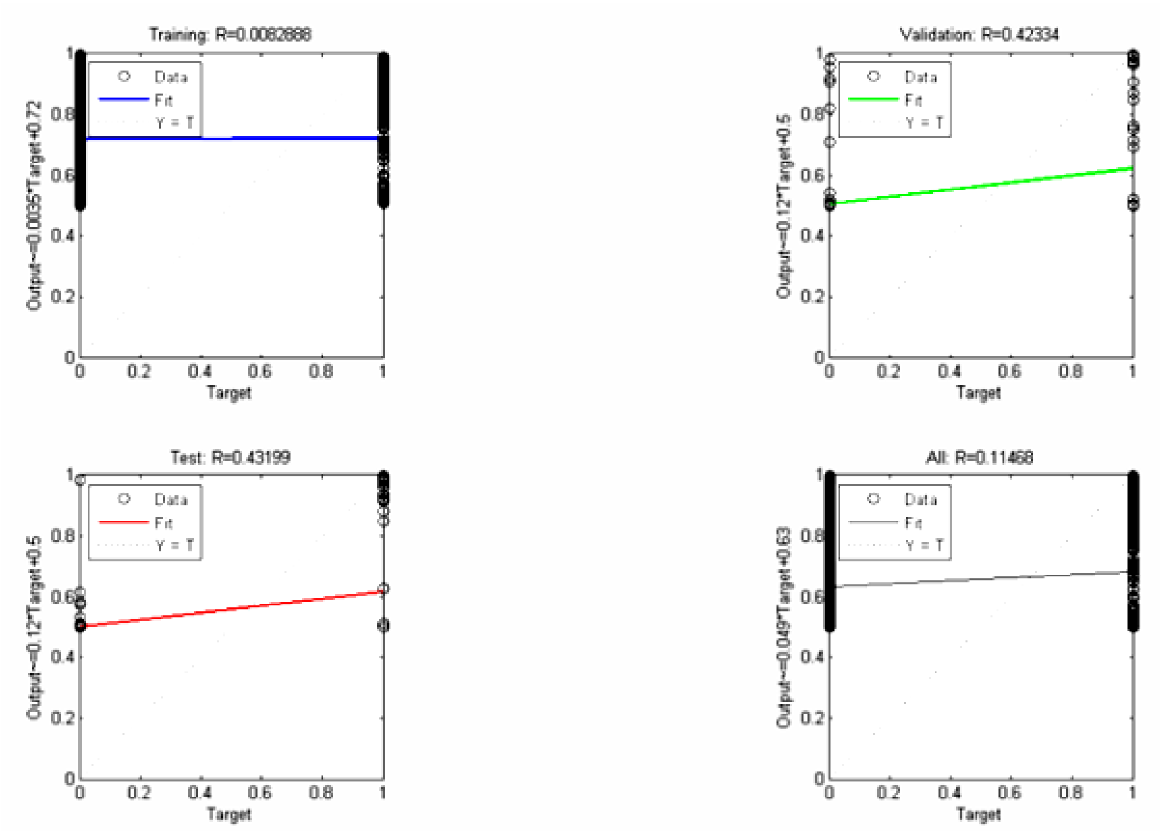
e) Variable Learning Rate Backpropagation





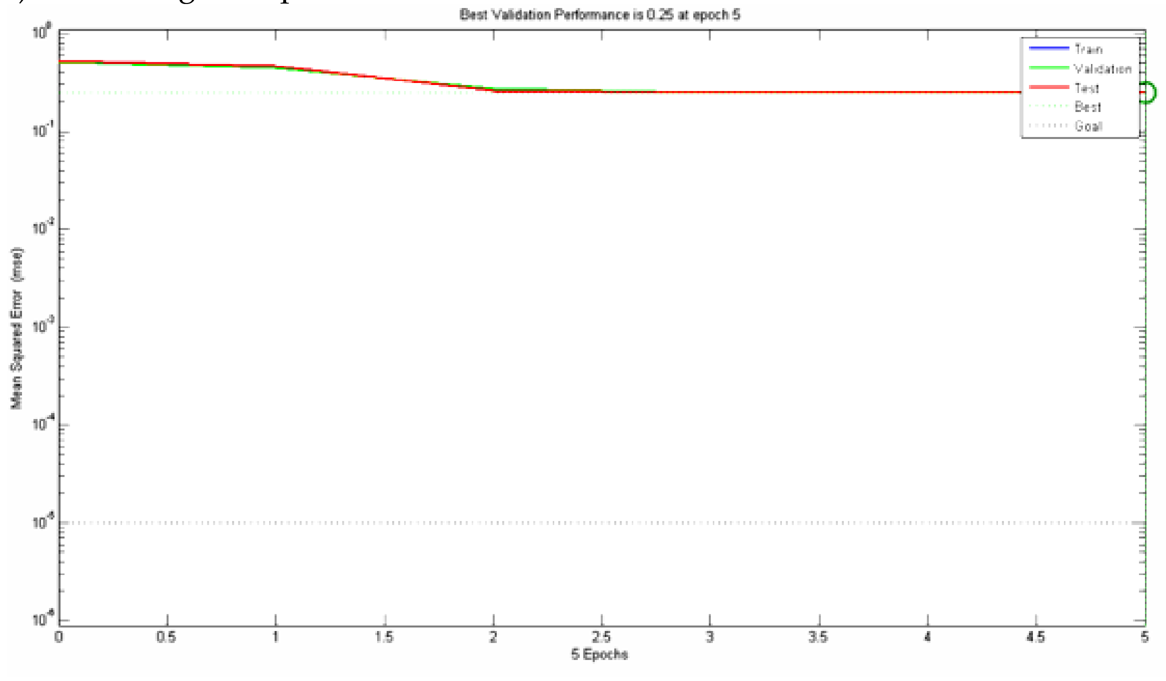
f) BFGS quasi–Newton Backpropagation

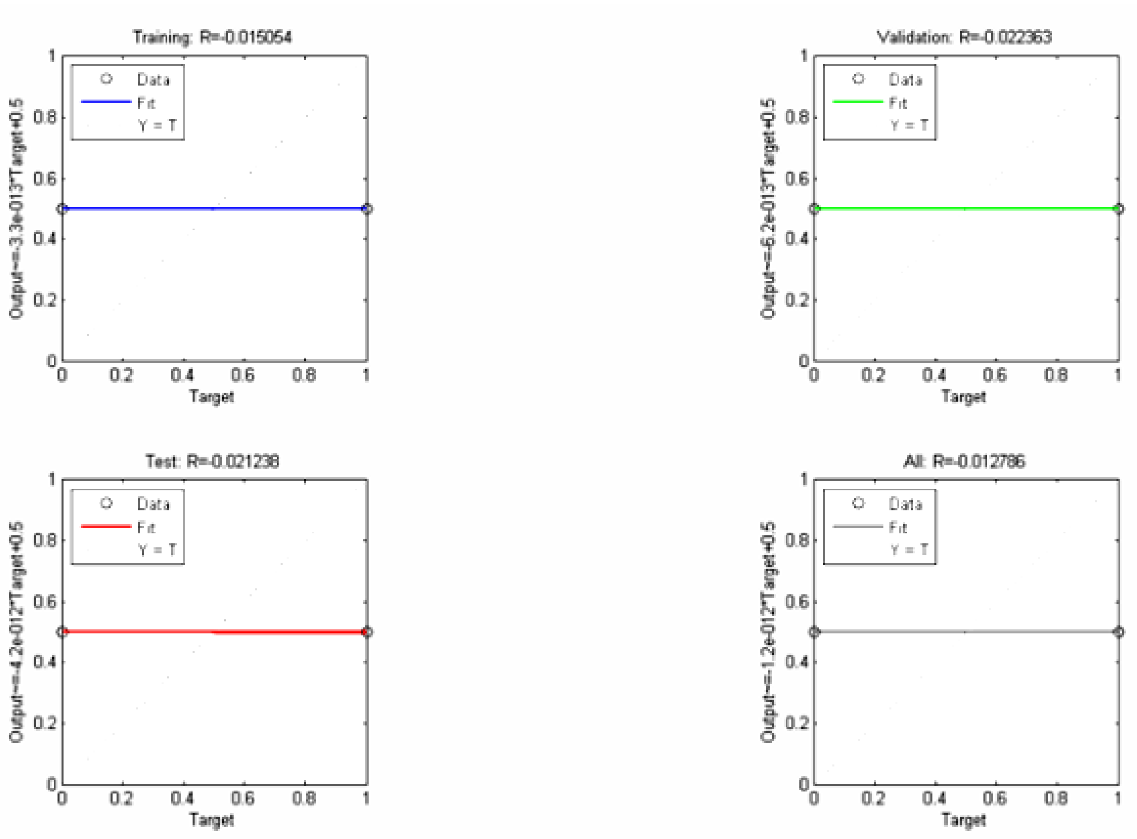




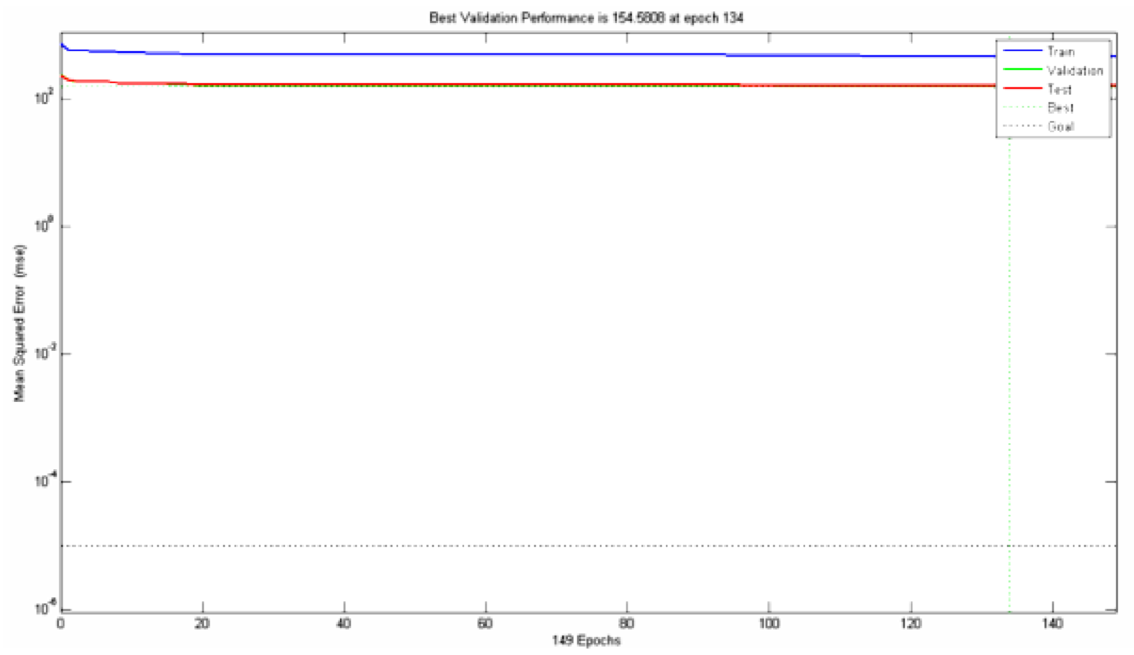
4) [5]

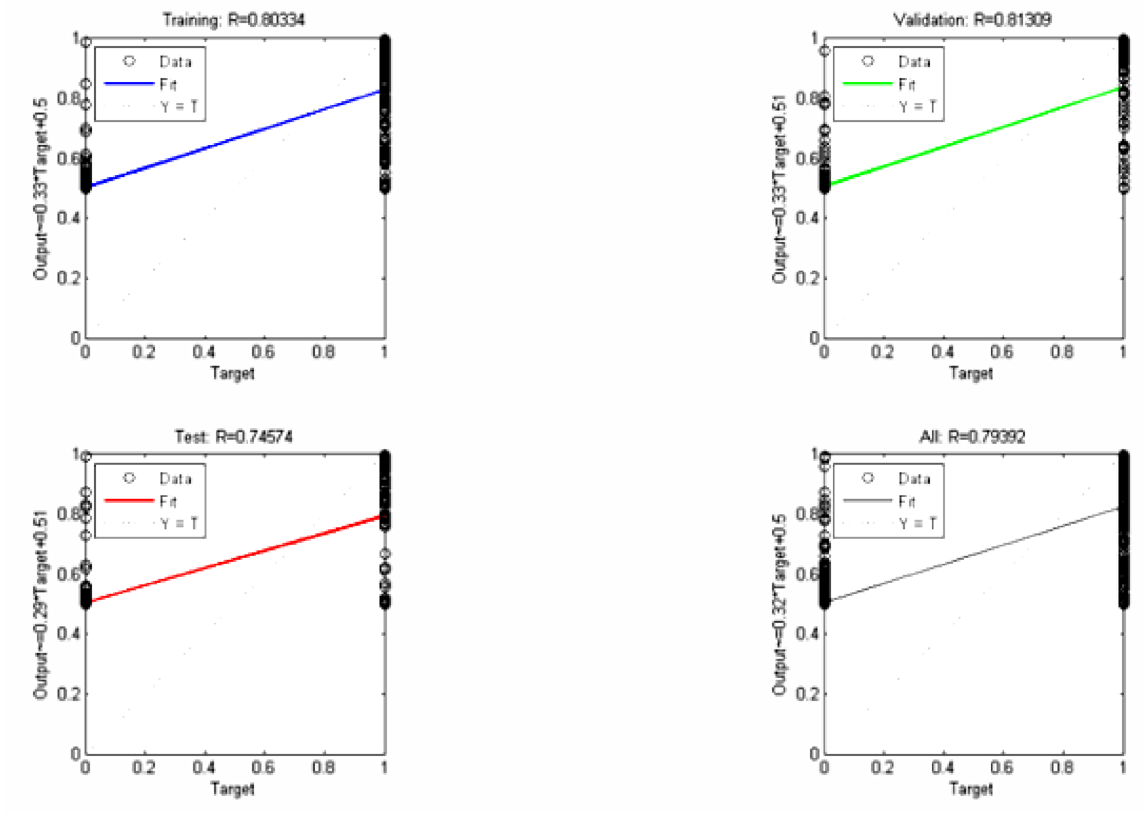
a) Levenberg–Marquardt



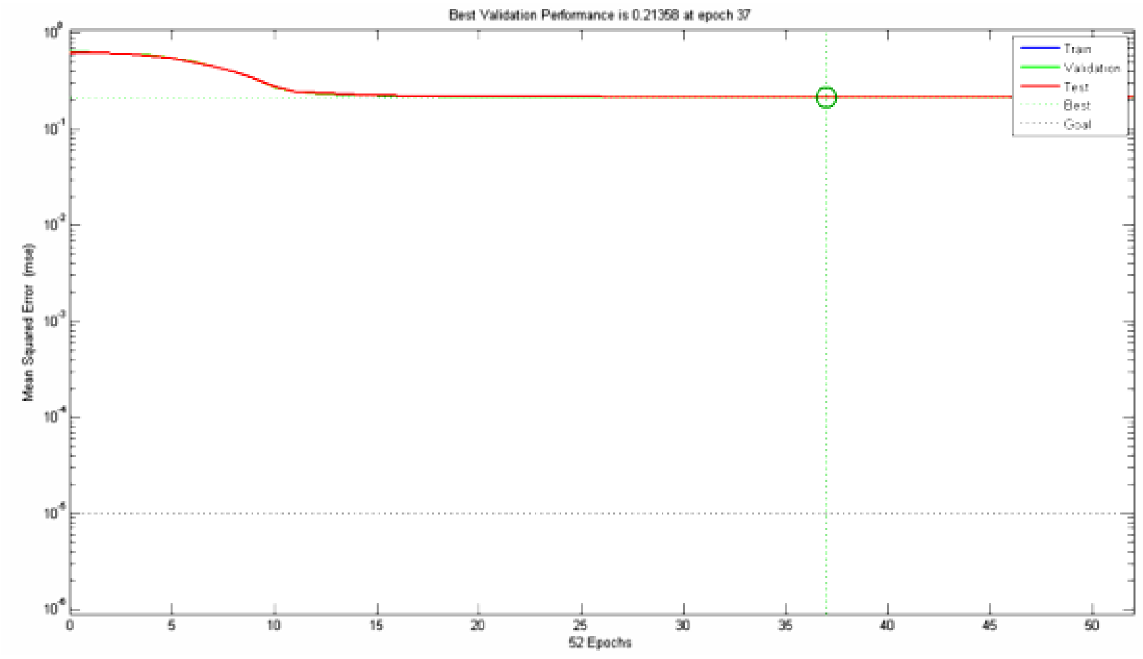


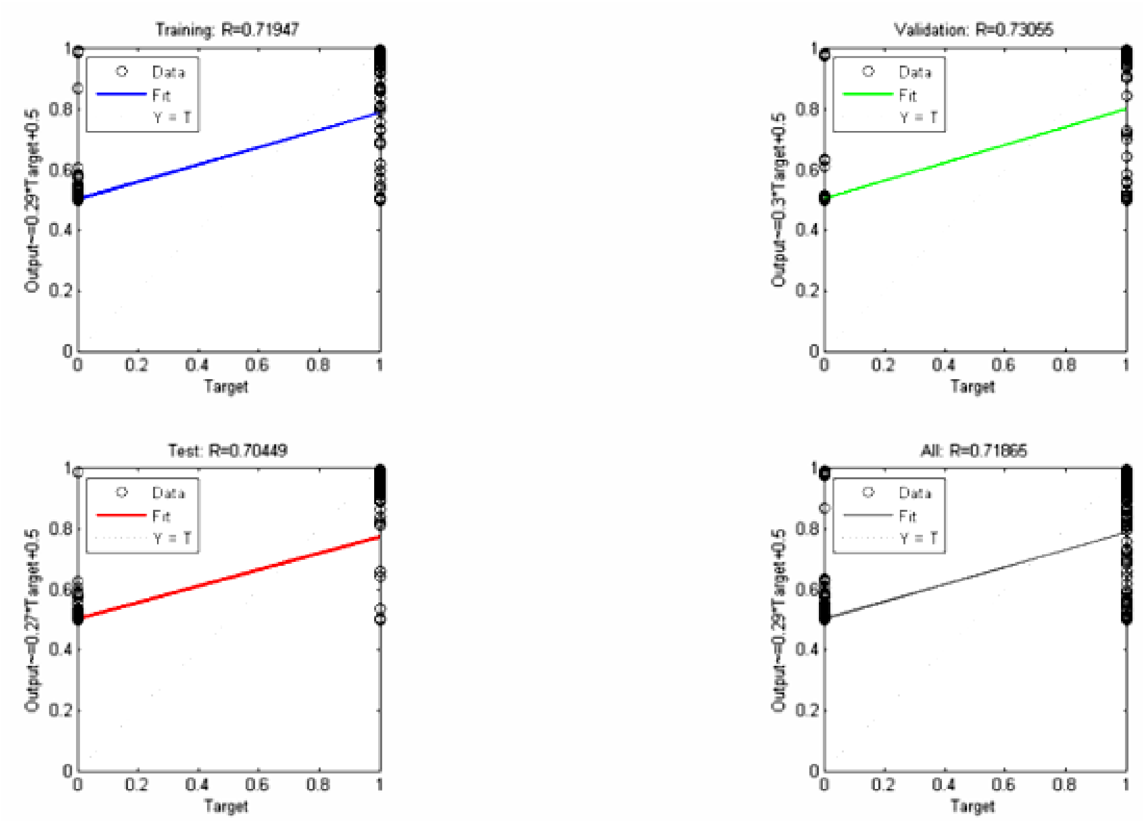
b) Bayesian Regularisation



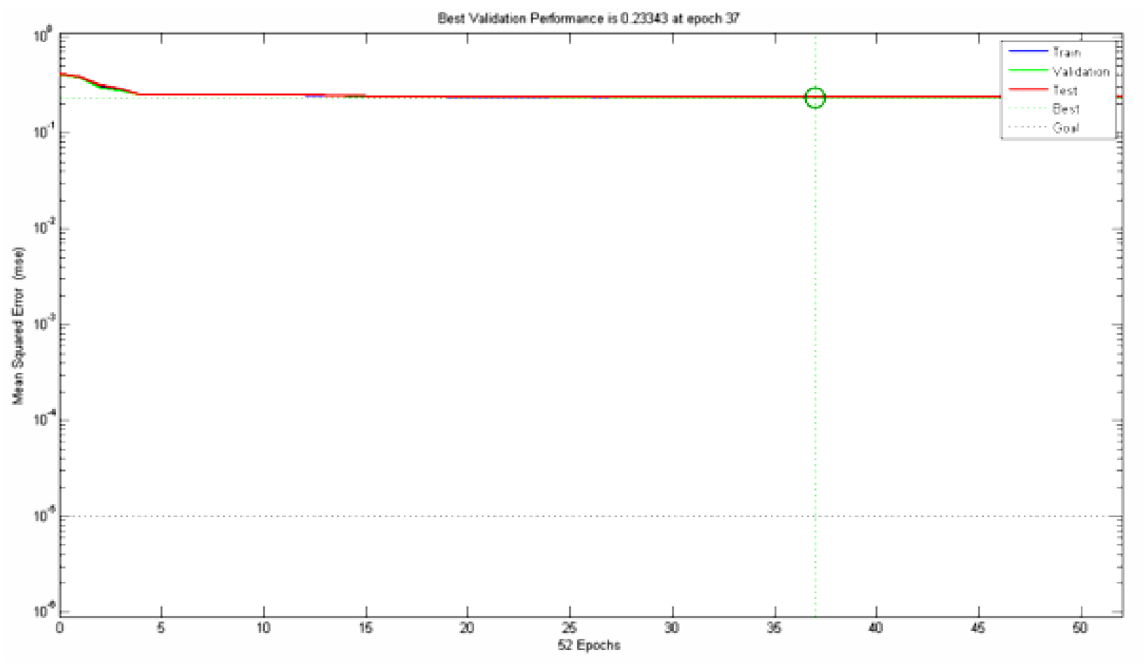


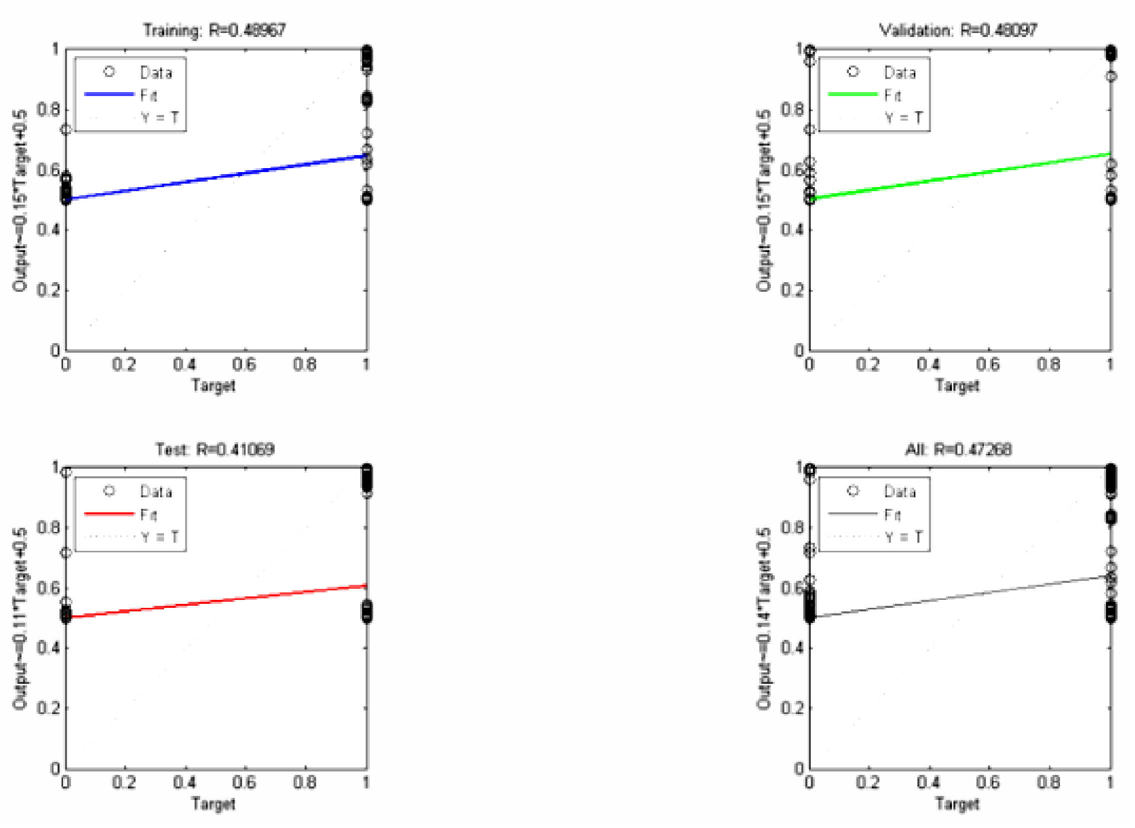
c) Resilient Backpropagation



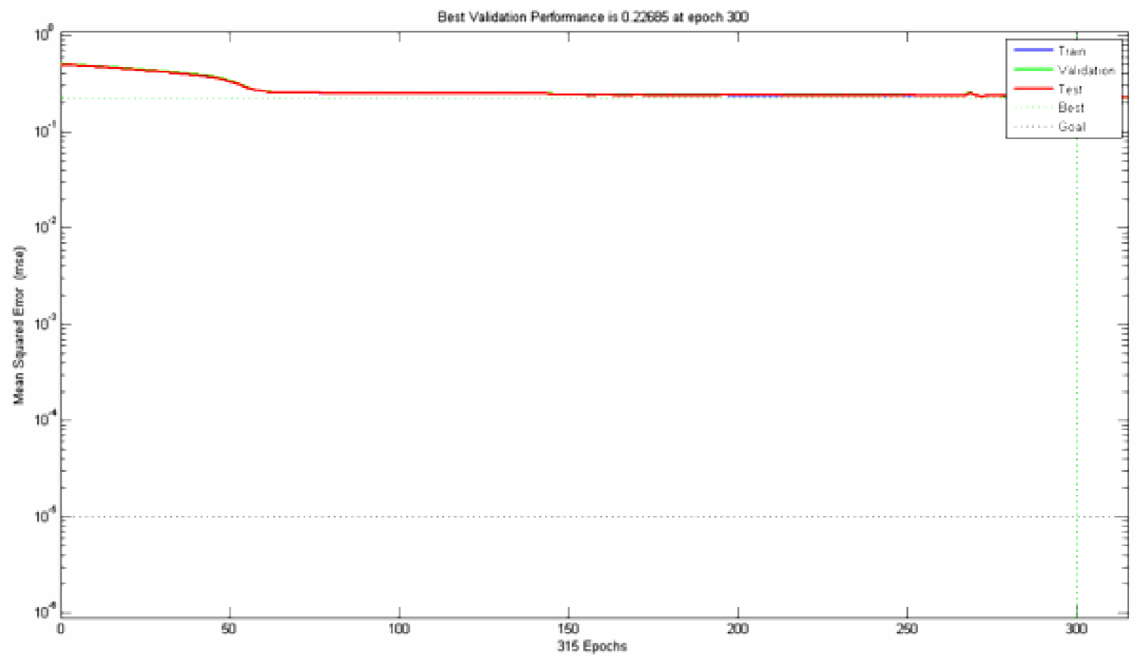


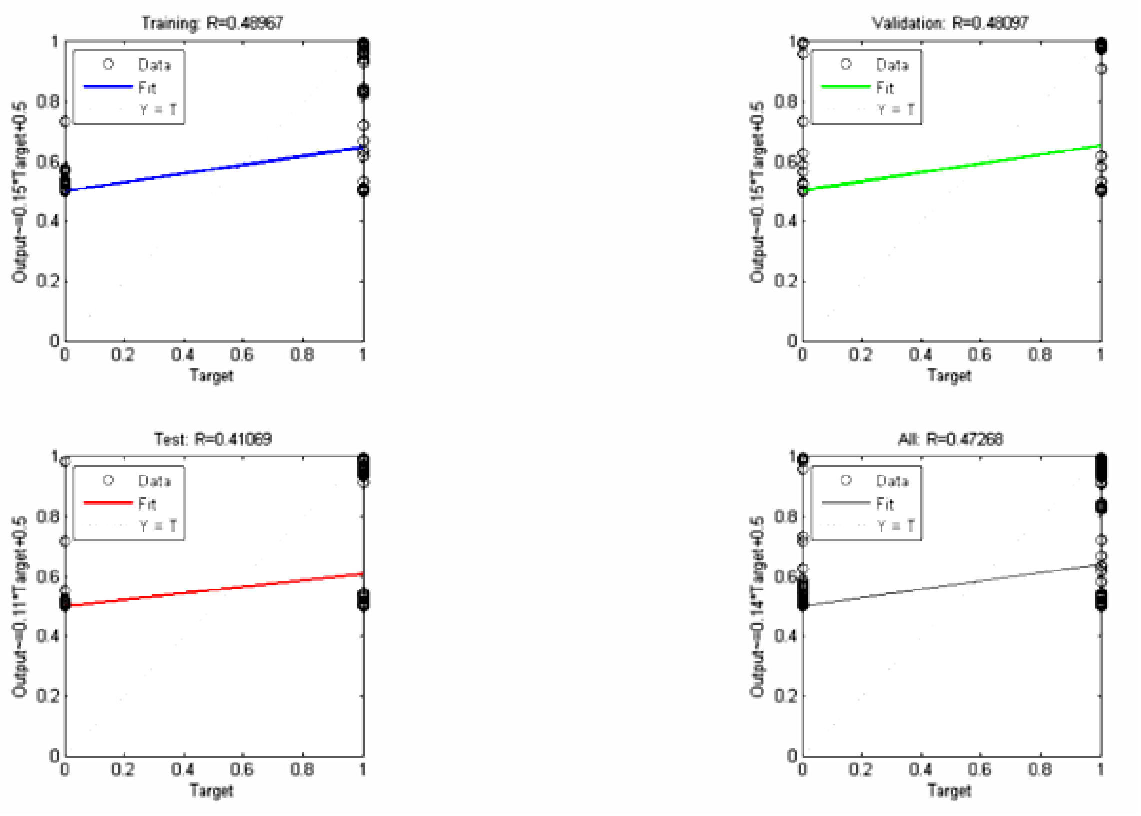
d) Scaled Conjugate Gradient



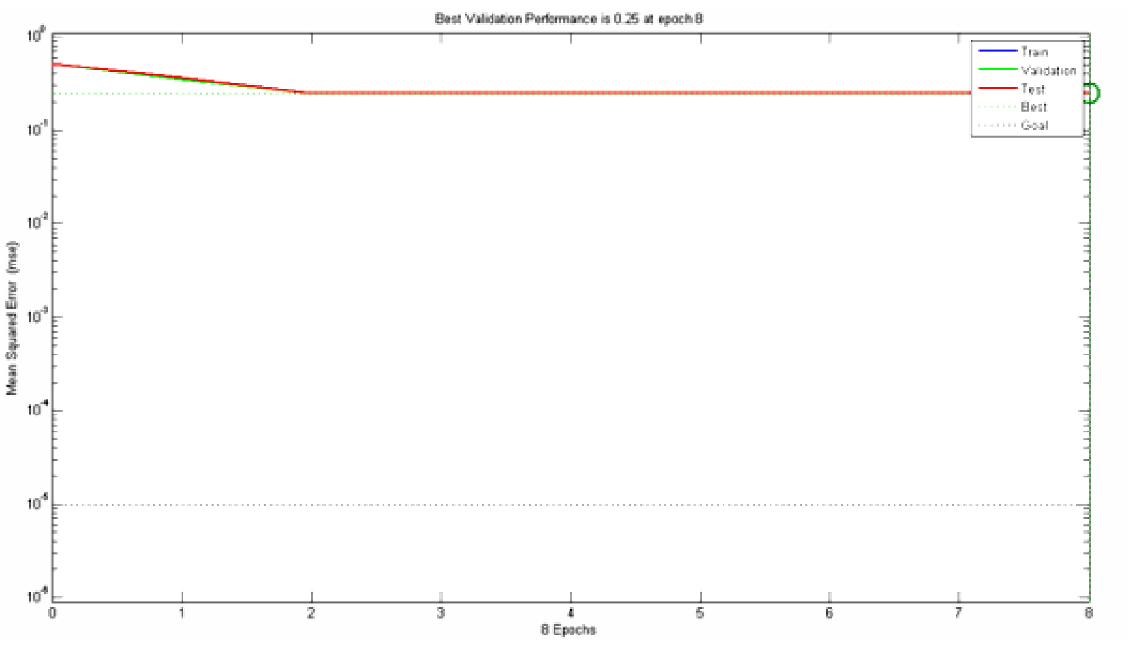


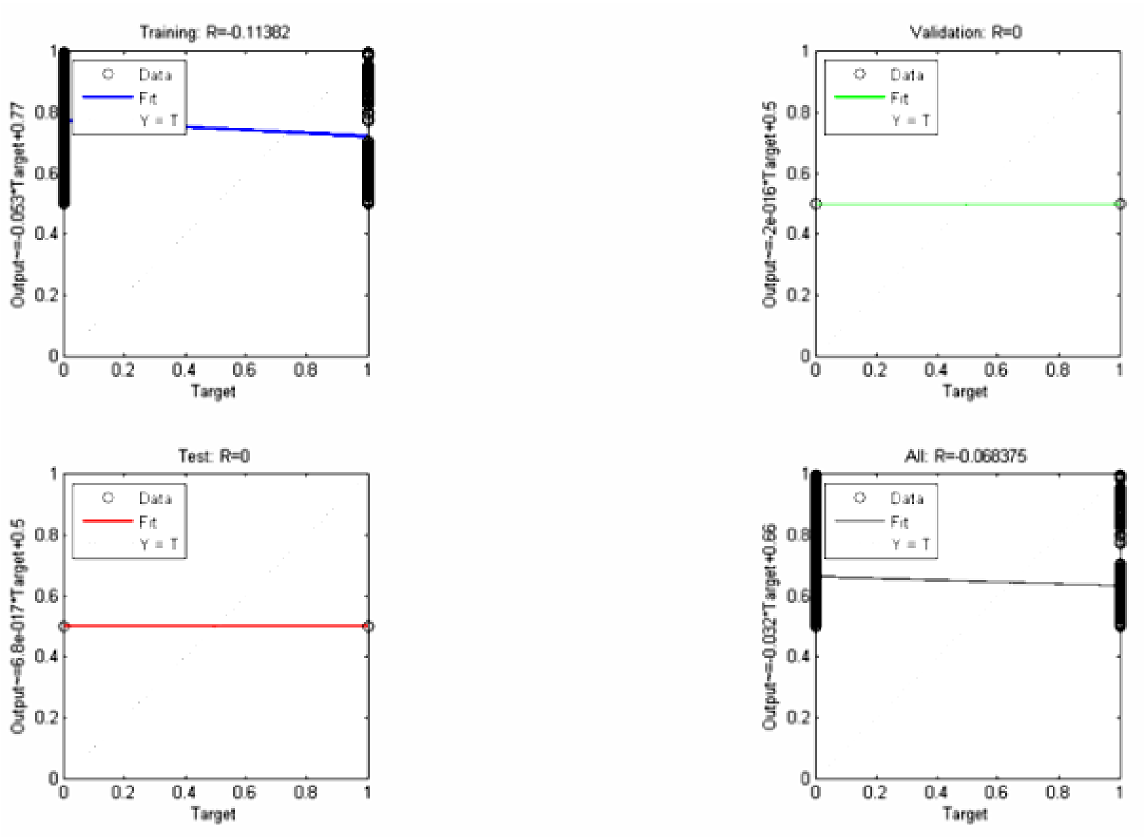
e) Variable Learning Rate Backpropagation





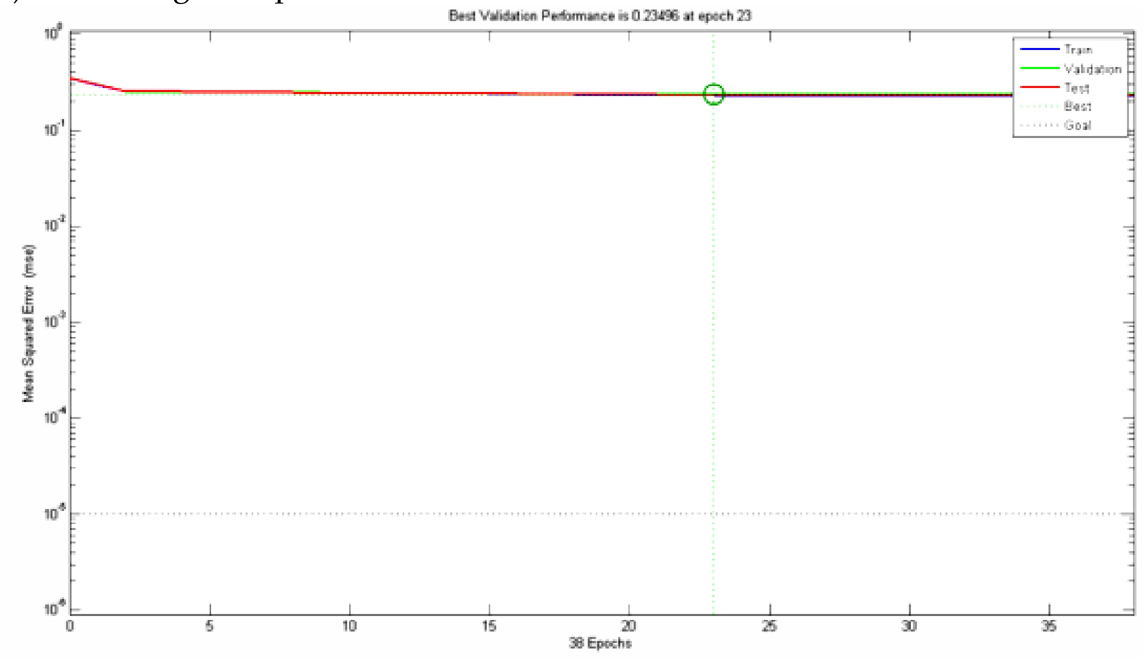
f) BFGS quasi–Newton Backpropagation

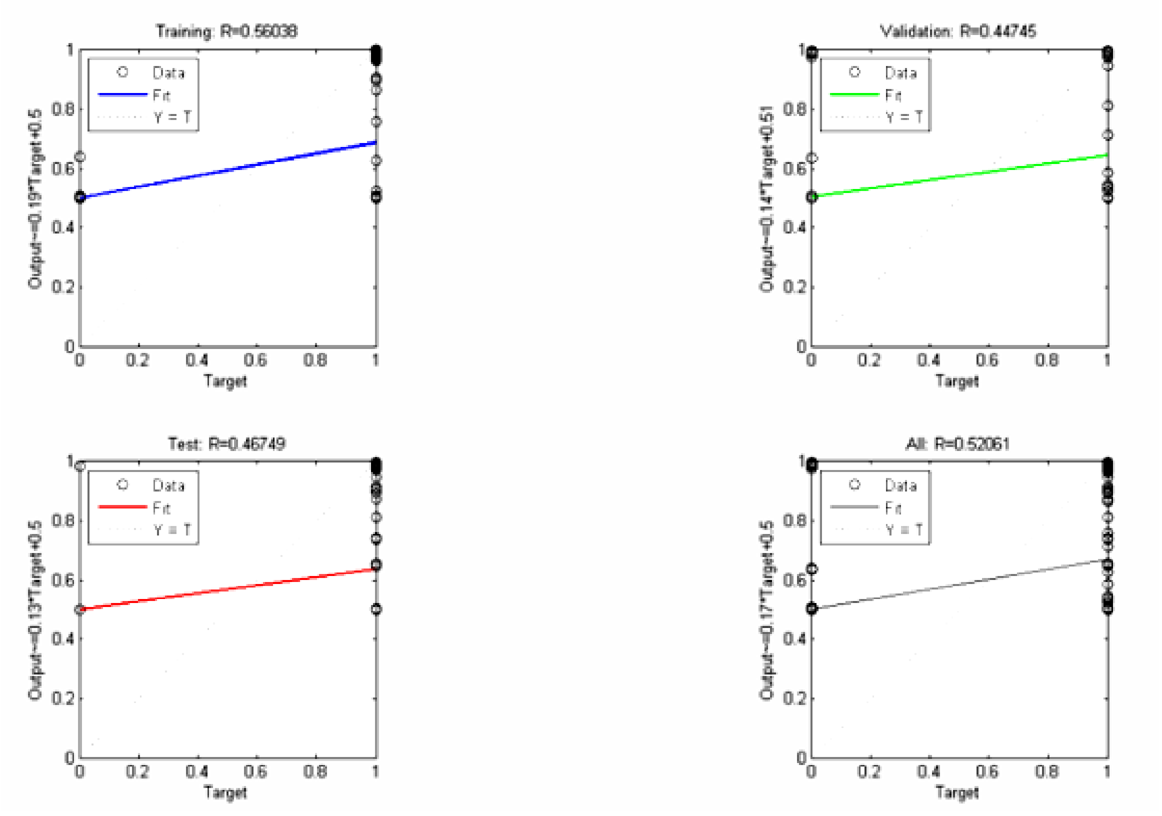




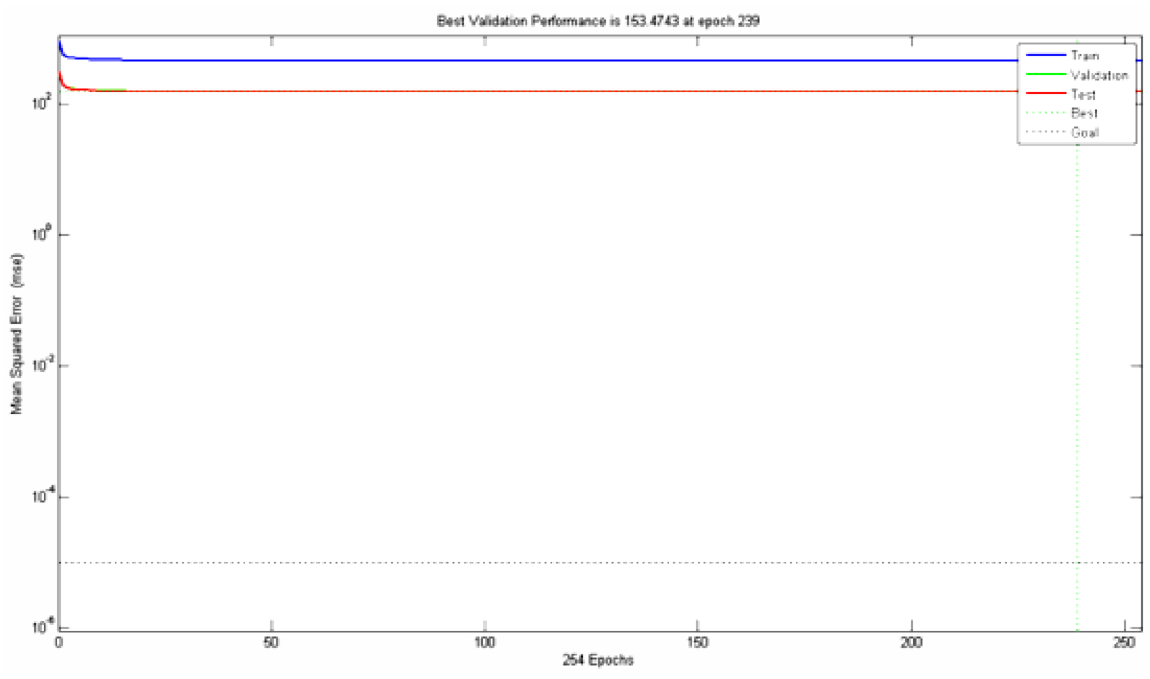
5) [8]

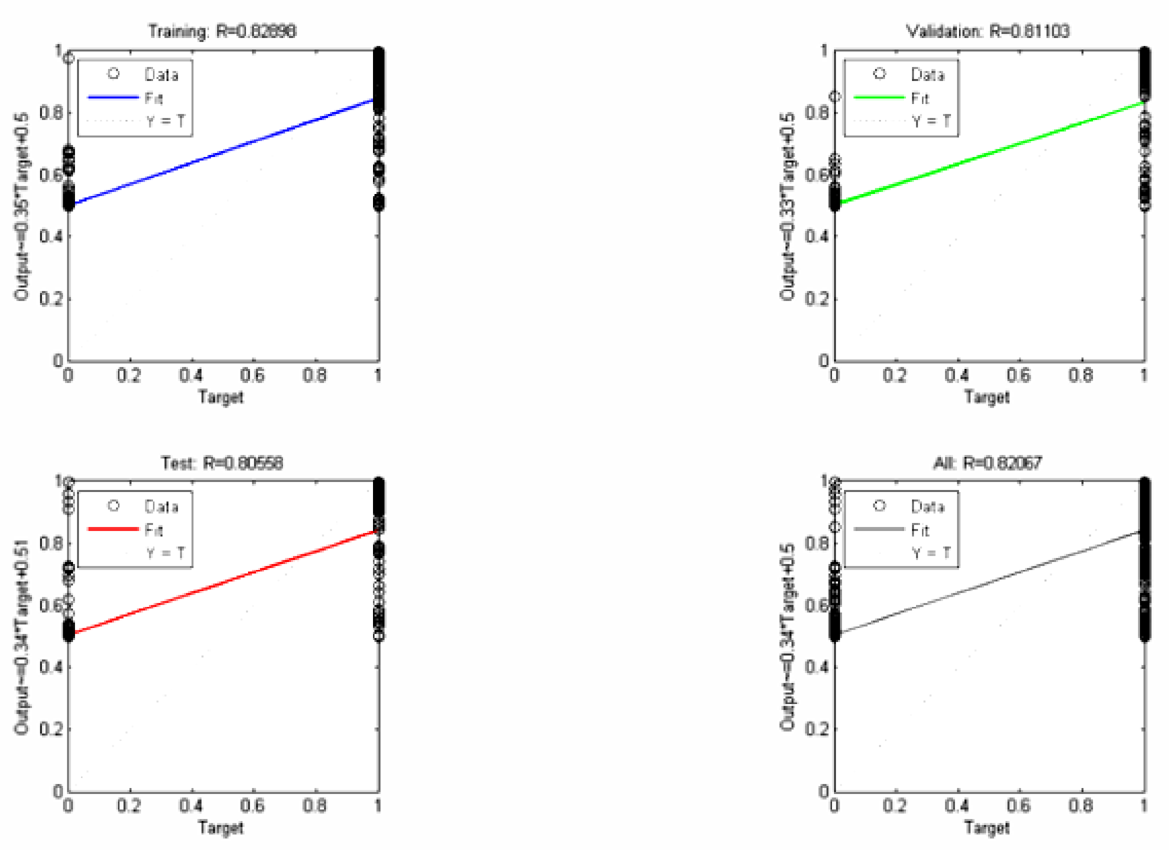
a) Levenberg–Marquardt



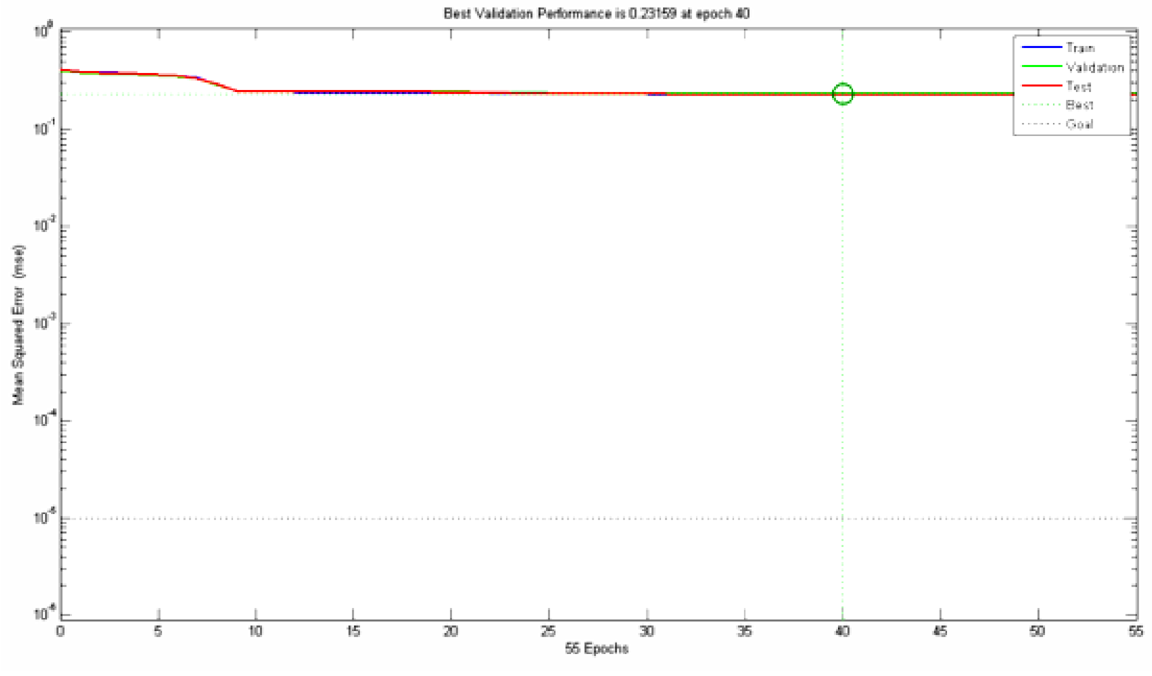


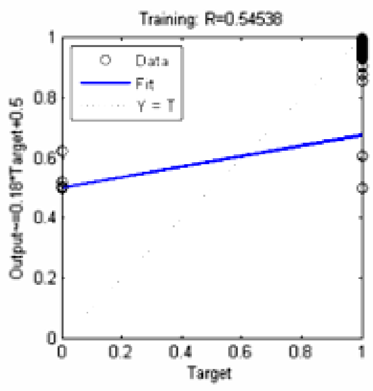
b) Bayesian Regularisation

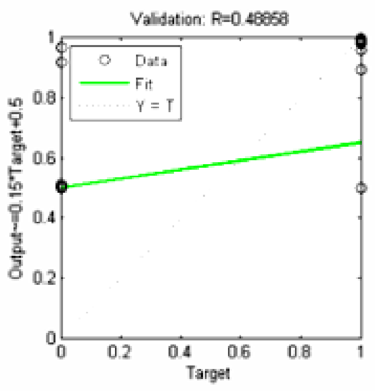


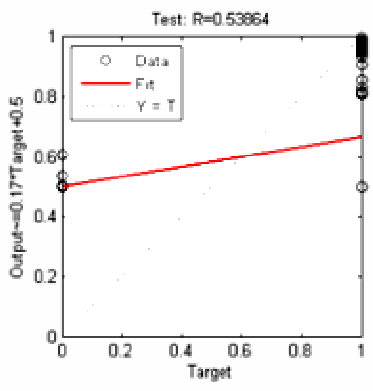


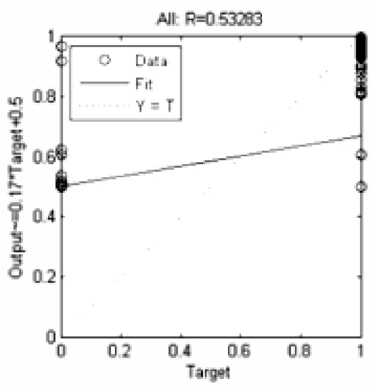
c) Resilient Backpropagation



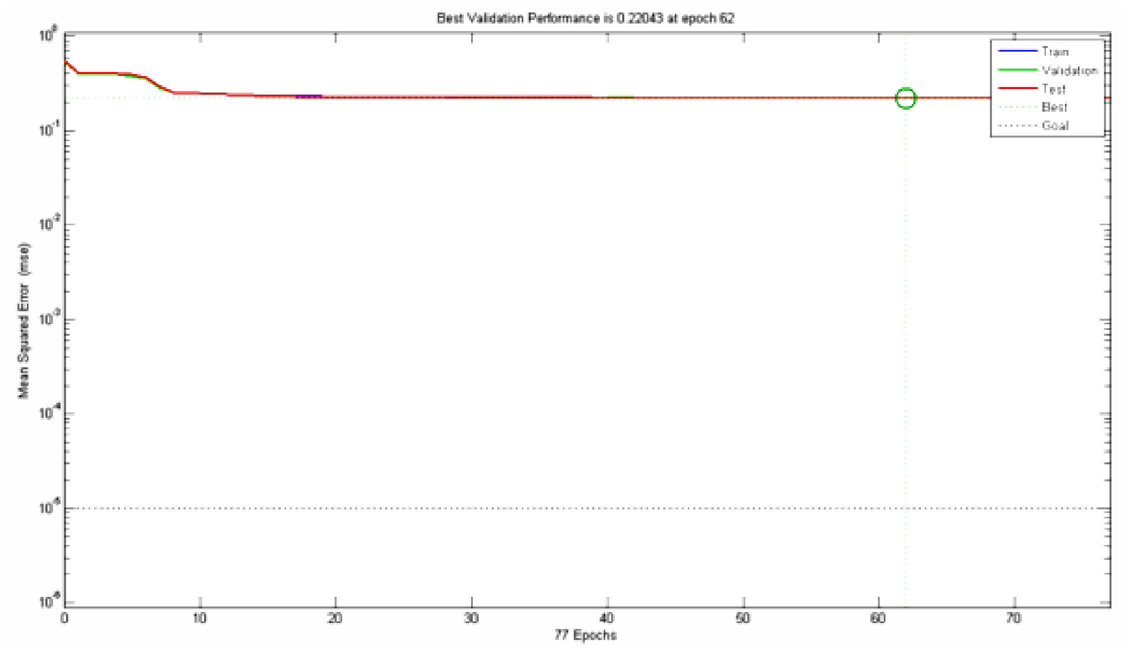


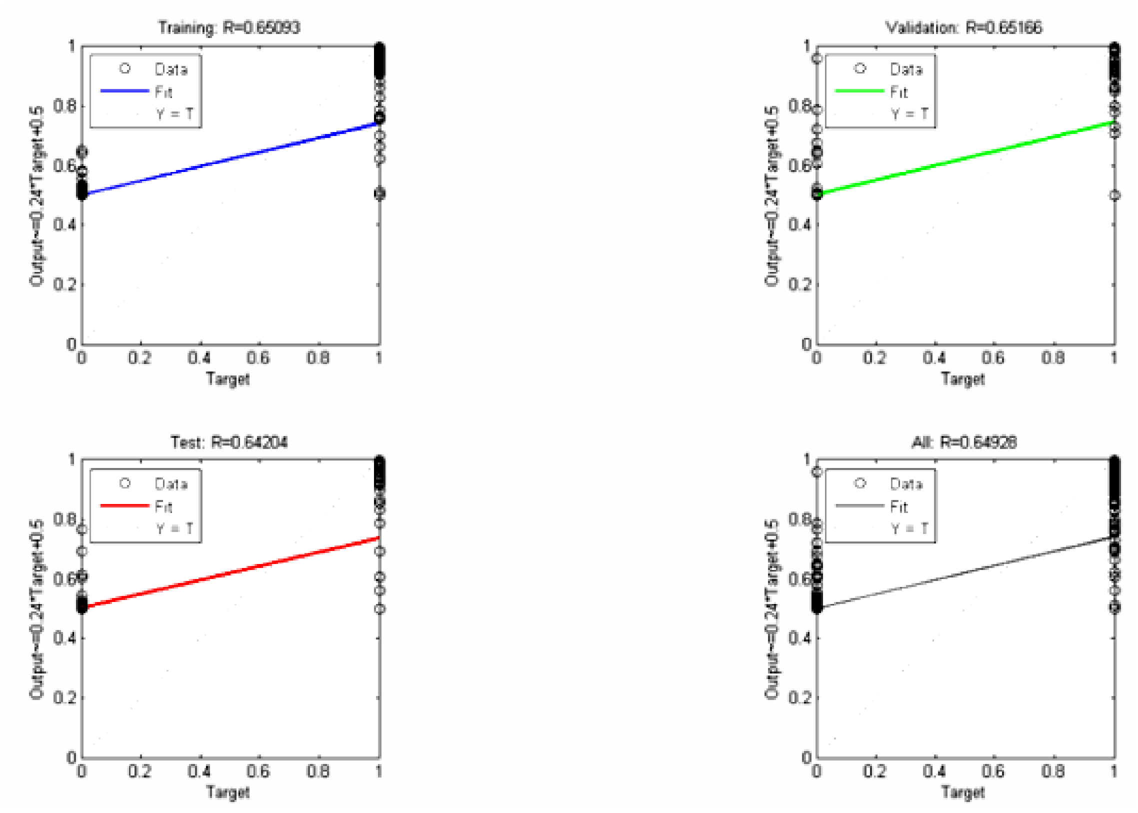




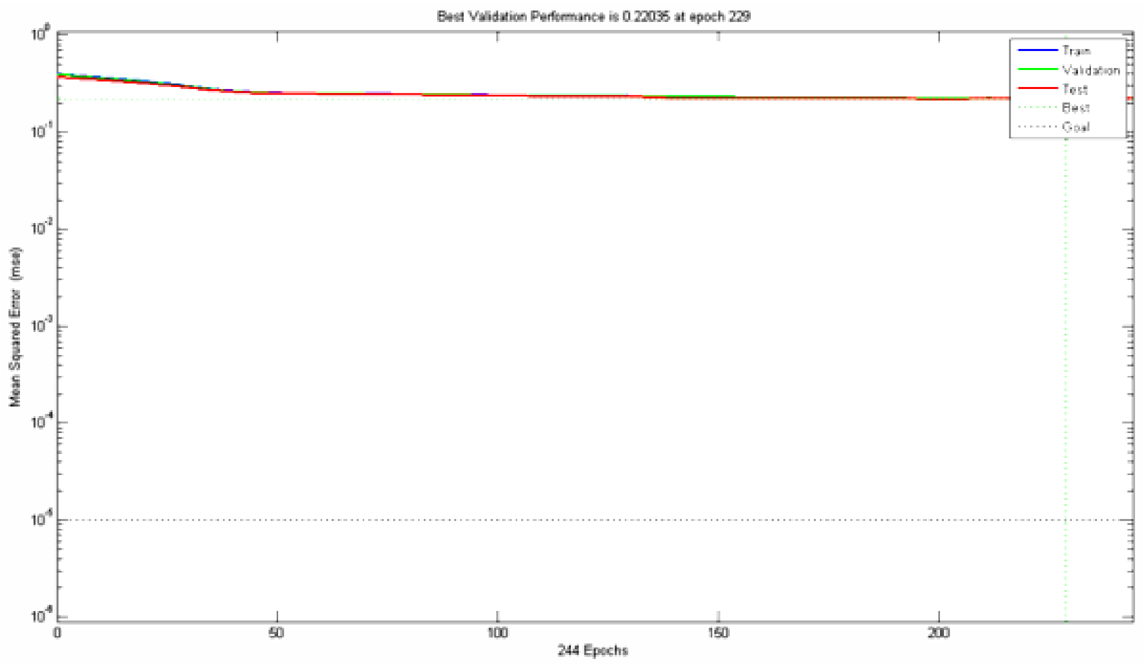


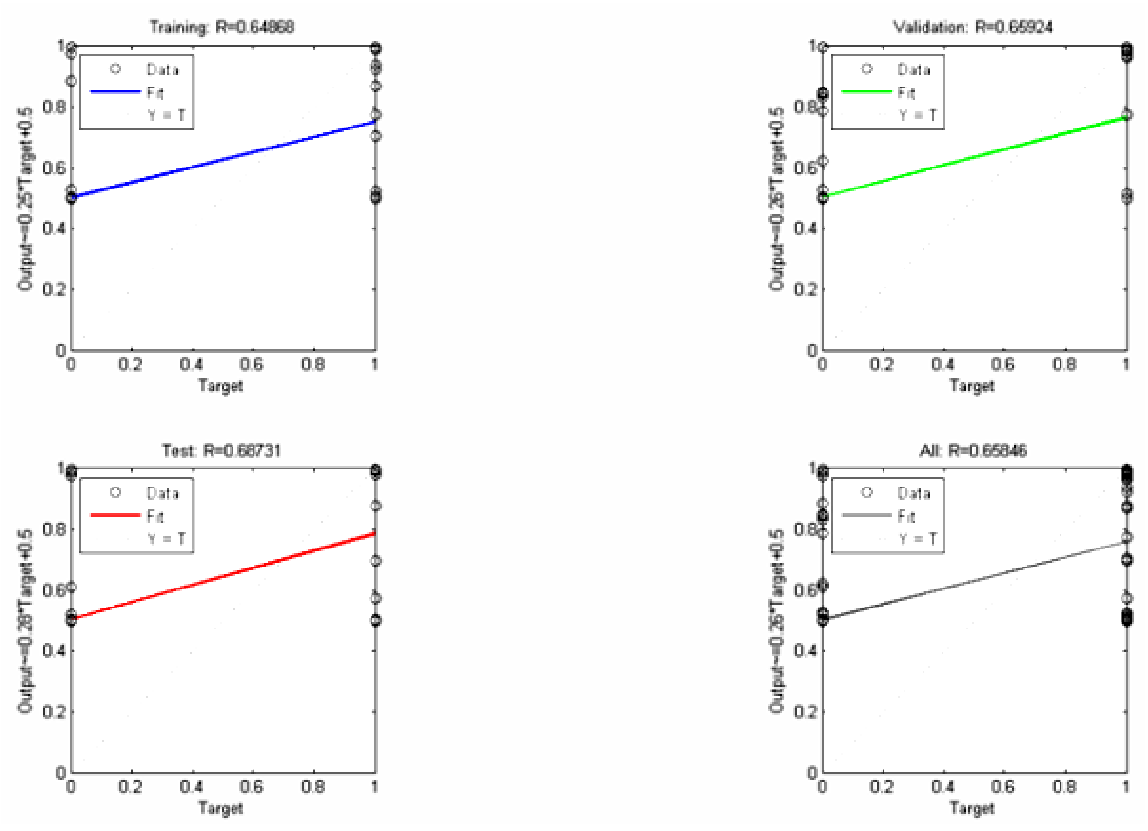
d) Scaled Conjugate Gradient



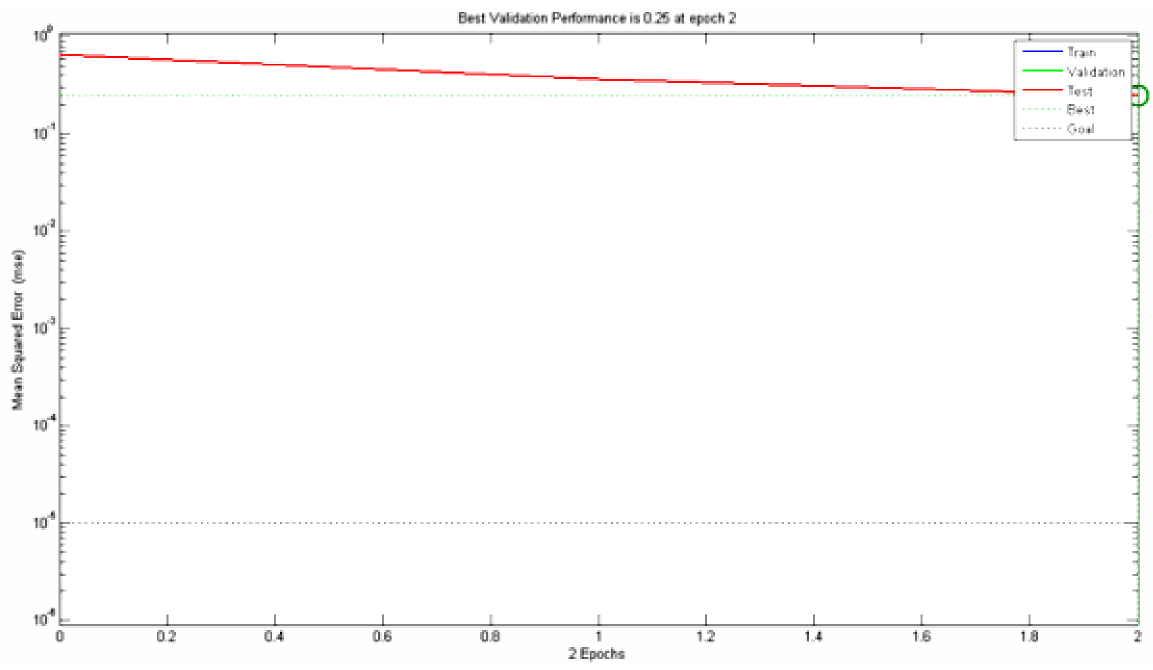


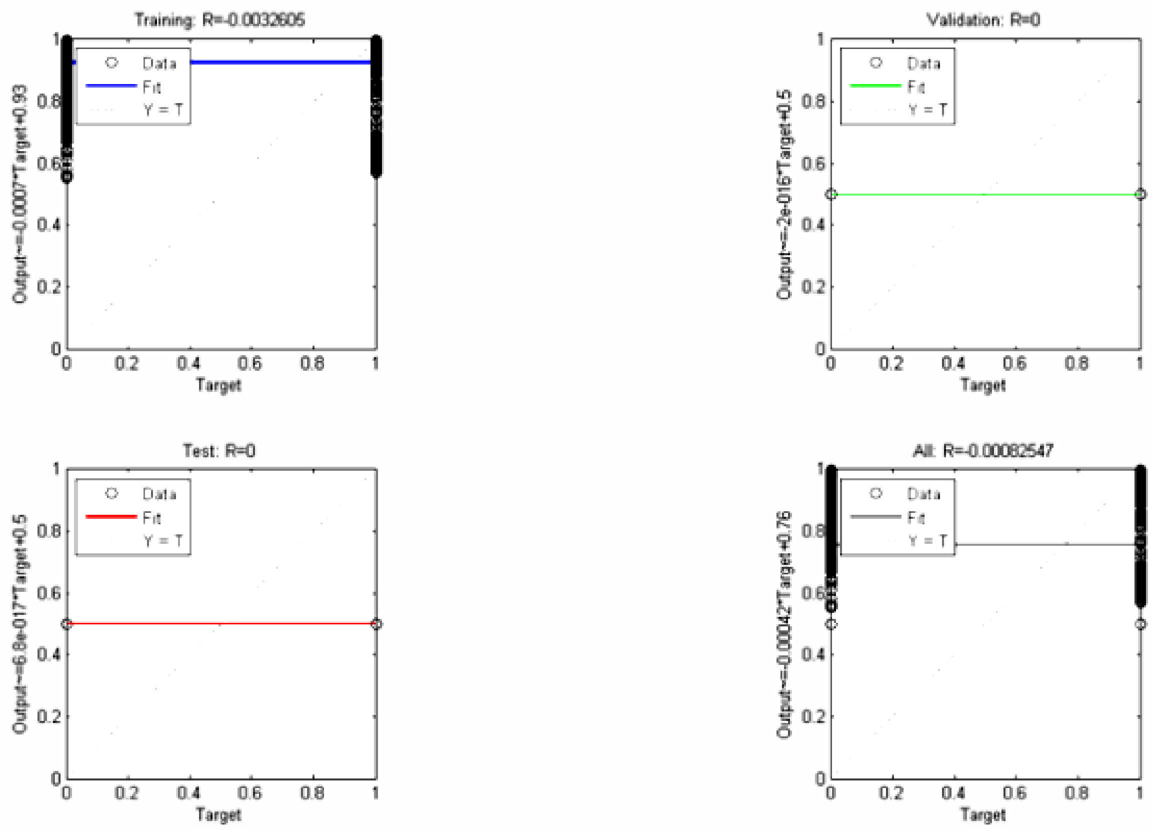
e) Variable Learning Rate Backpropagation





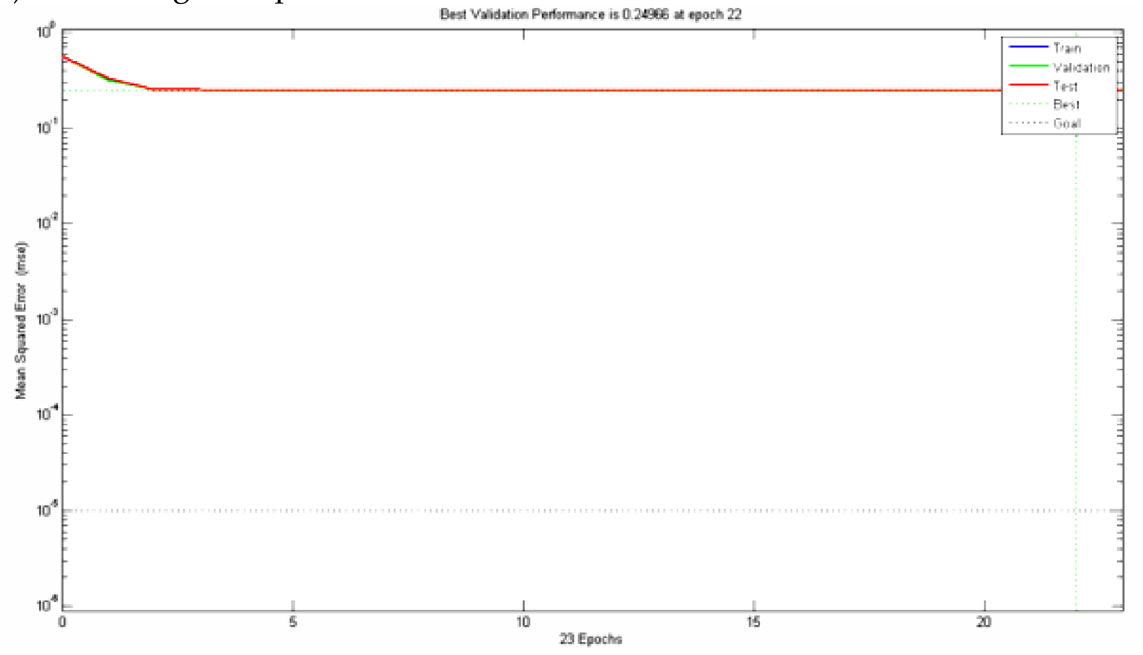
f) BFGS quasi–Newton Backpropagation

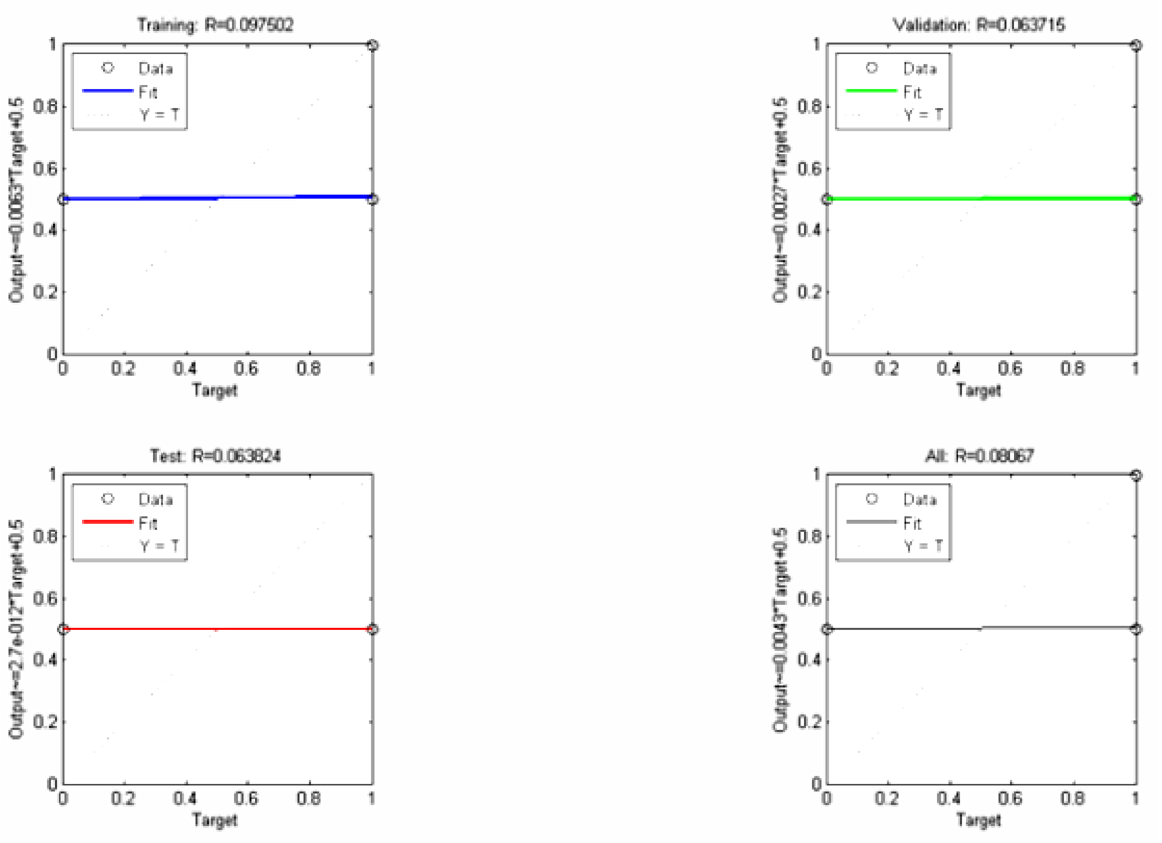




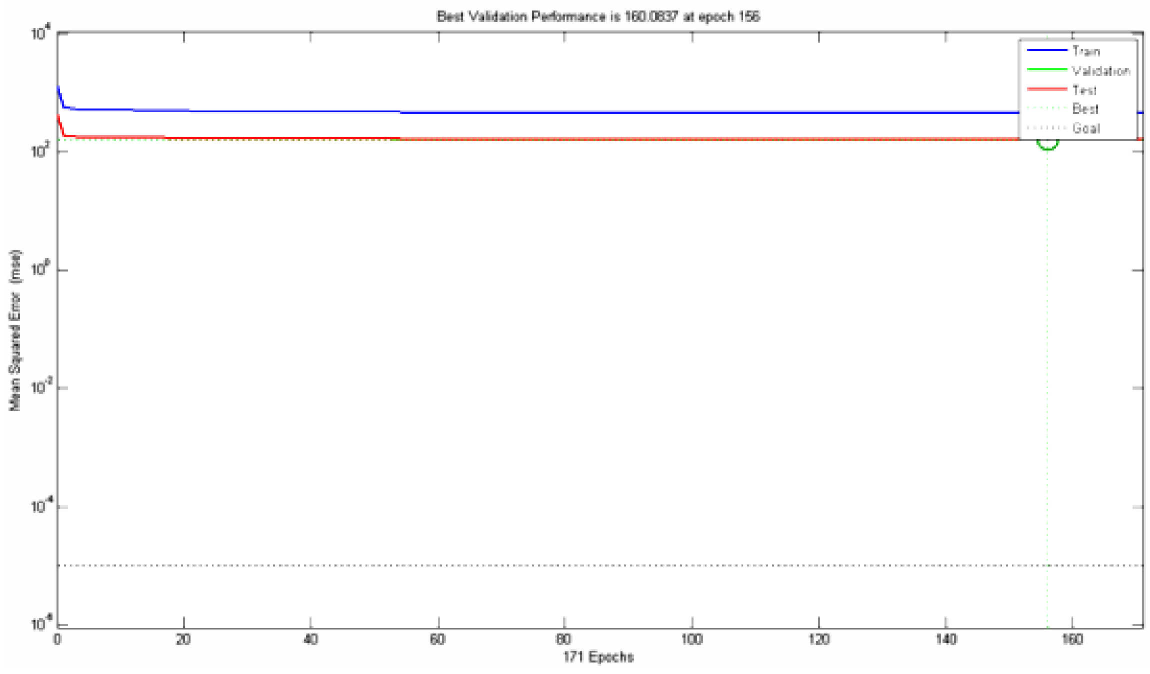
6) [10]

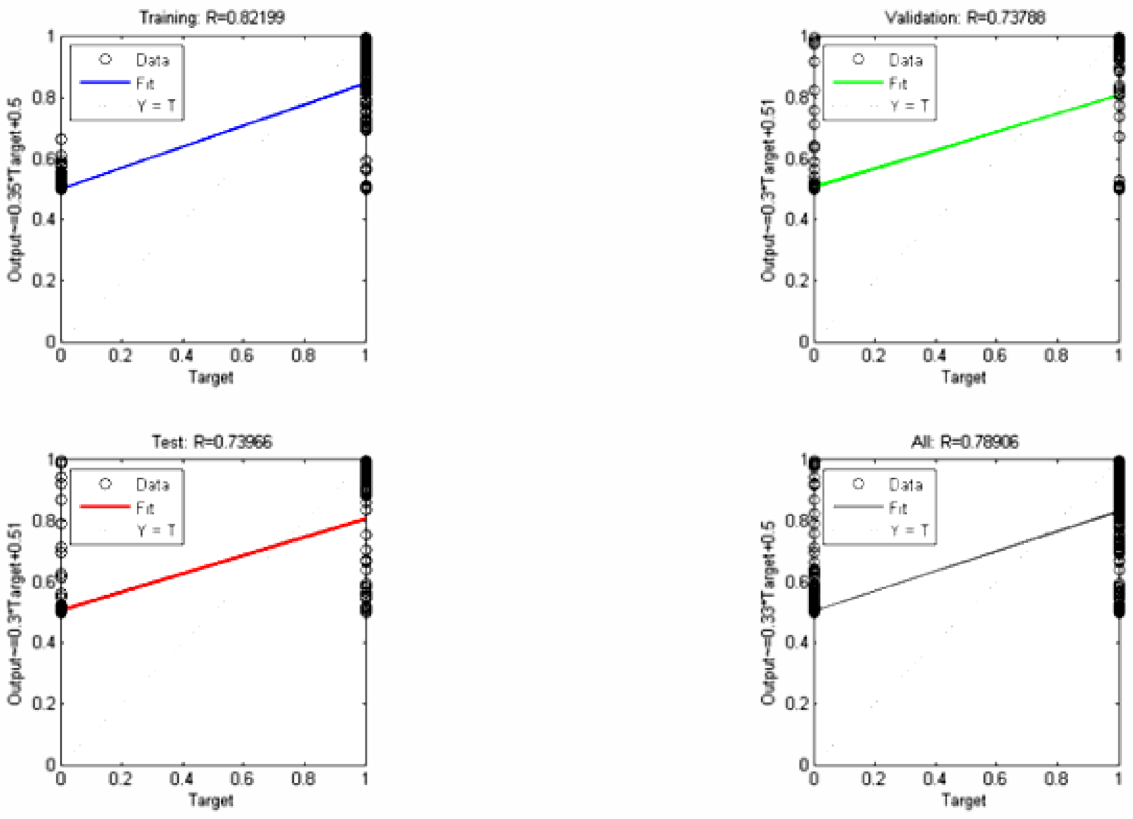
a) Levenberg–Marquardt



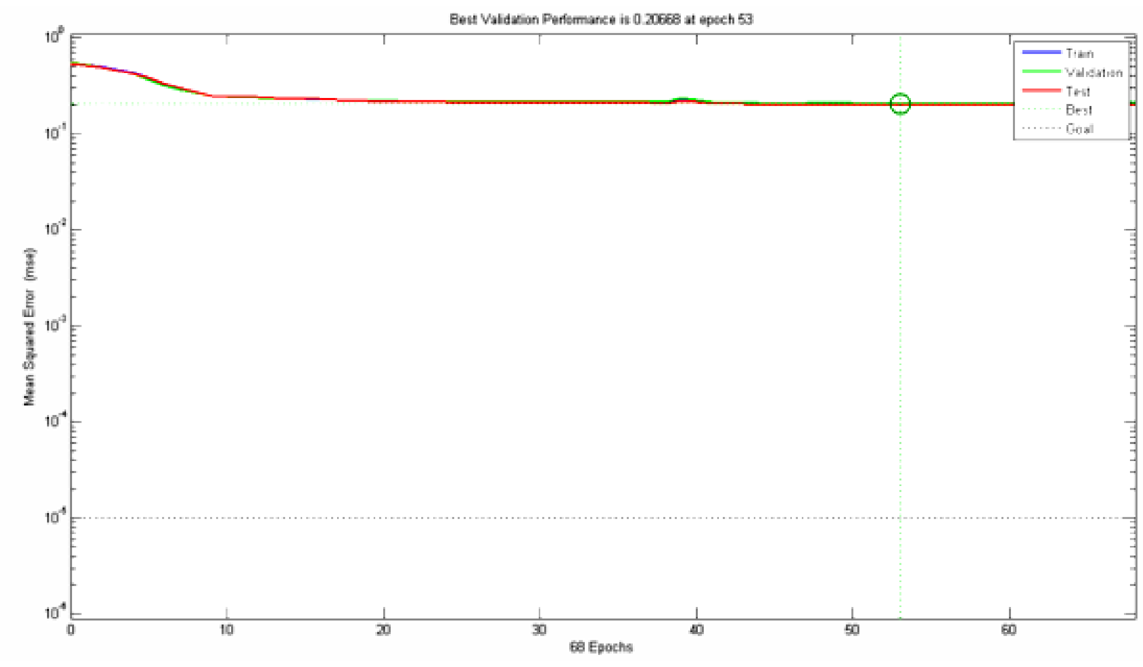


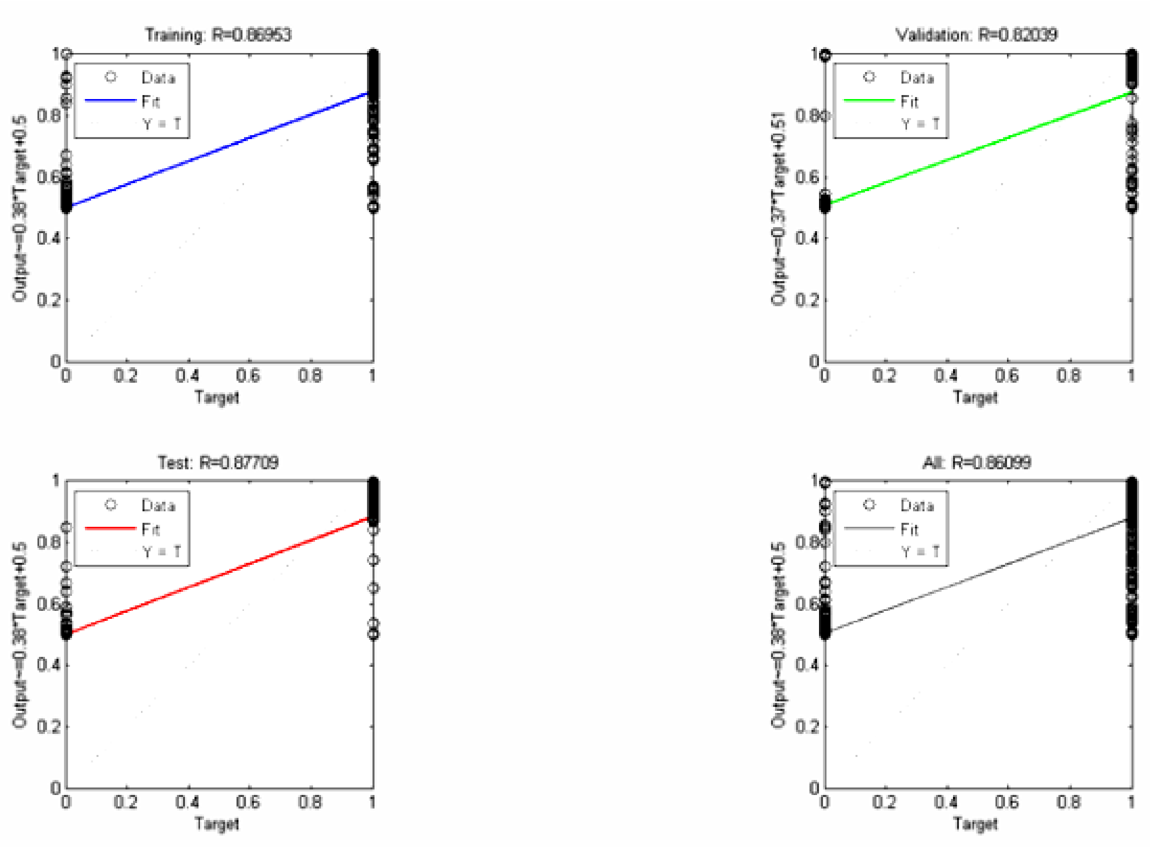
b) Bayesian Regularisation



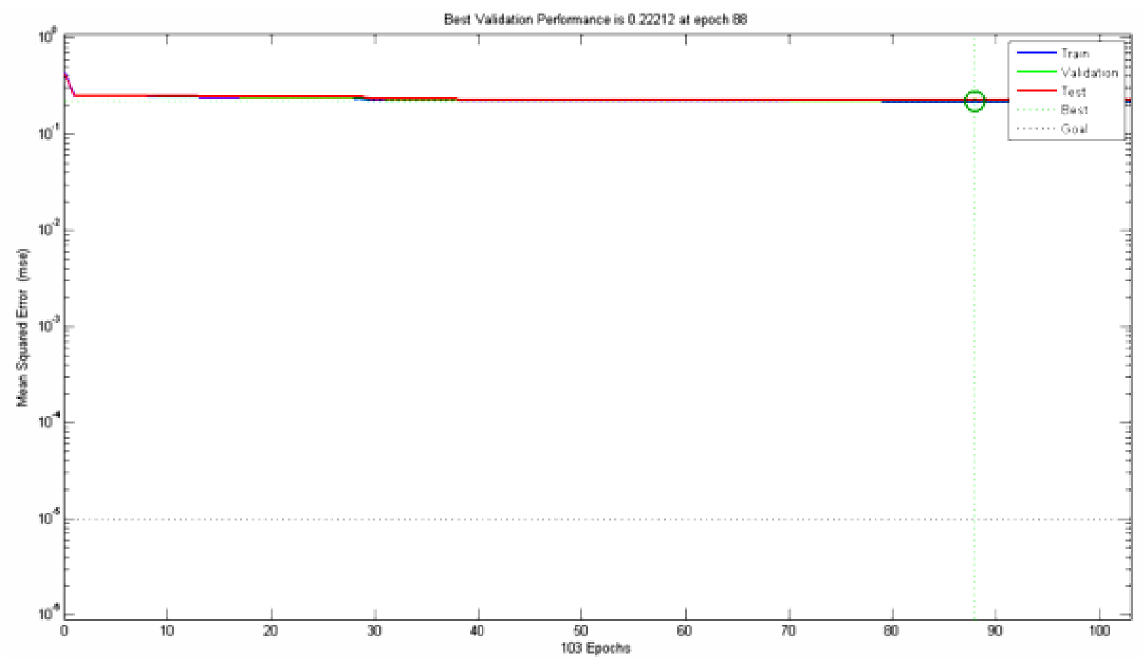


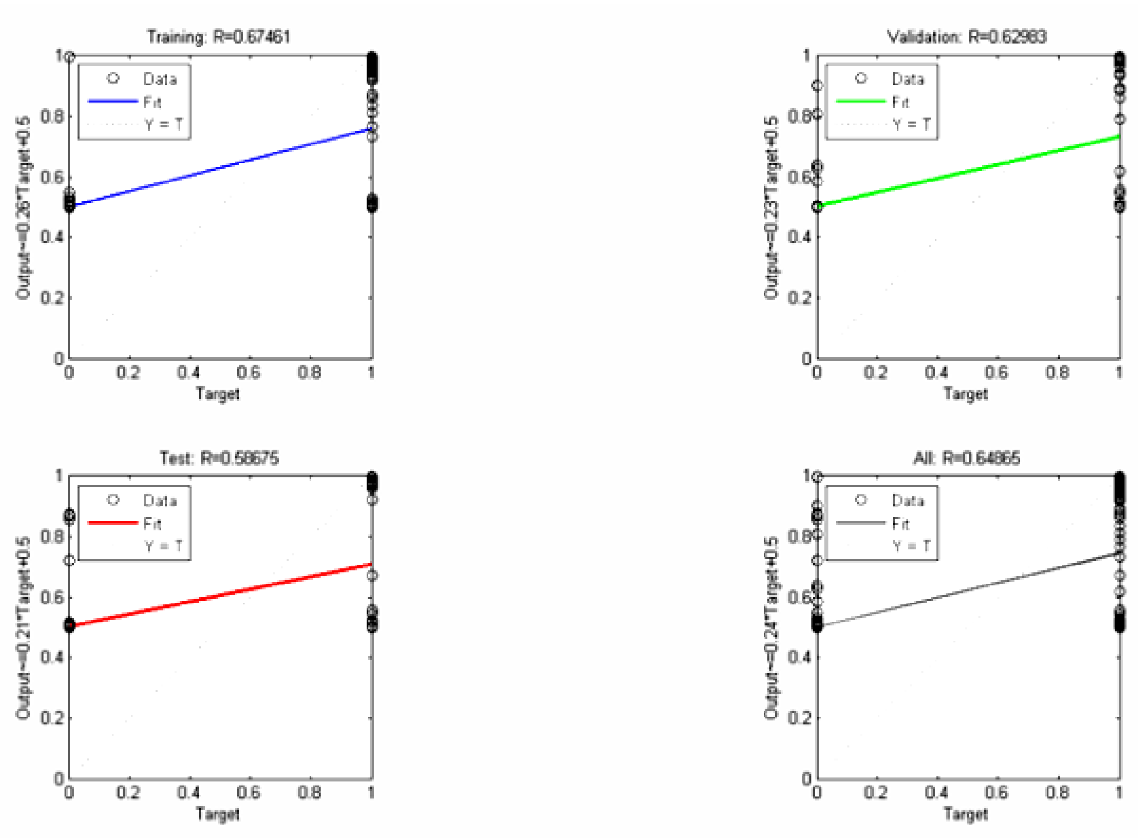
c) Resilient Backpropagation



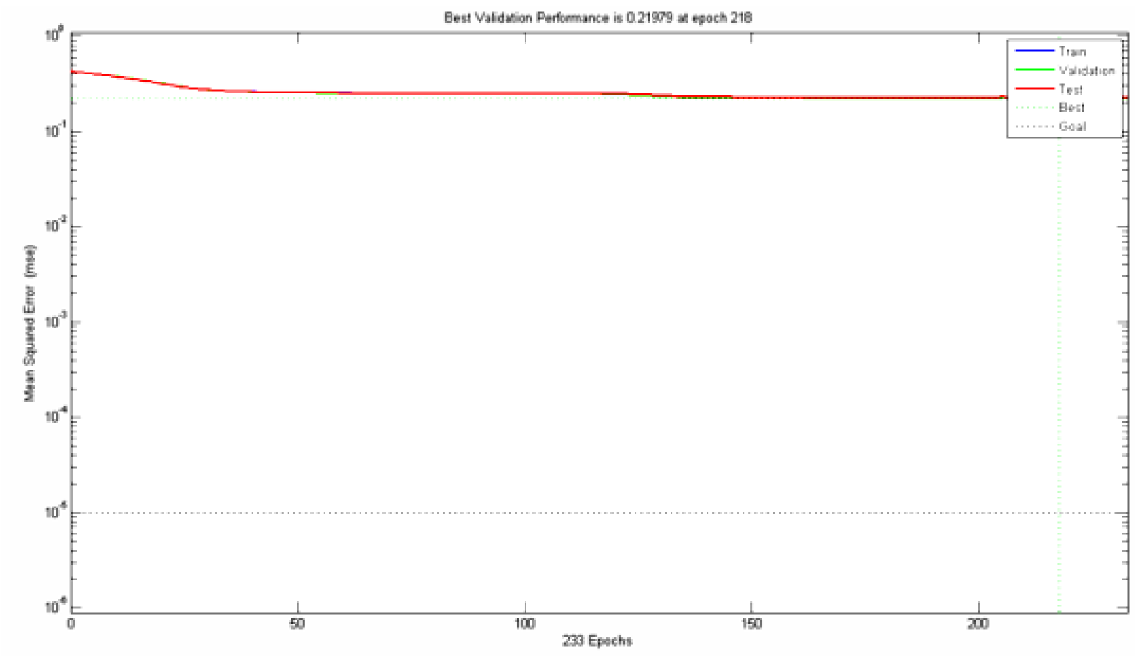


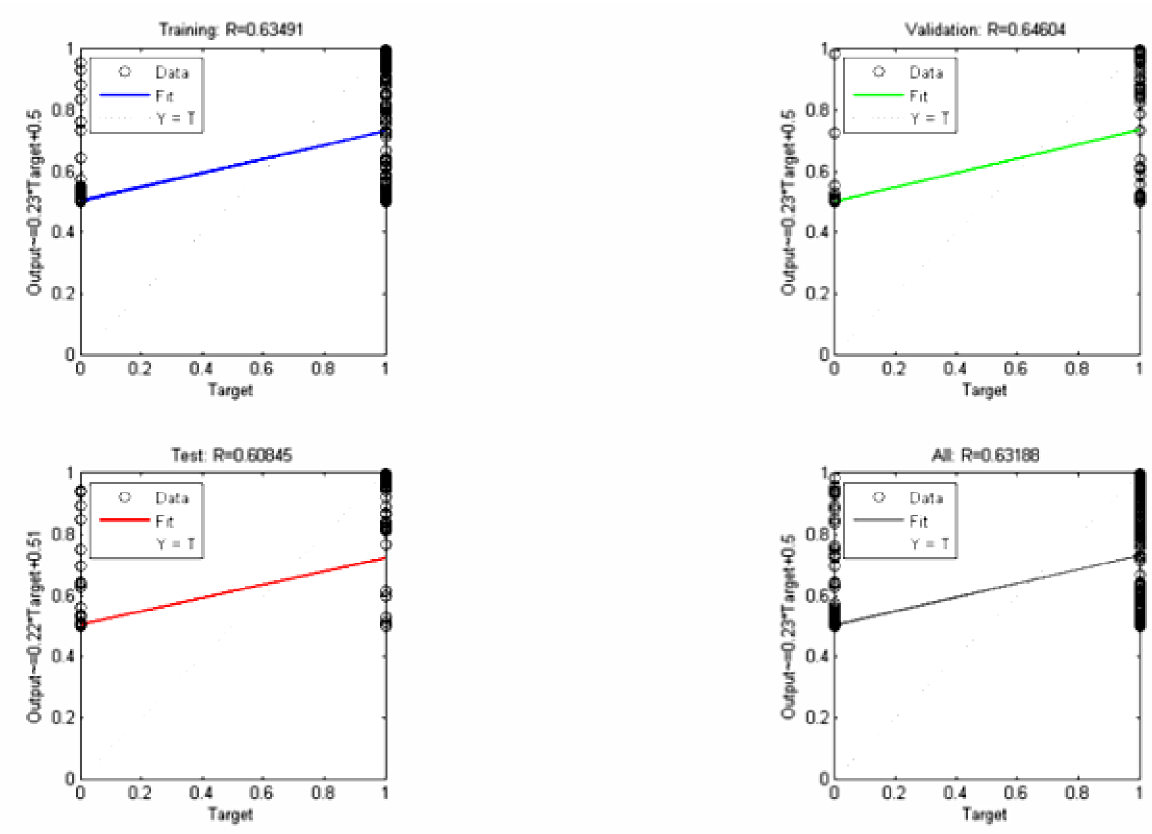
d) Scaled Conjugate Gradient



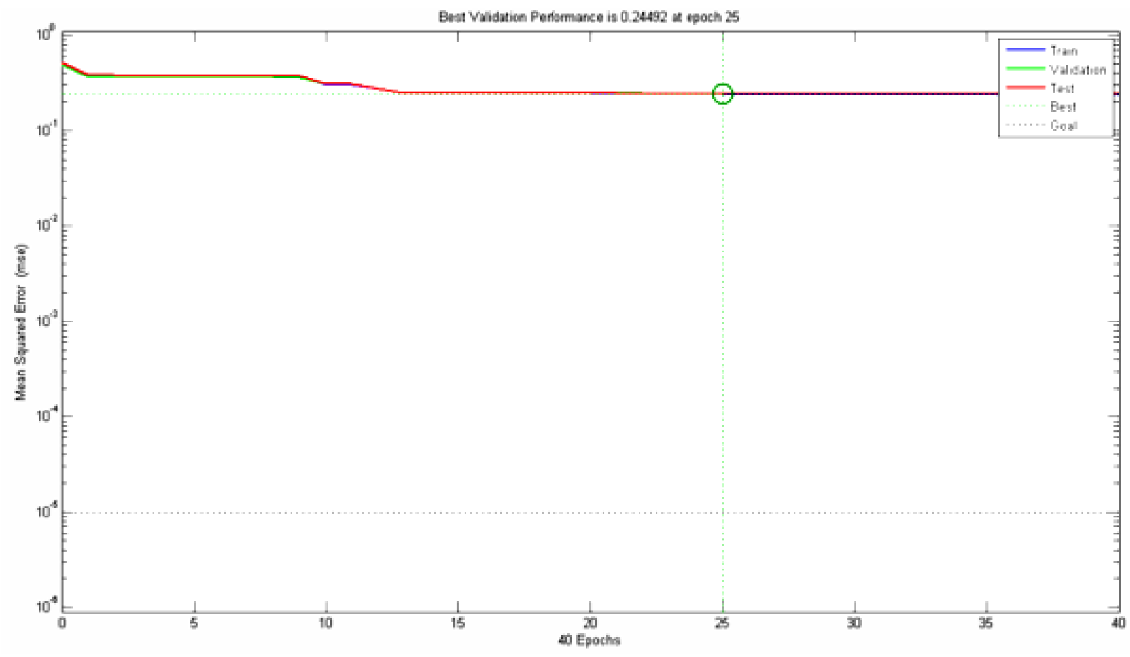


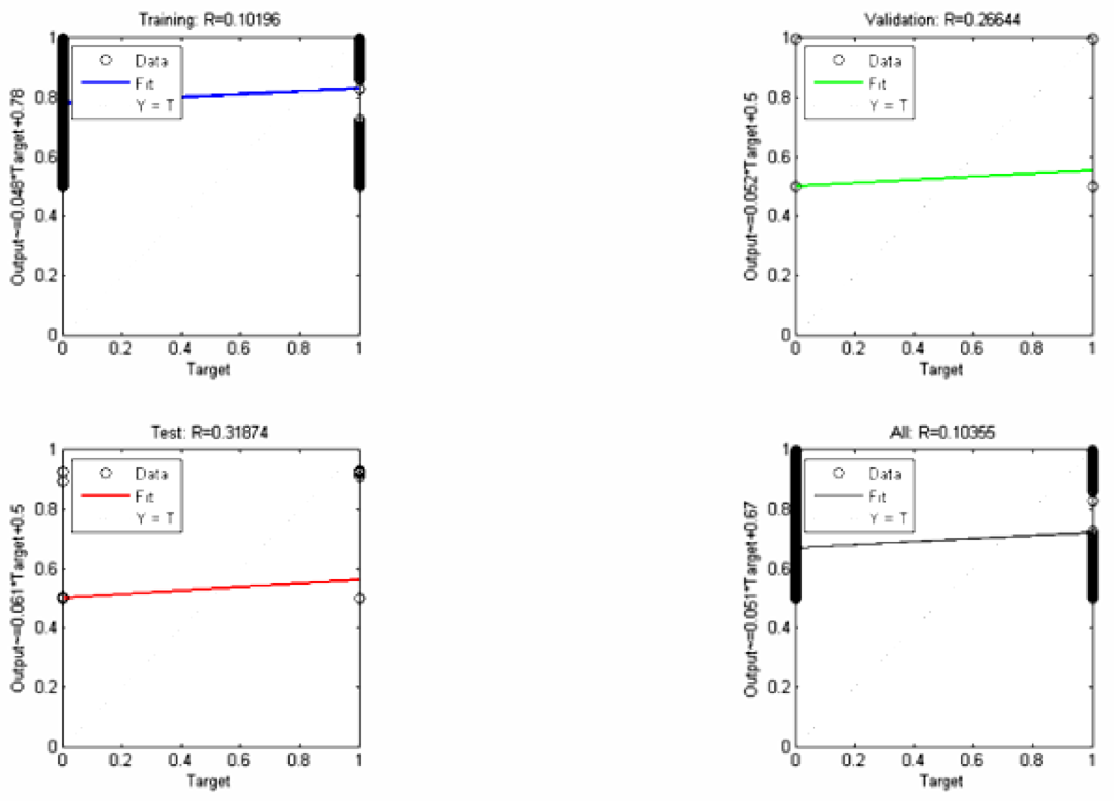
e) Variable Learning Rate Backpropagation





f) BFGS quasi–Newton Backpropagation

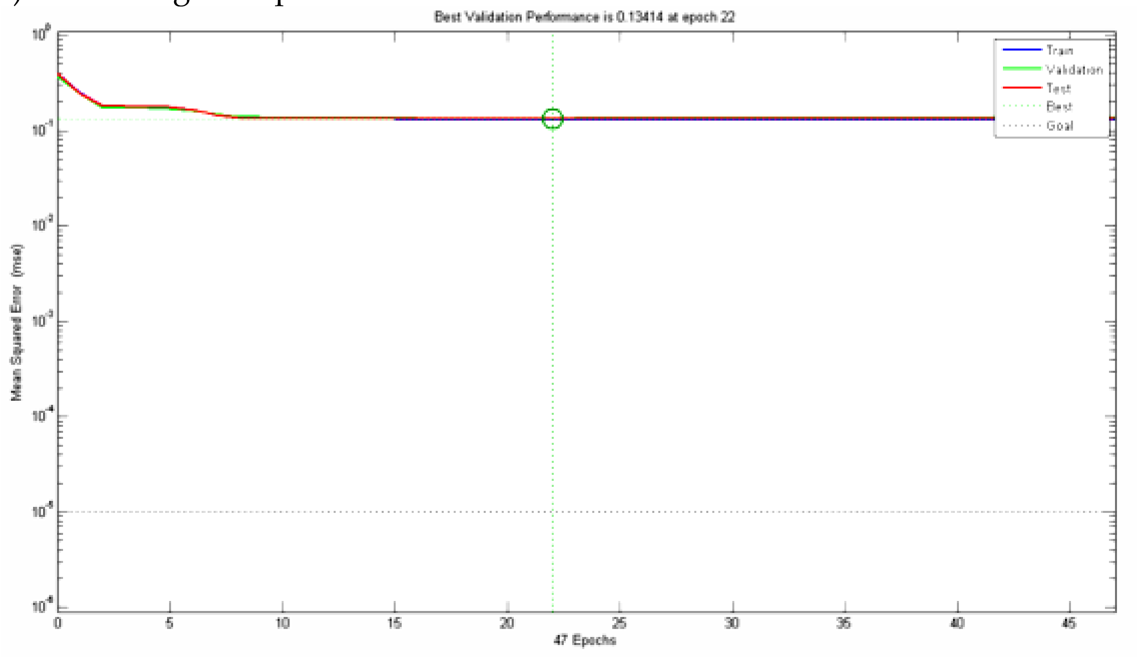


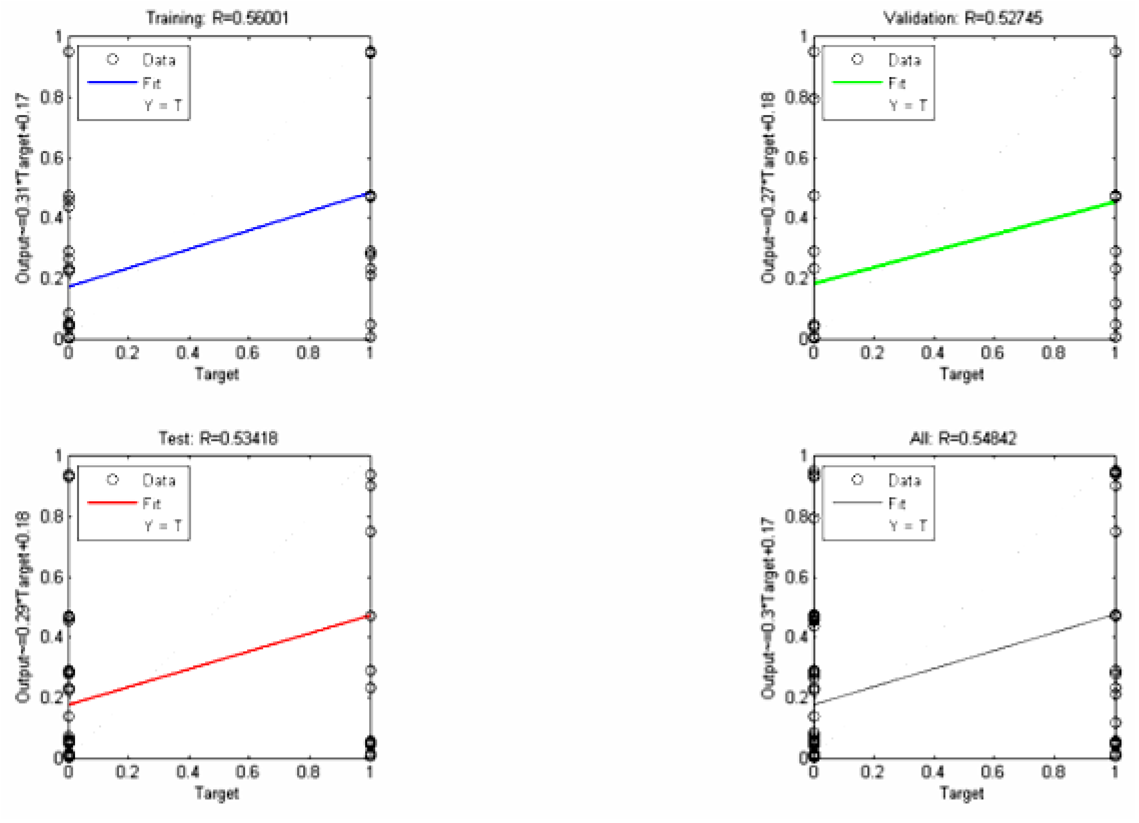


2. Two hidden layers

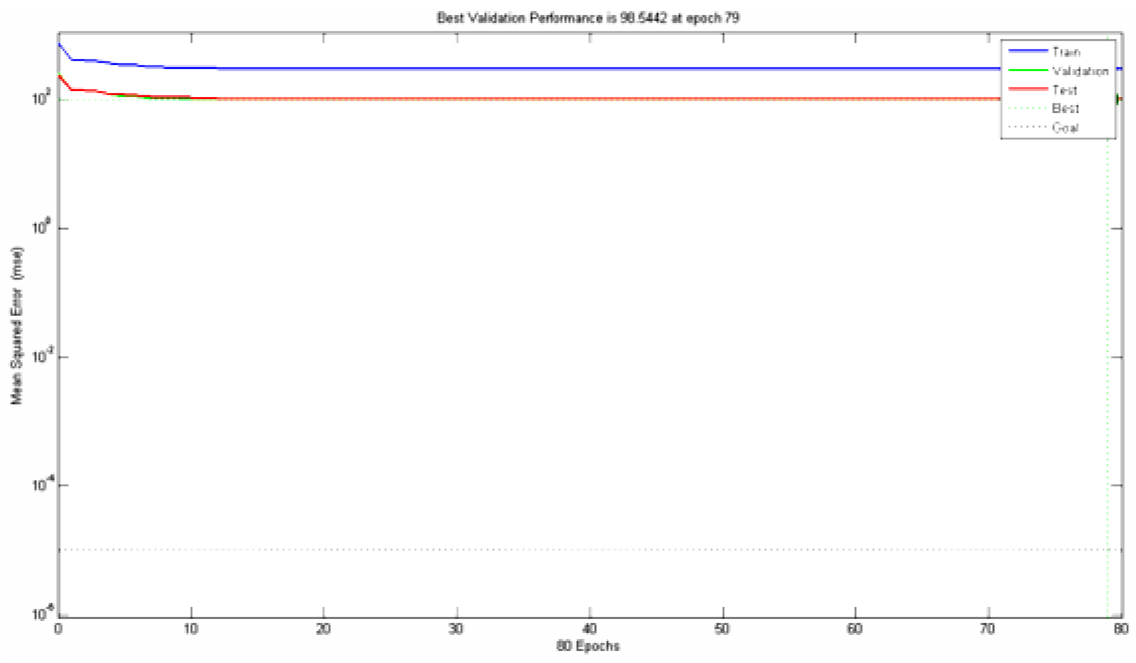
1. [1 1]

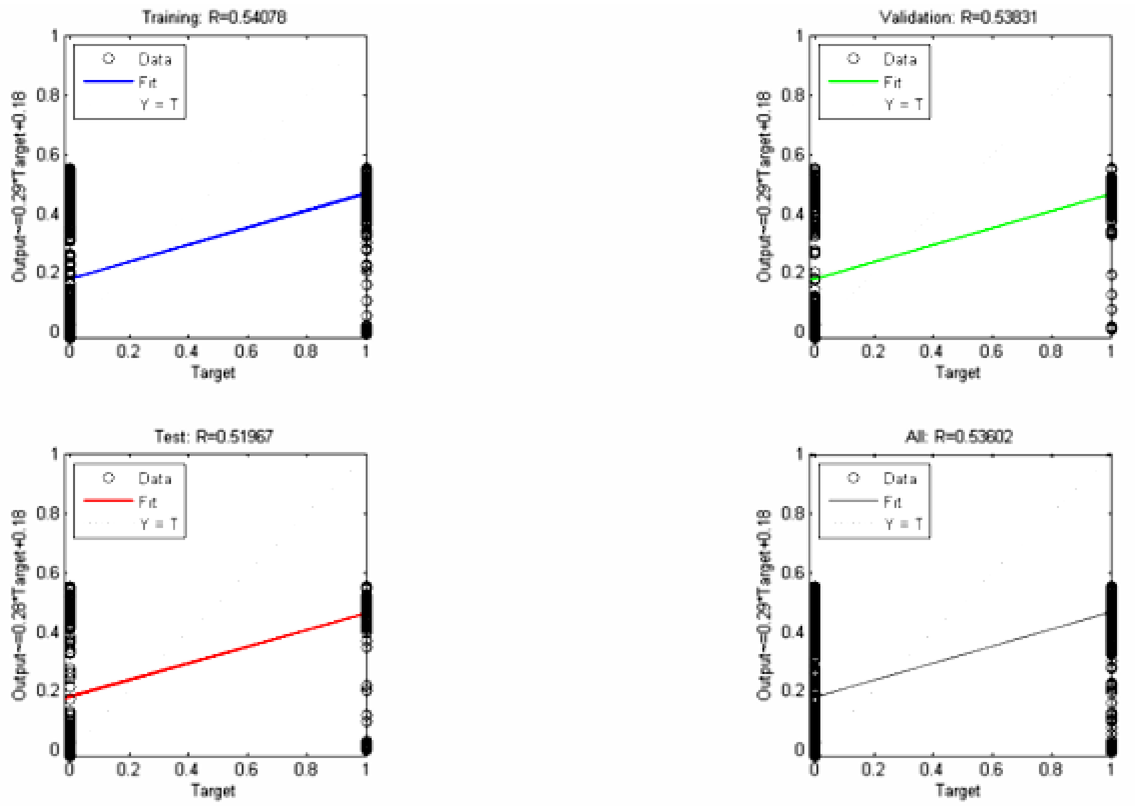
a) Levenberg–Marquardt



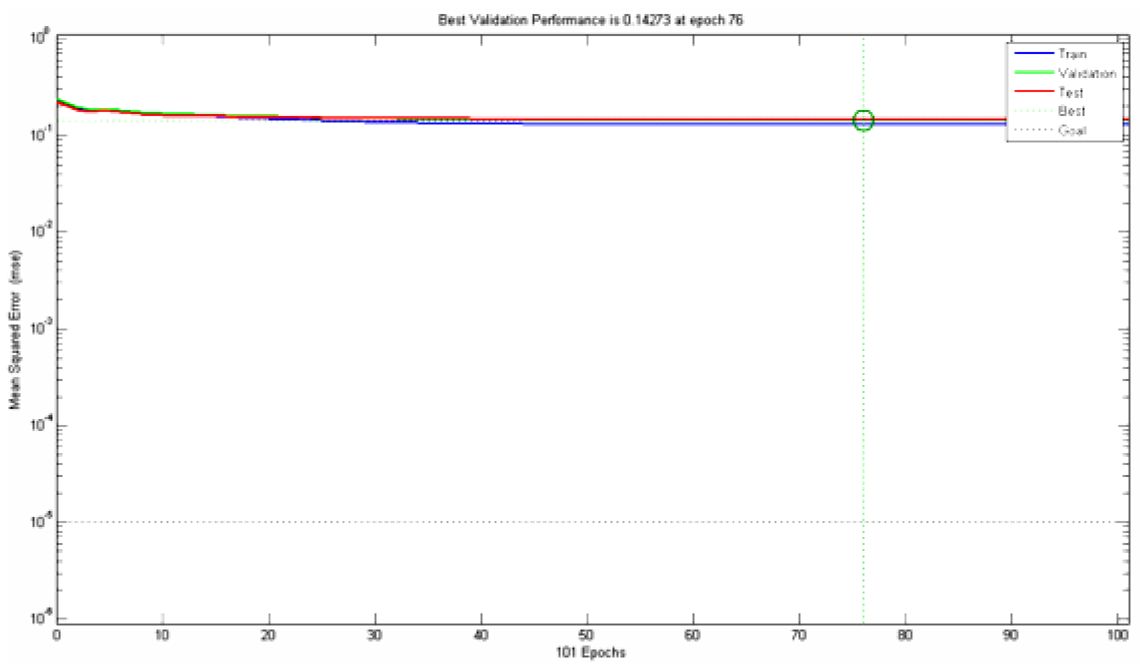


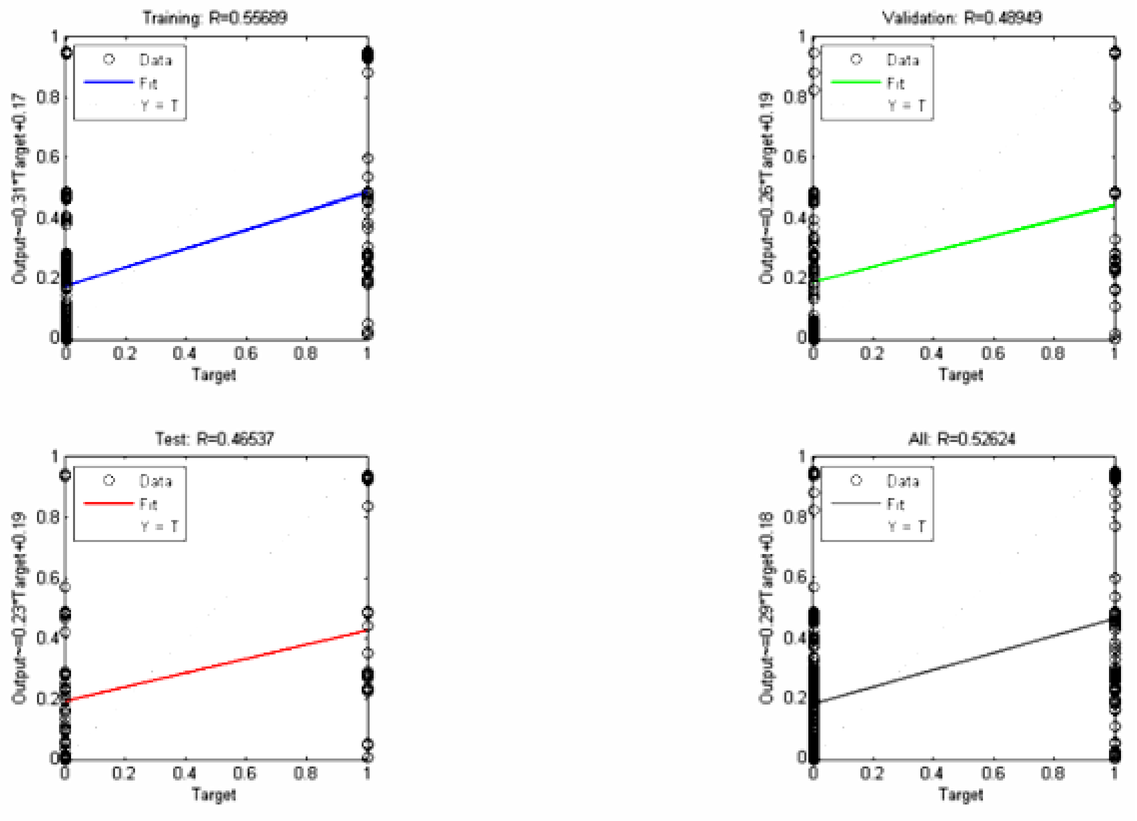
b) Bayesian Regularisation



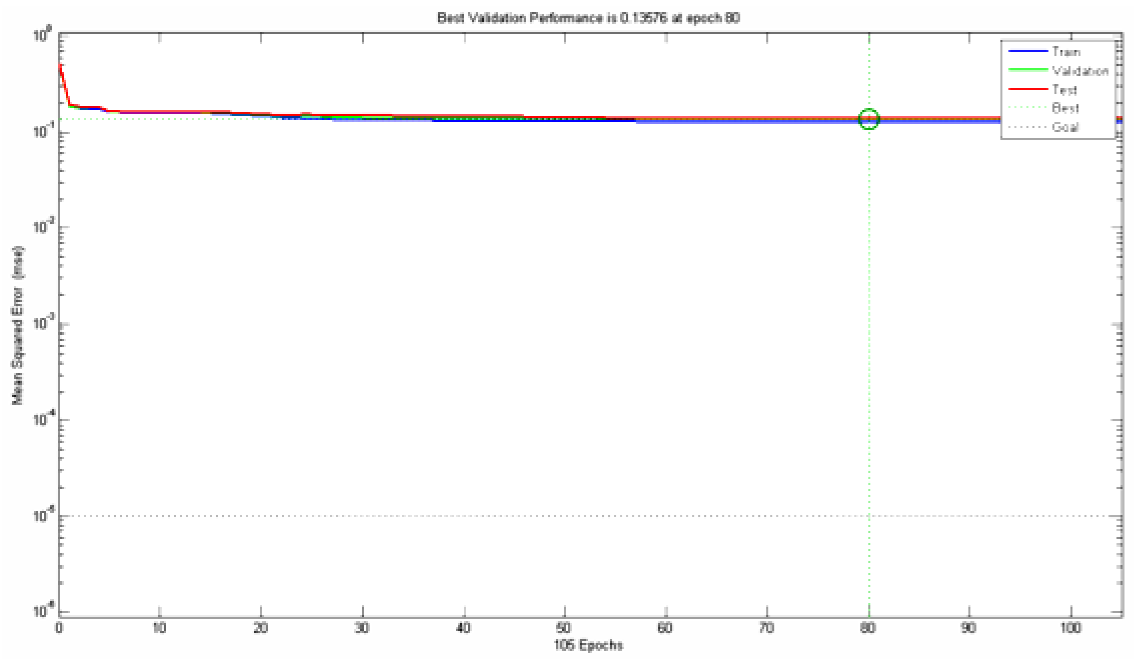


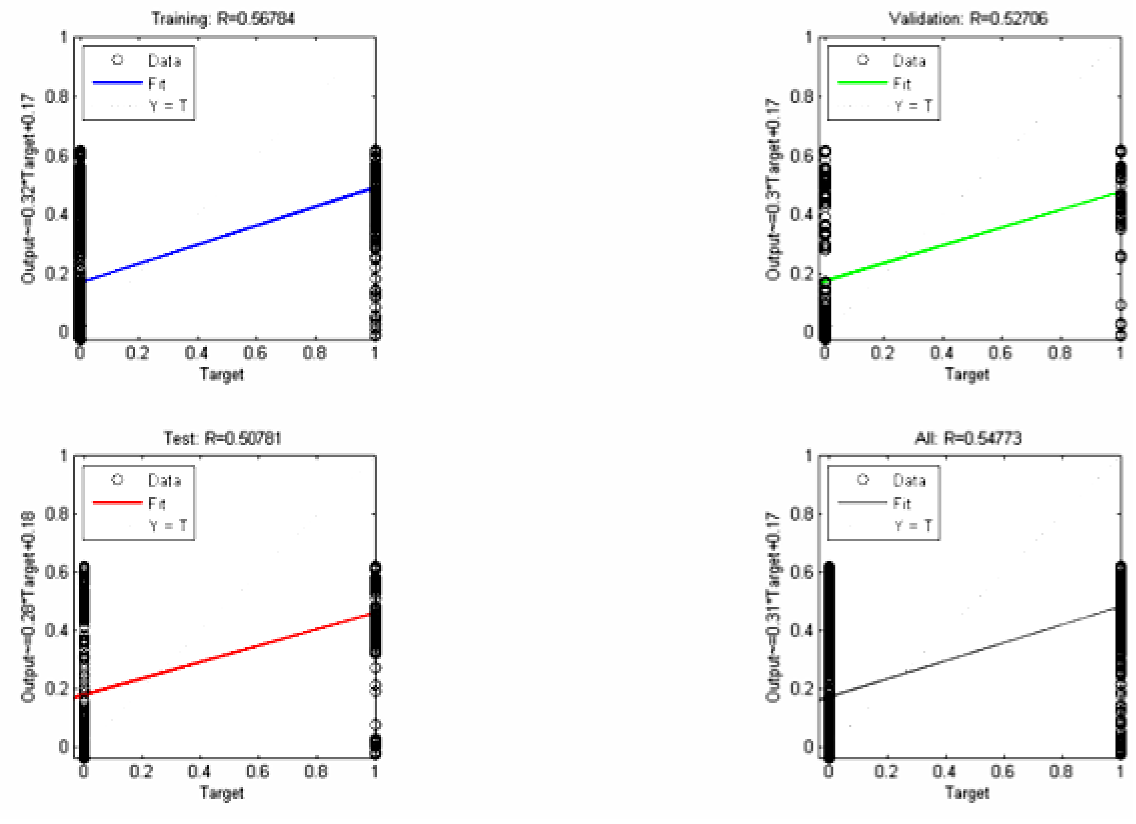
c) Resilient Backpropagation



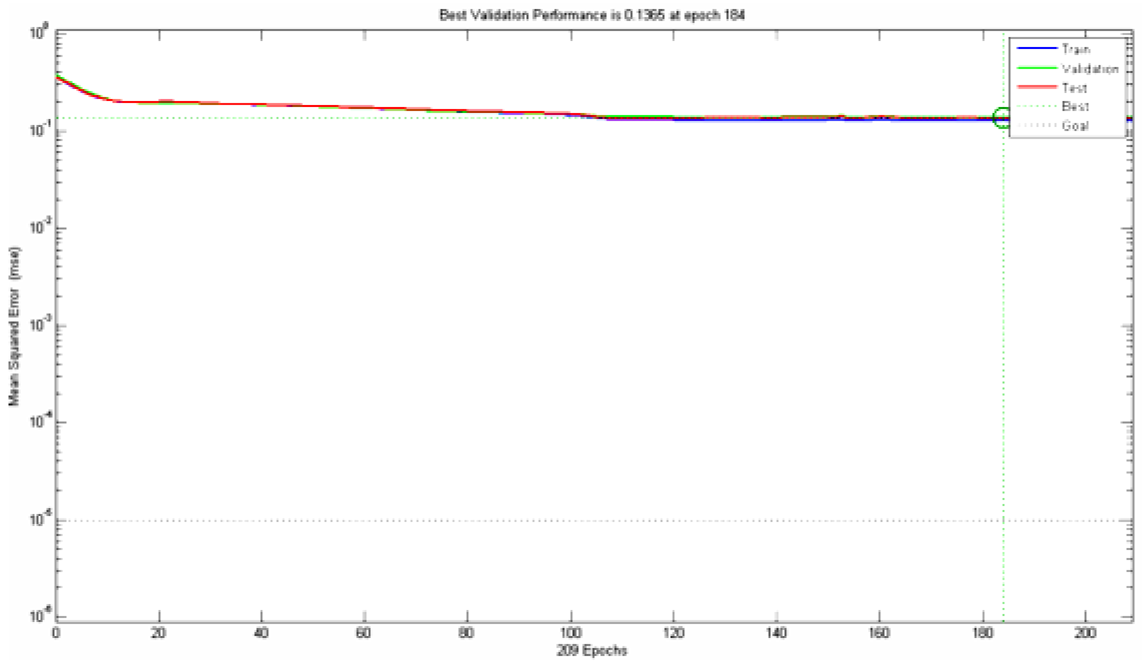


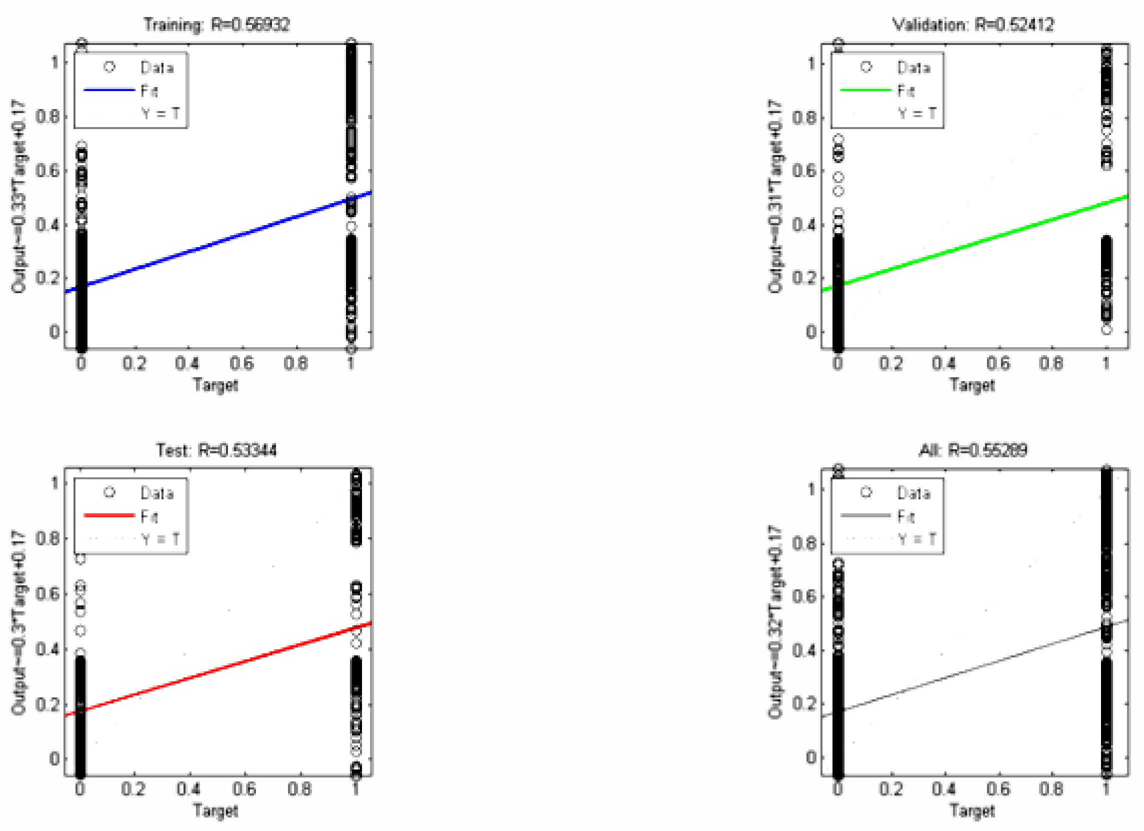
d) Scaled Conjugate Gradient



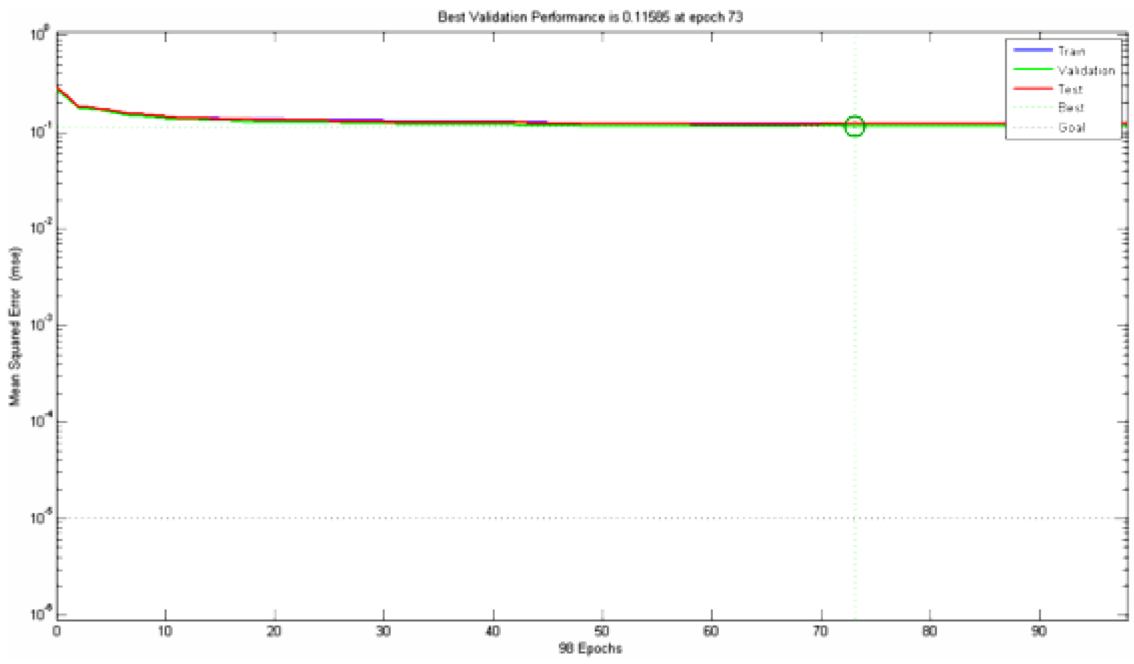


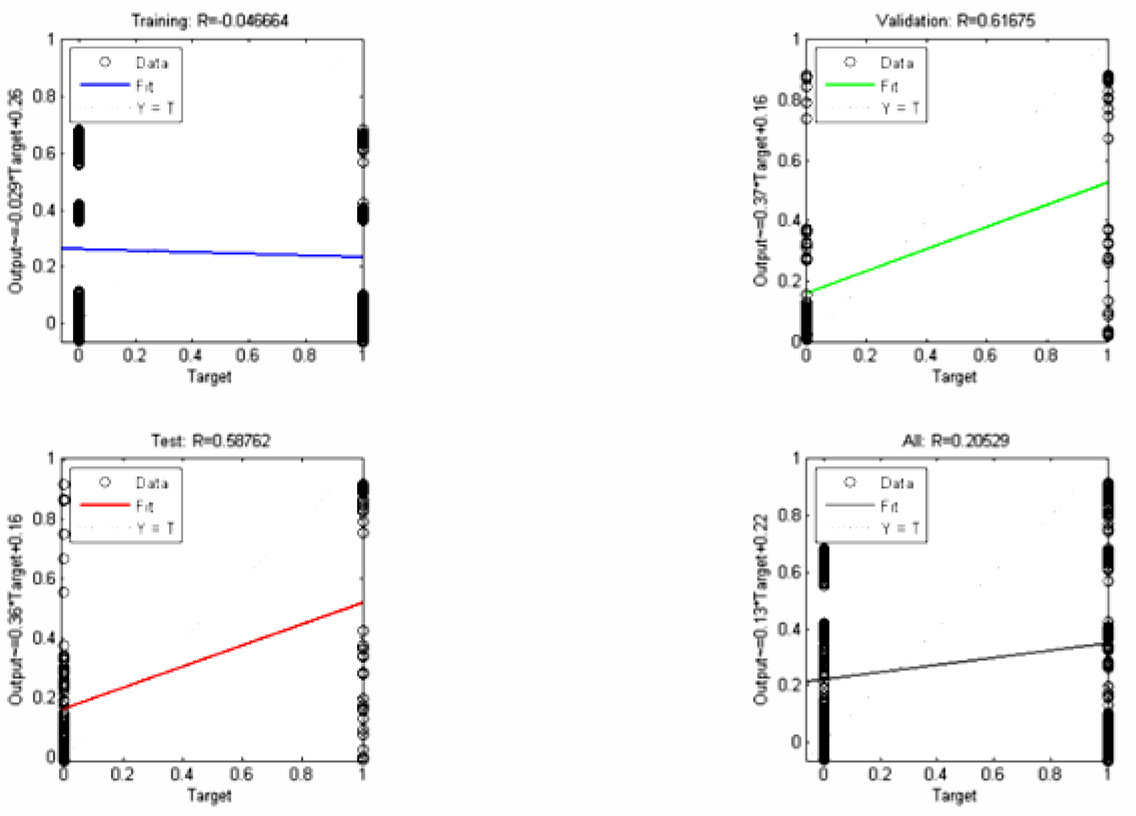
e) Variable Learning Rate Backpropagation





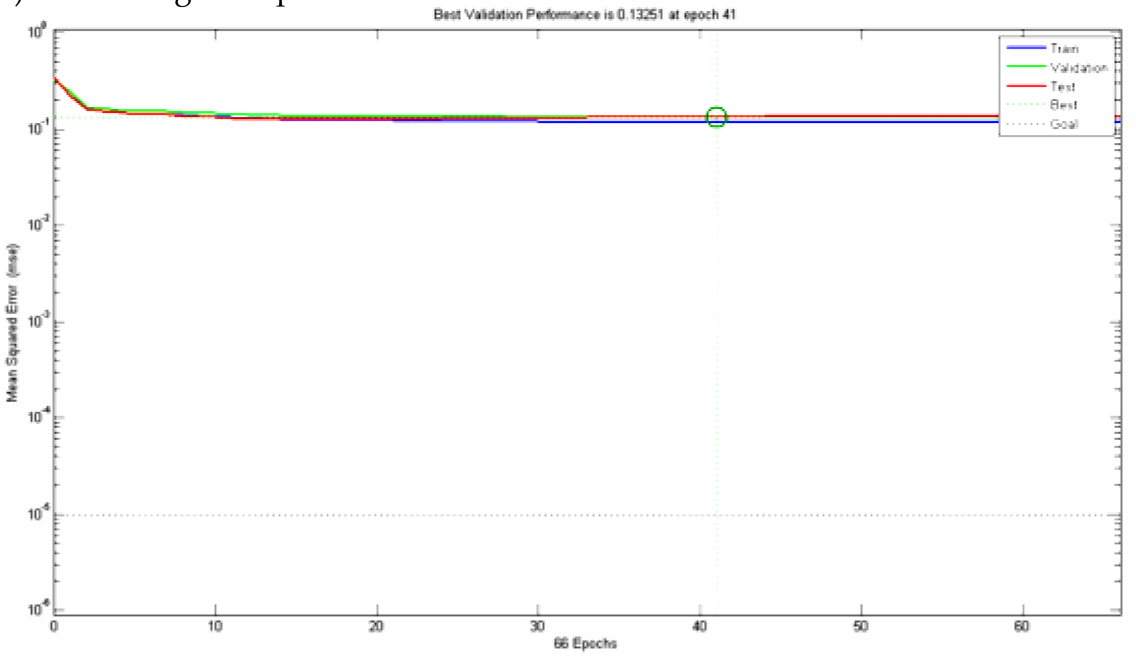
f) BFGS quasi–Newton Backpropagation

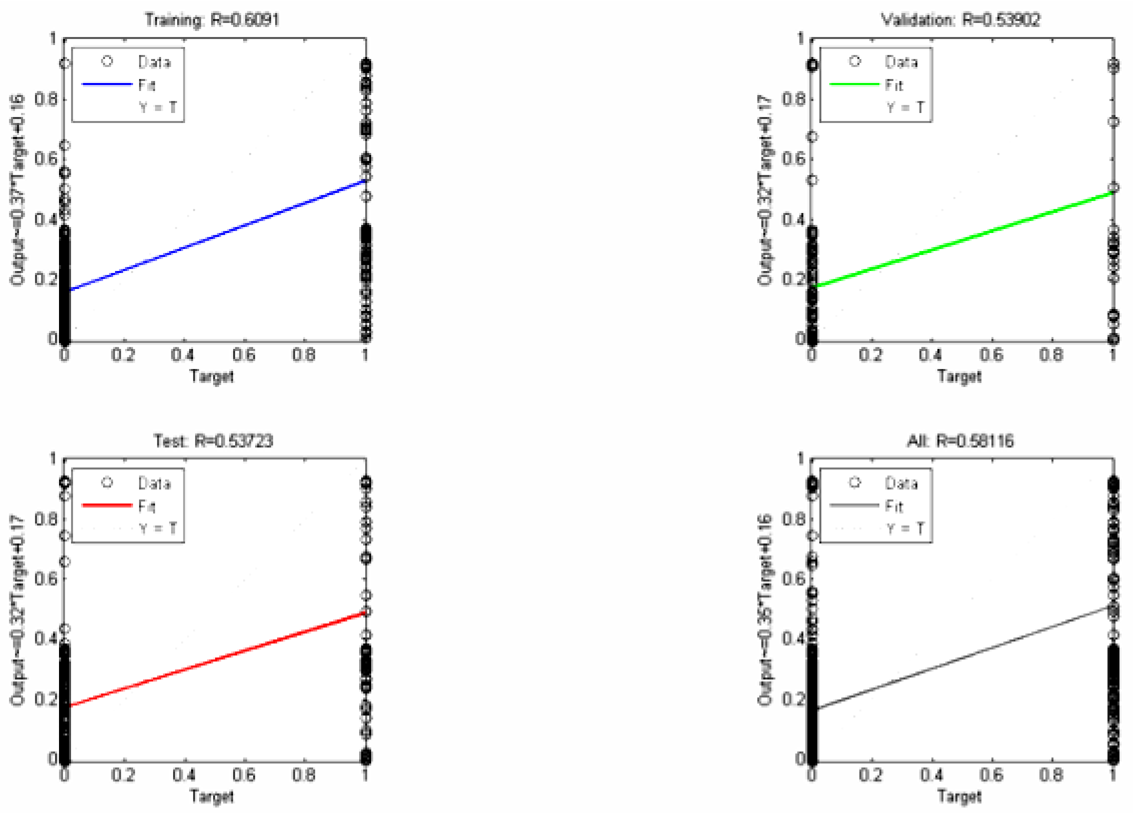




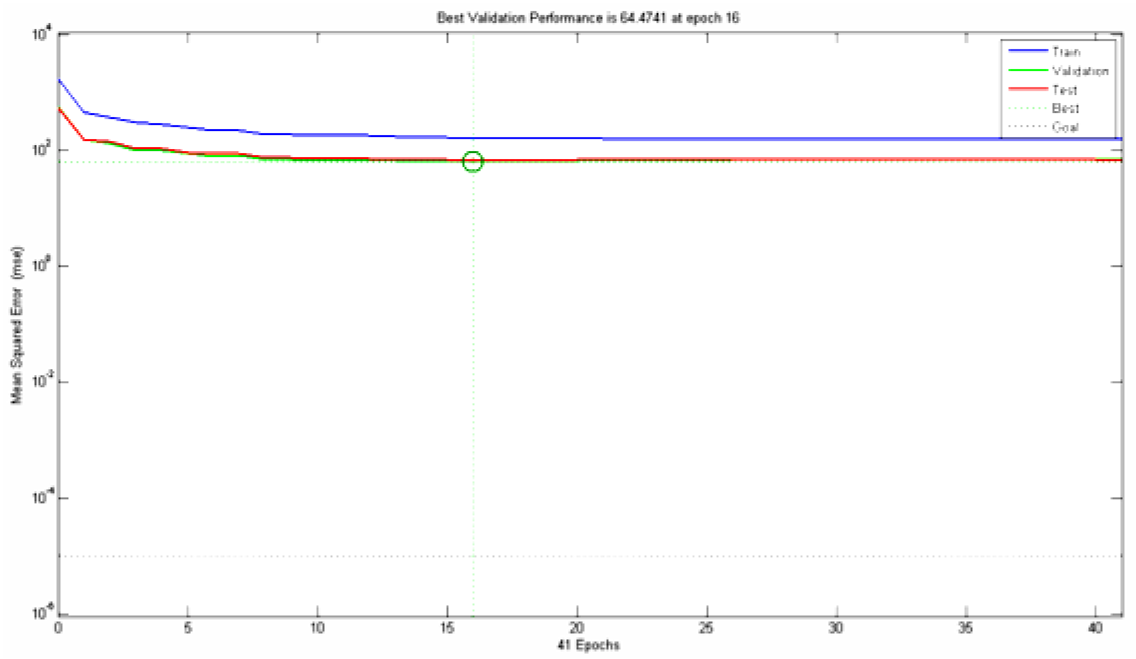
2) [2 2]

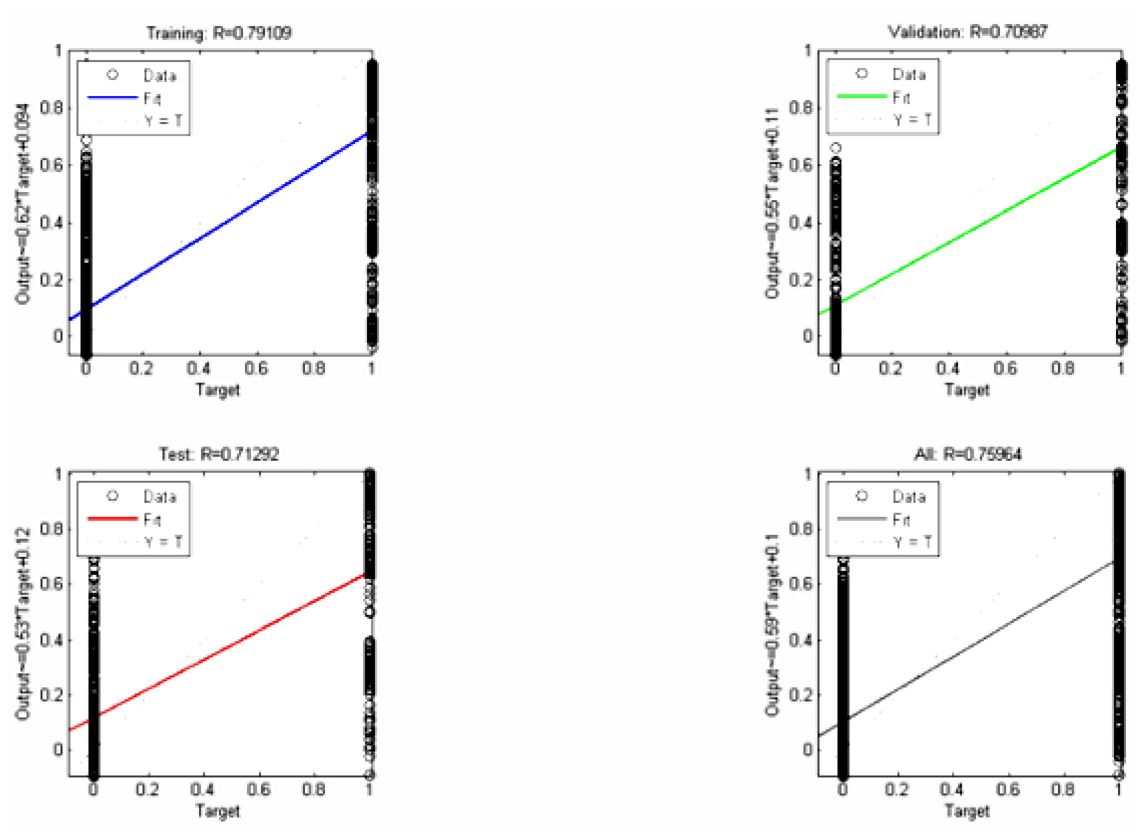
a) Levenberg–Marquardt



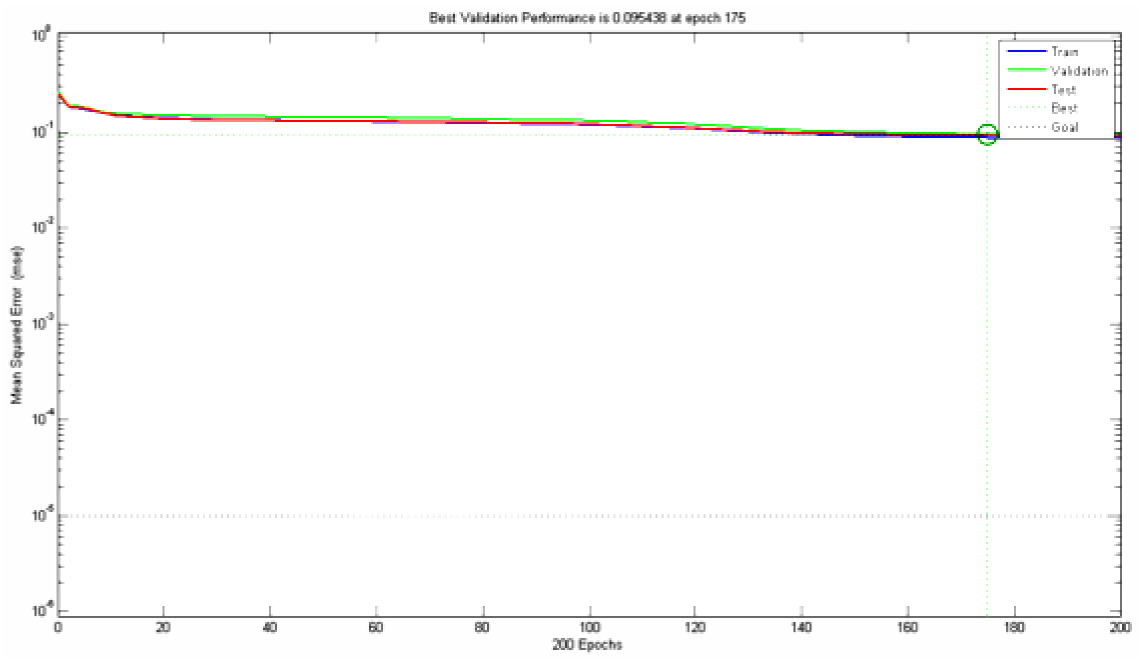


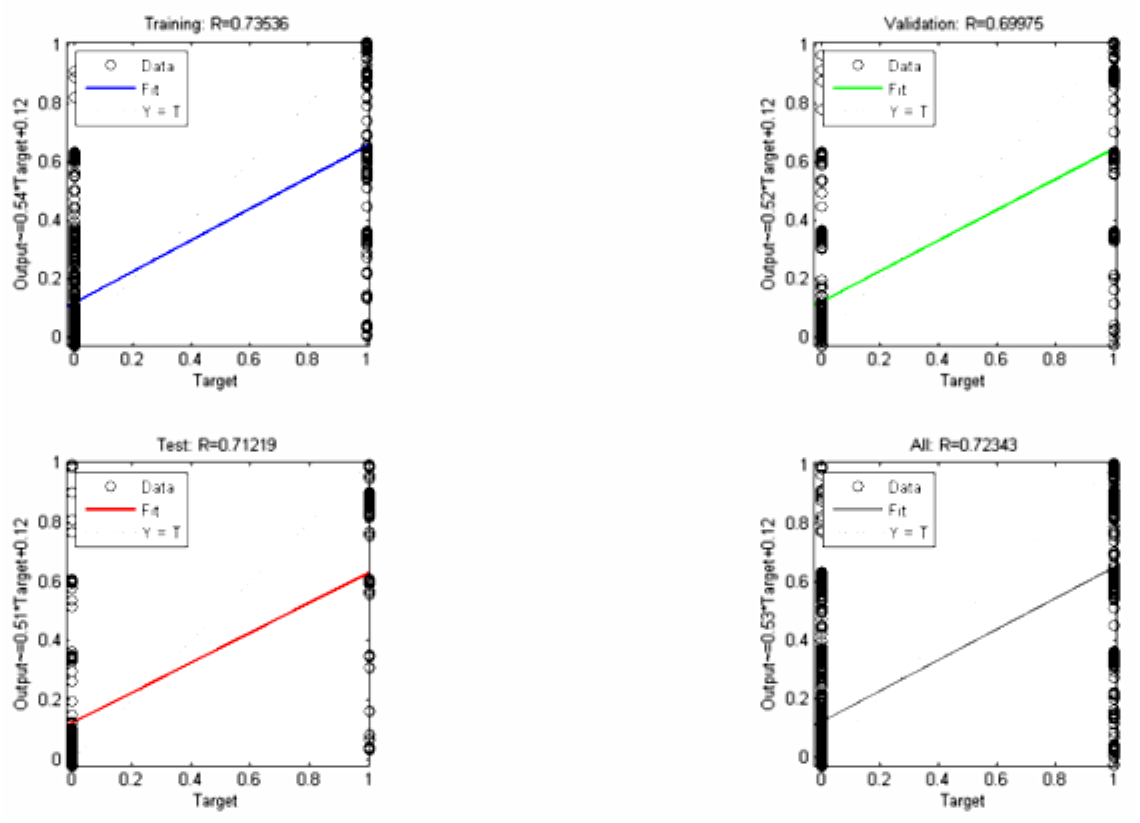
b) Bayesian Regularisation



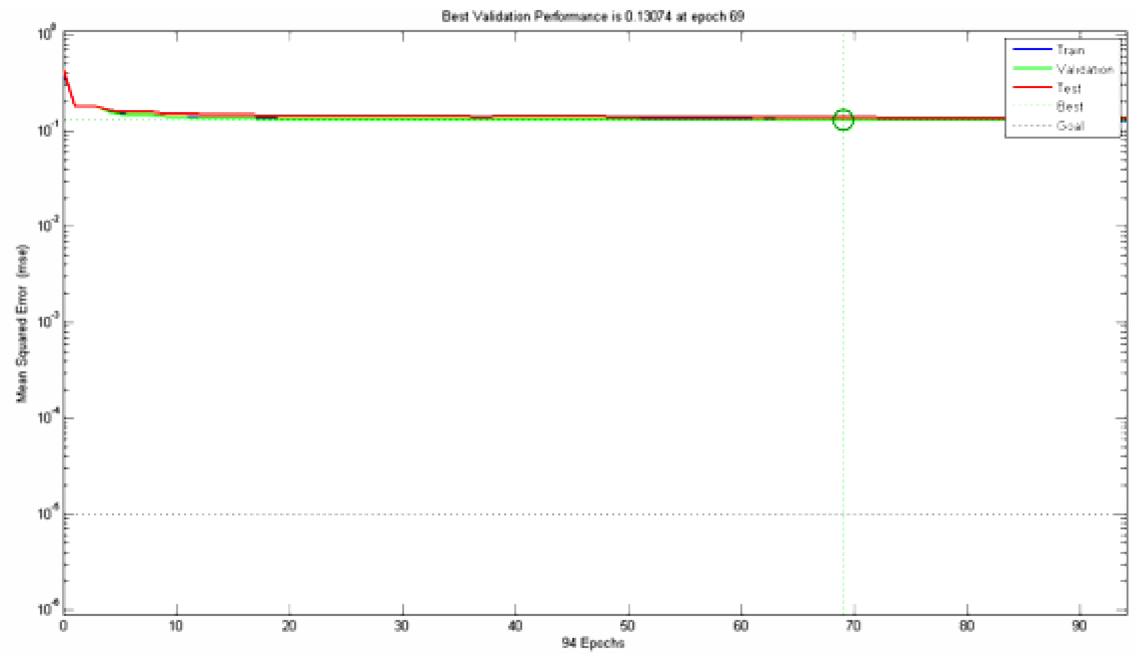


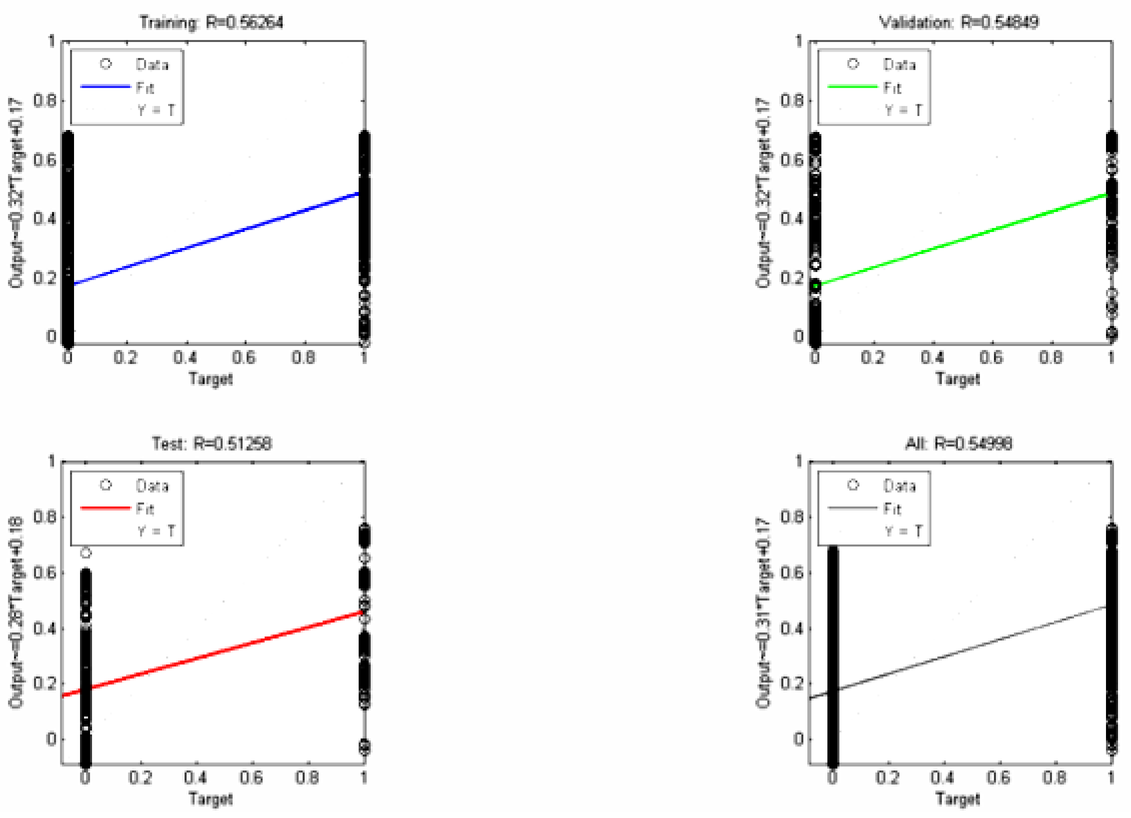
c) Resilient Backpropagation



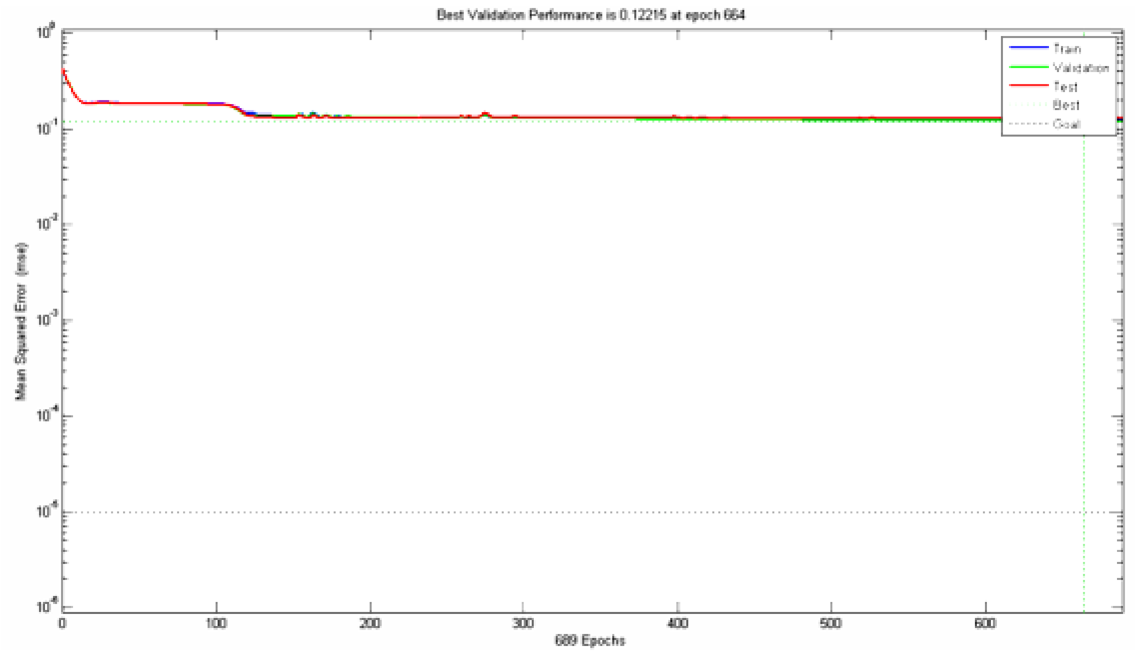


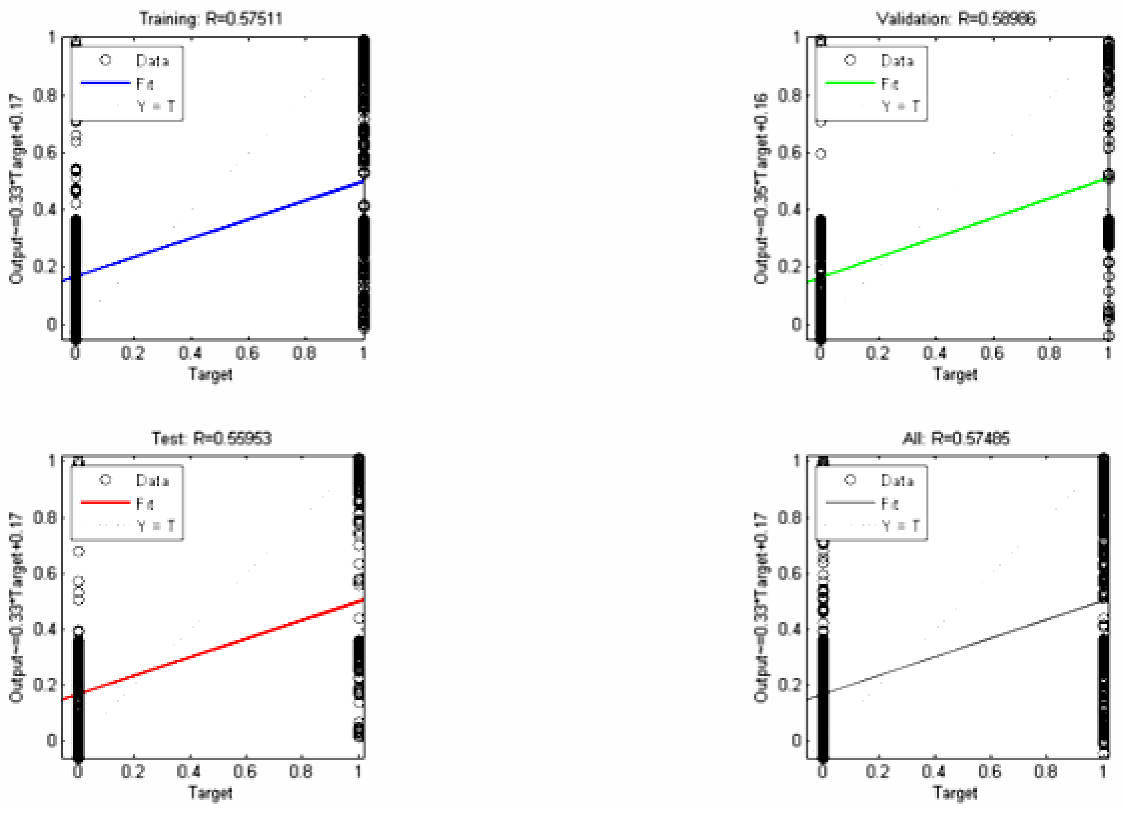
d) Scaled Conjugate Gradient



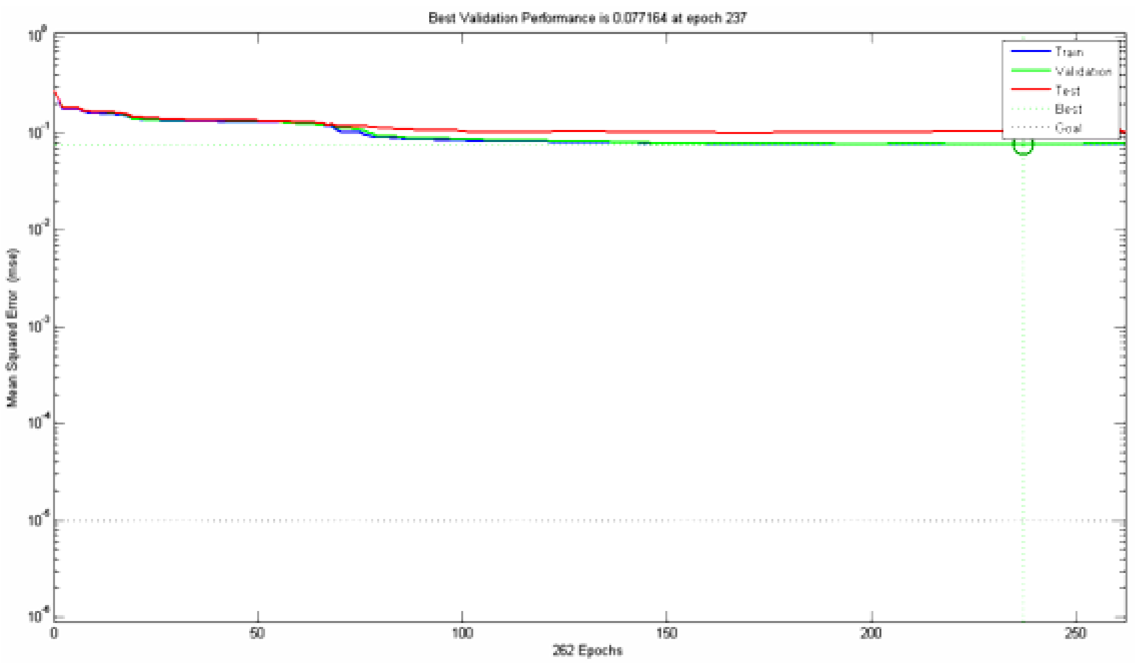


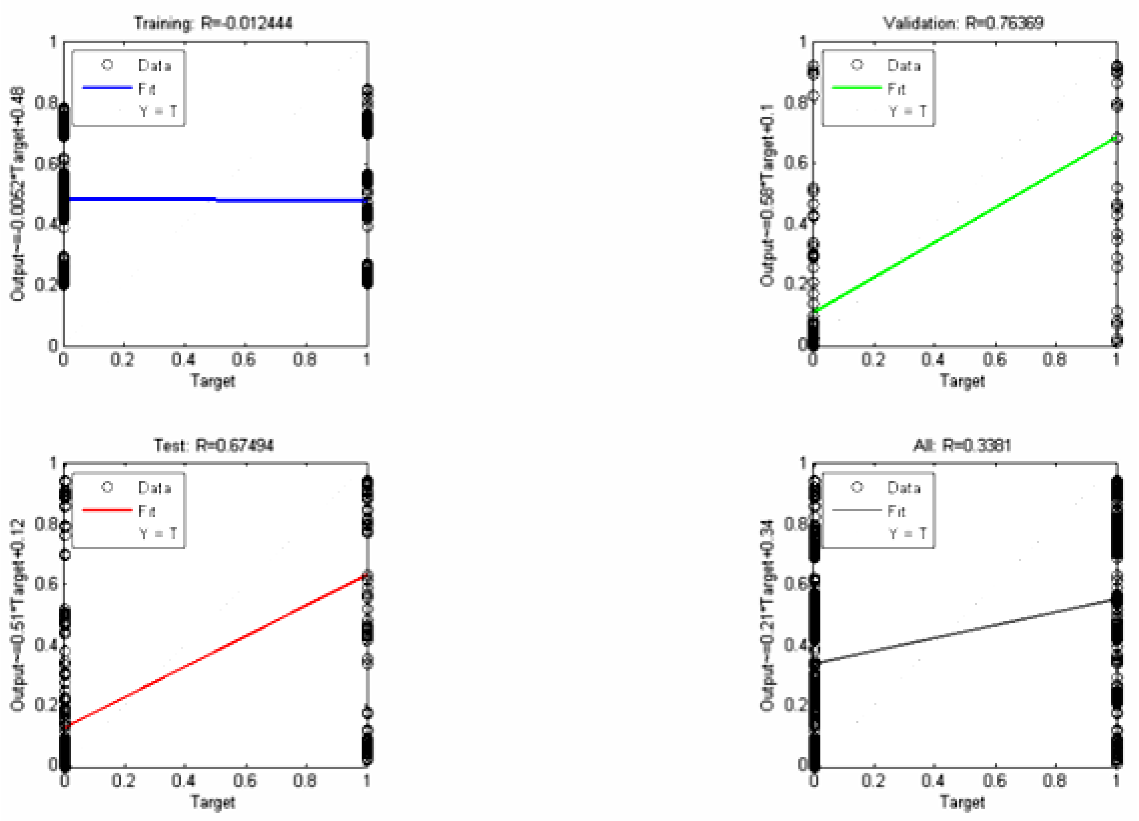
e) Variable Learning Rate Backpropagation





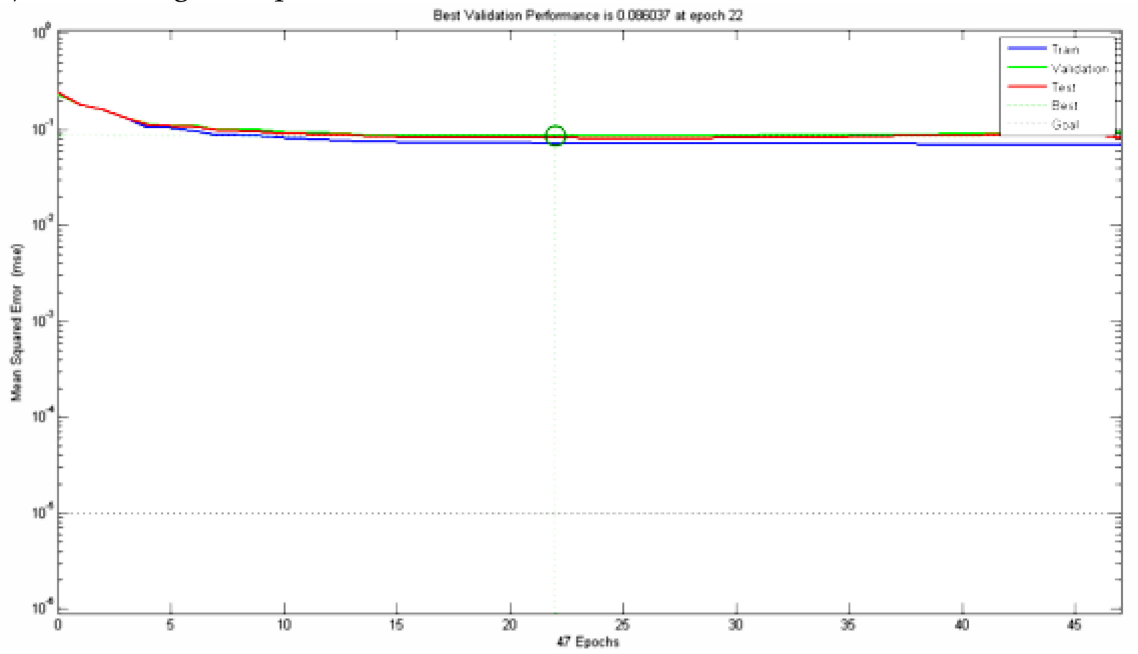
f) BFGS quasi–Newton Backpropagation

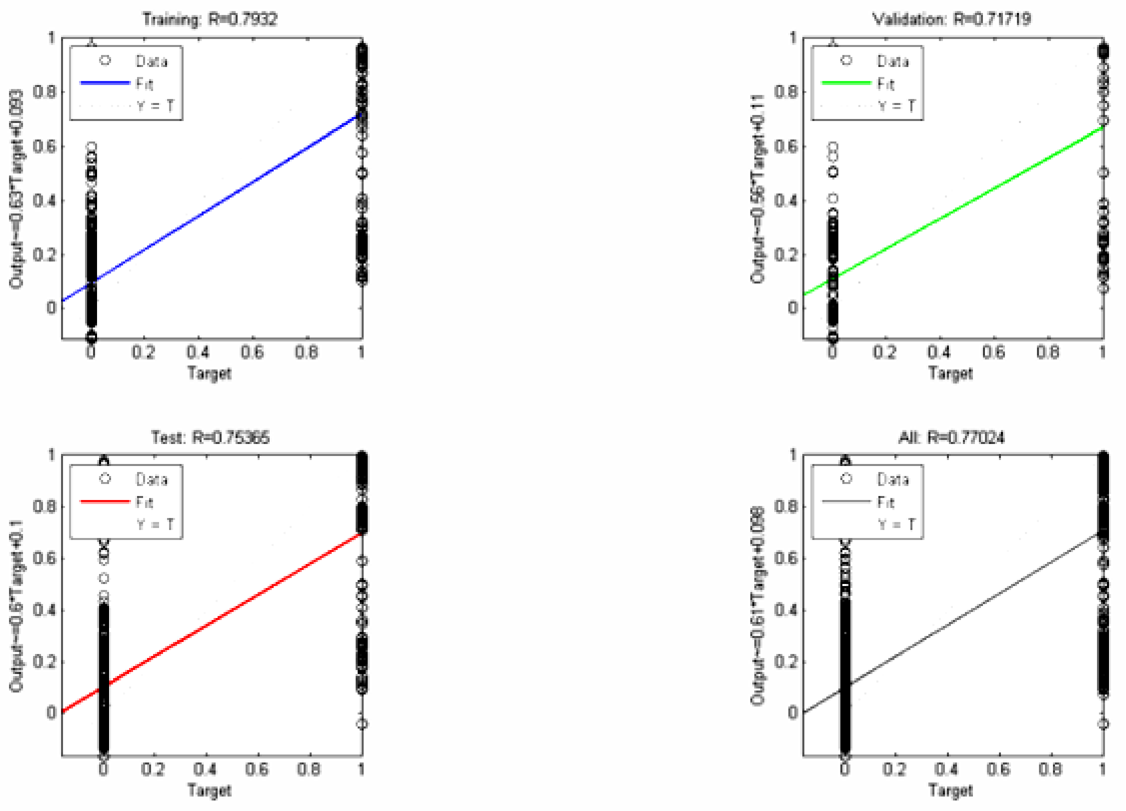




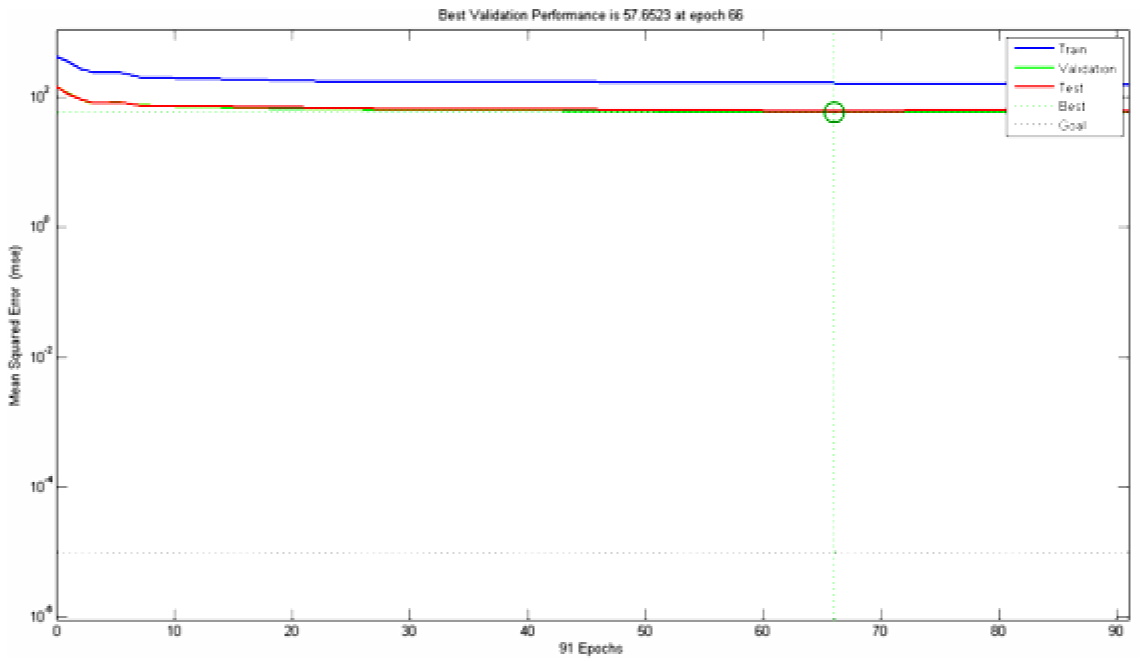
3) [3 2]

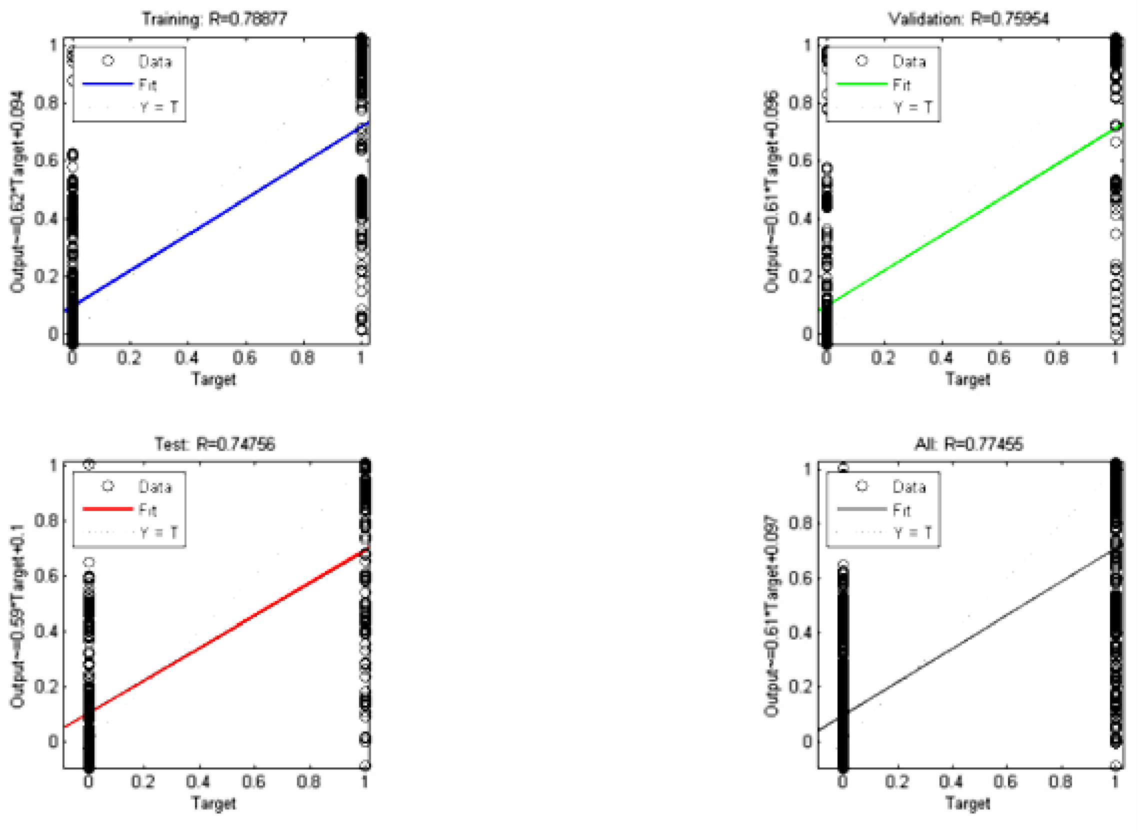
a) Levenberg–Marquardt



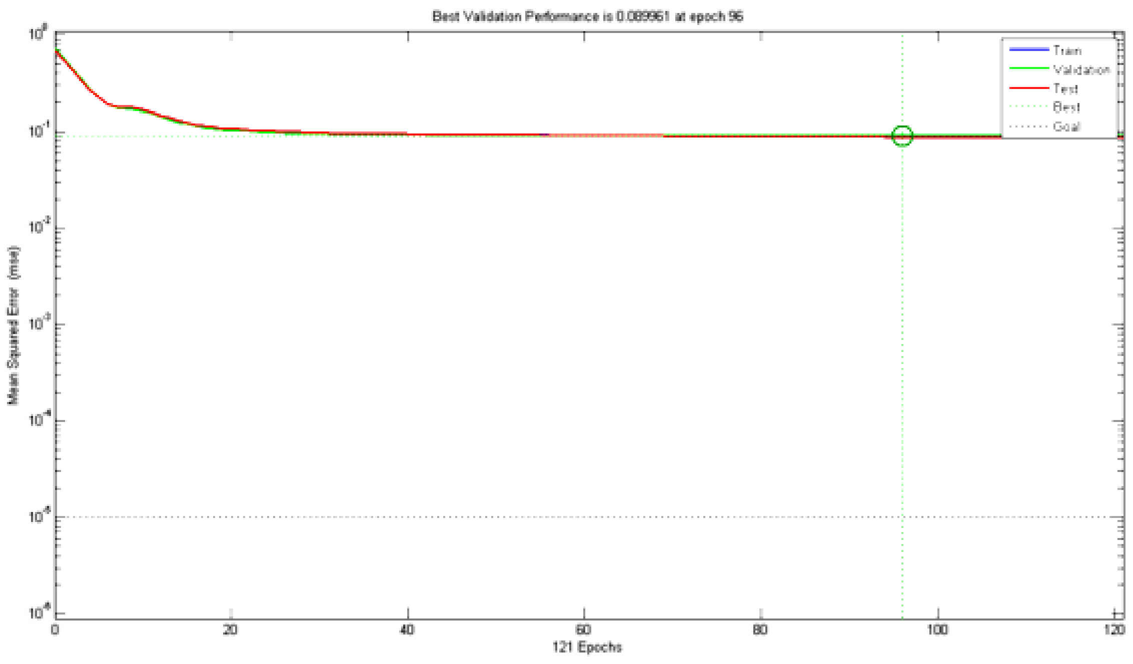


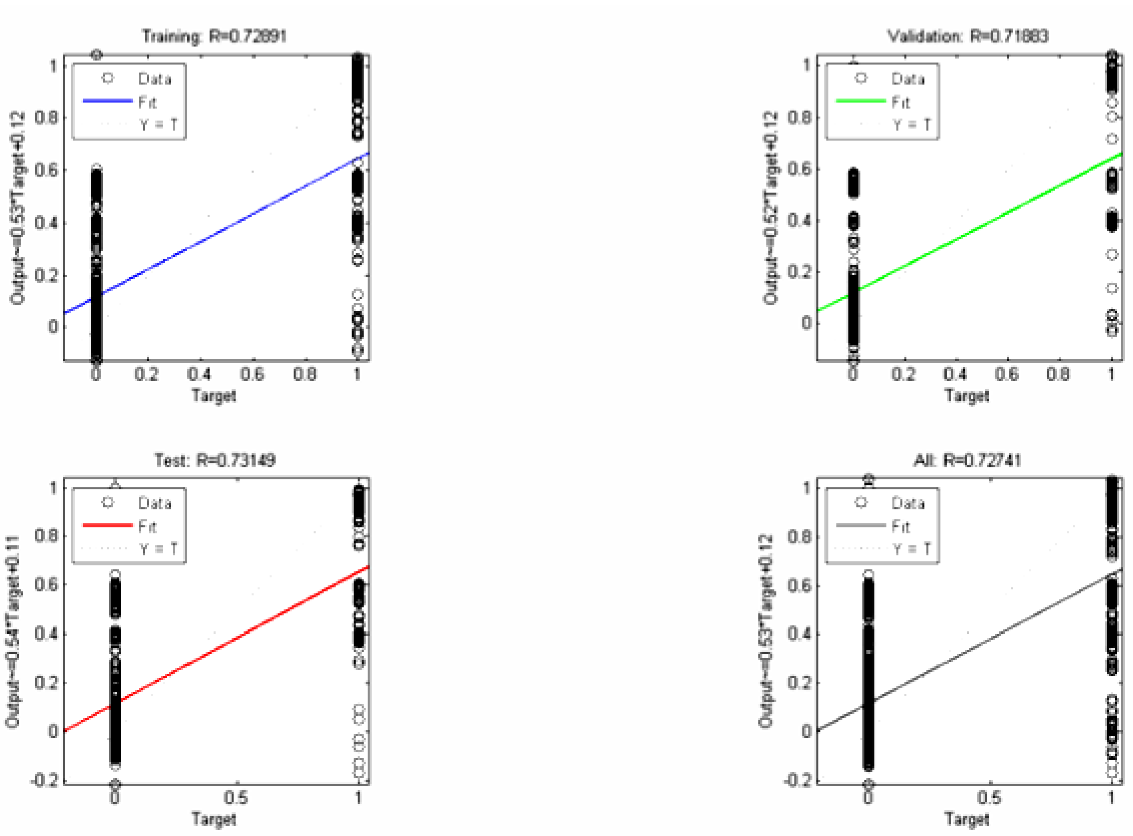
b) Bayesian Regularisation



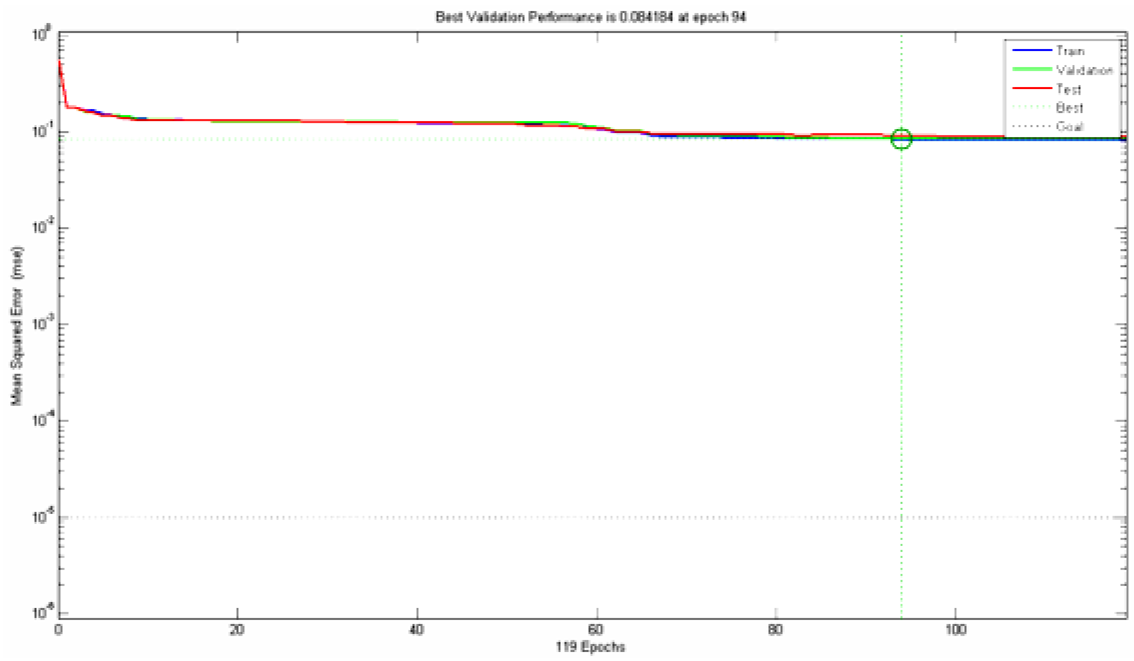


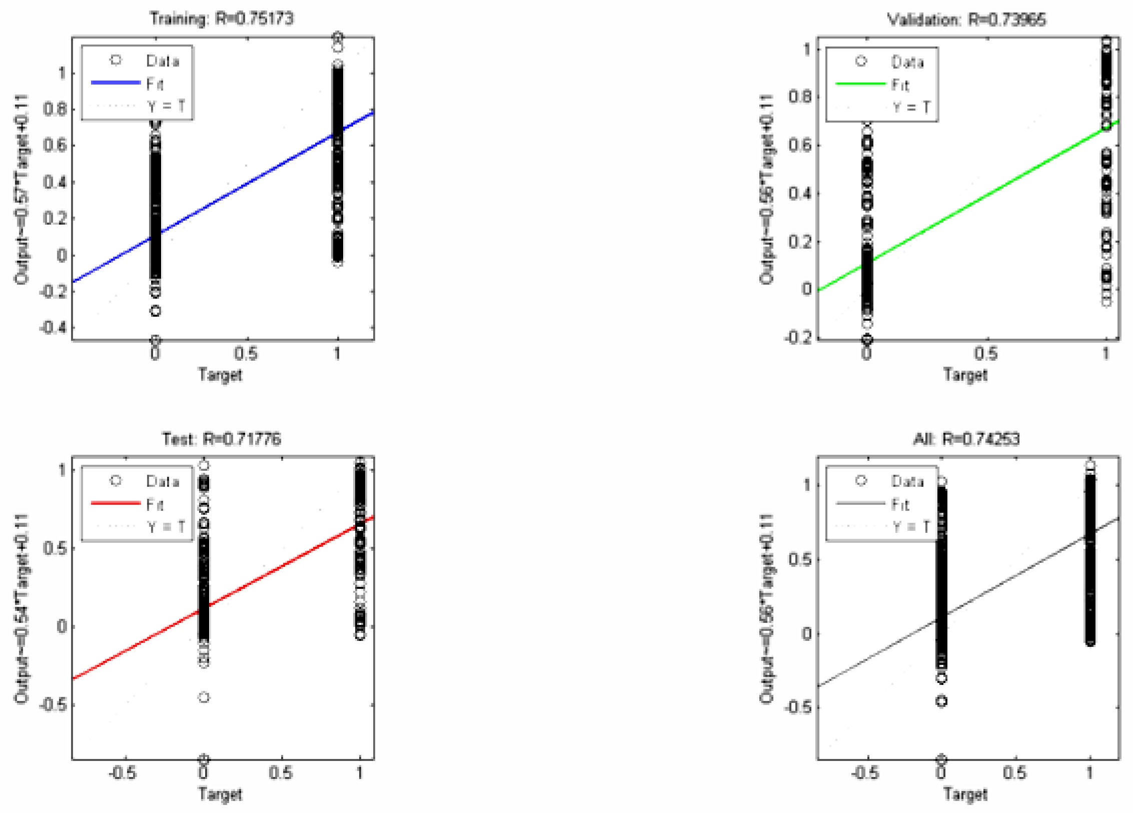
c) Resilient Backpropagation



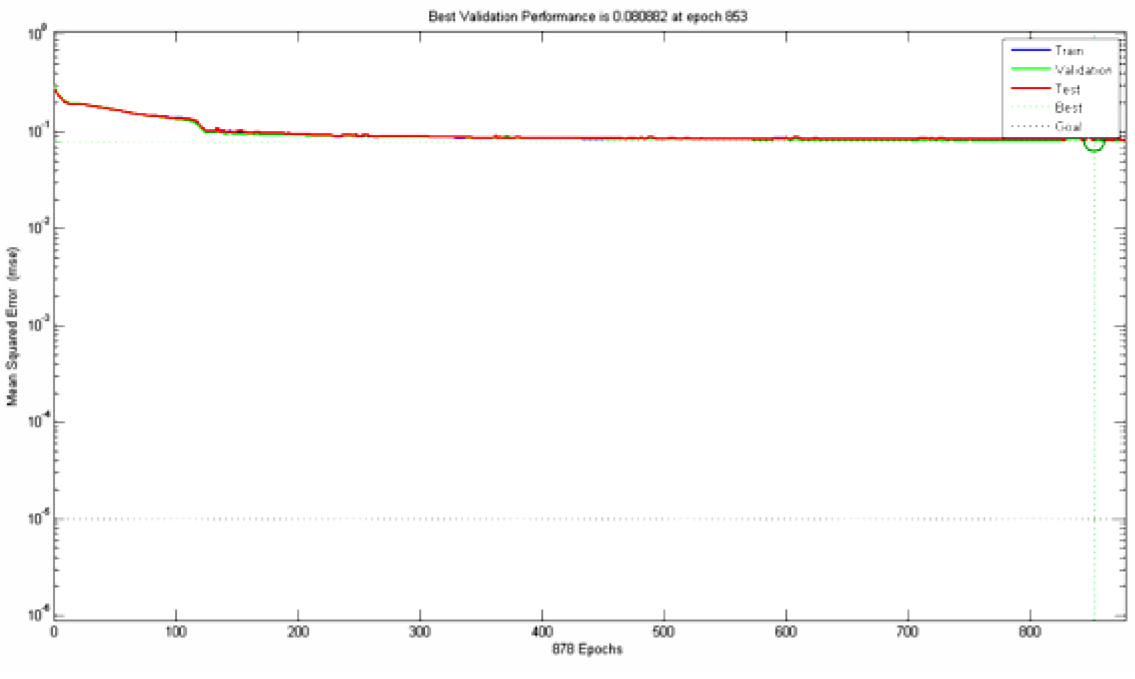


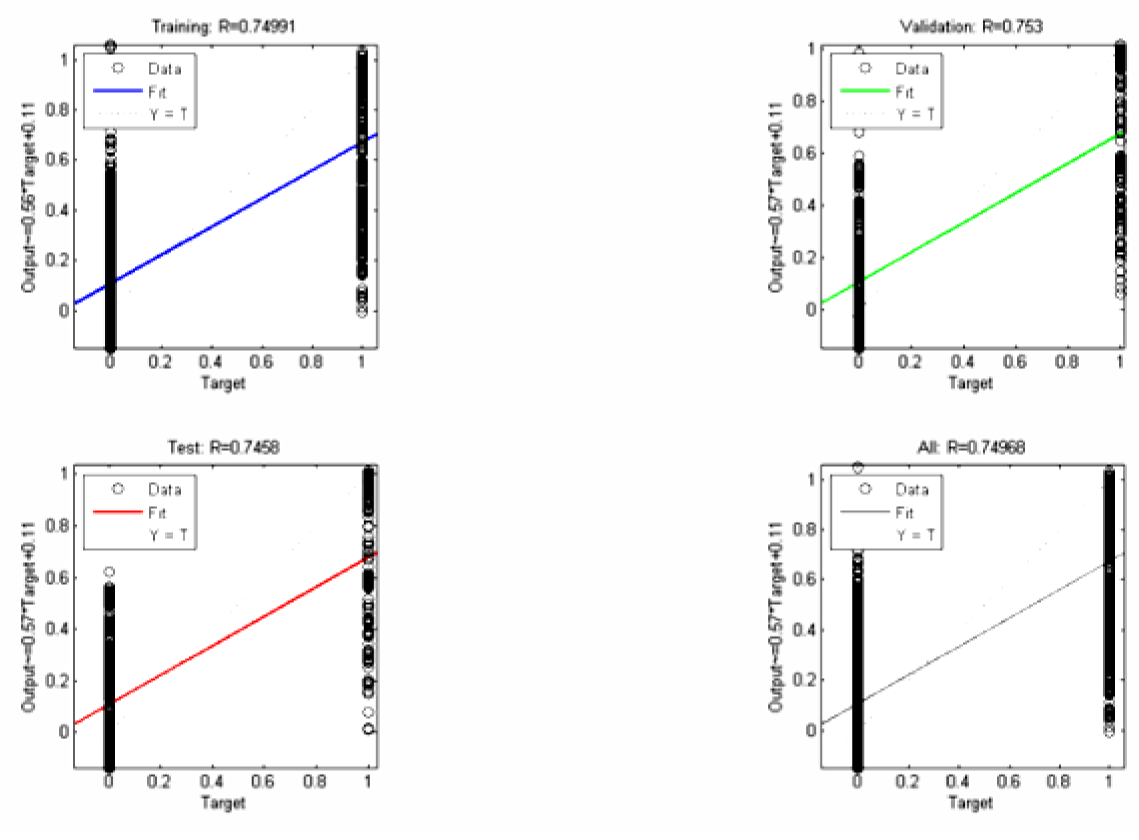
d) Scaled Conjugate Gradient



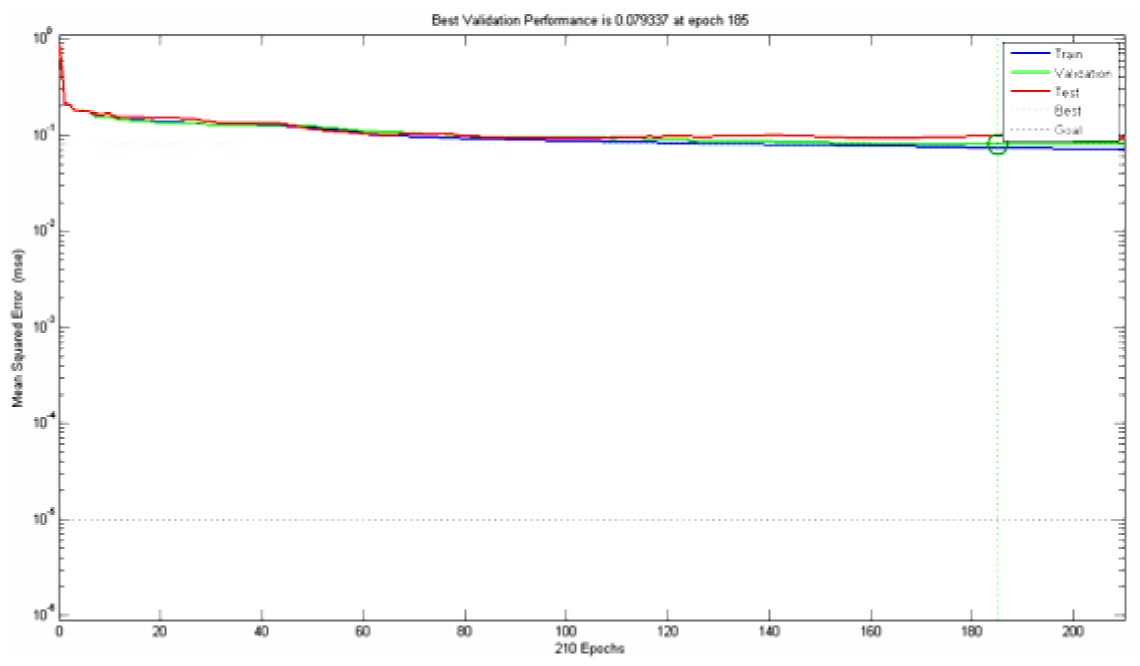


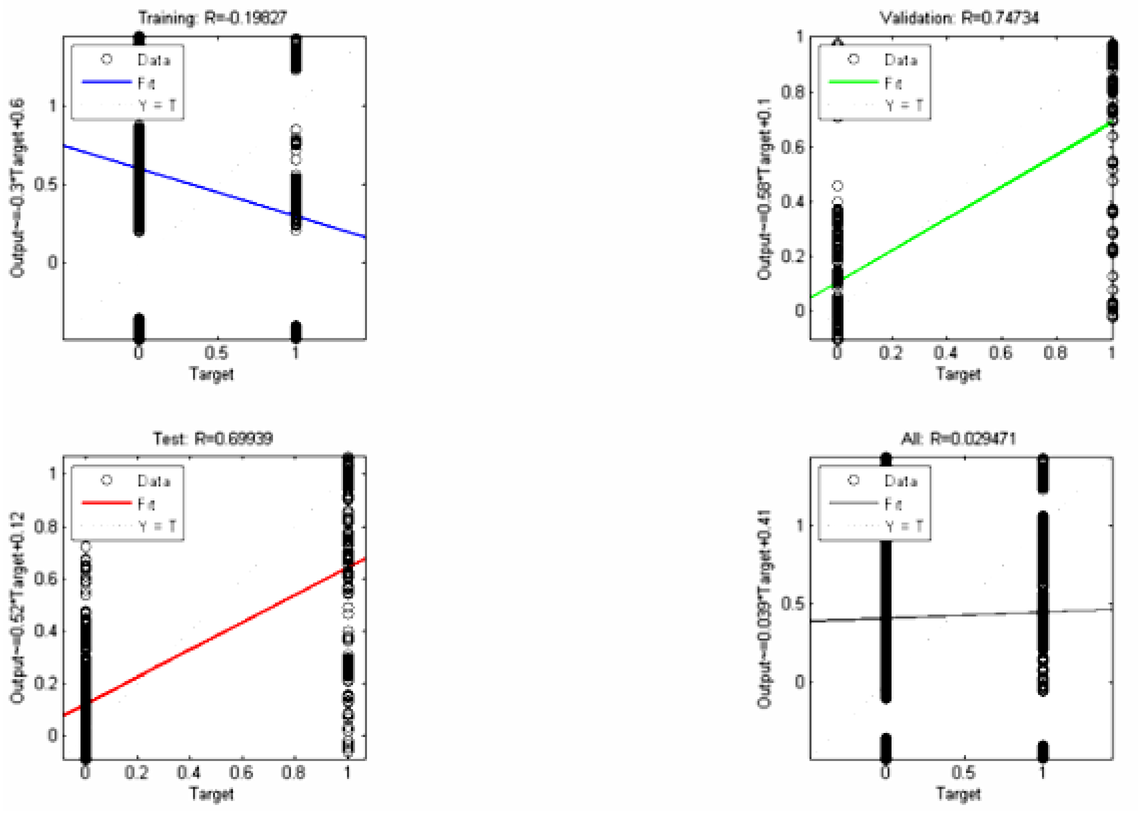
e) Variable Learning Rate Backpropagation





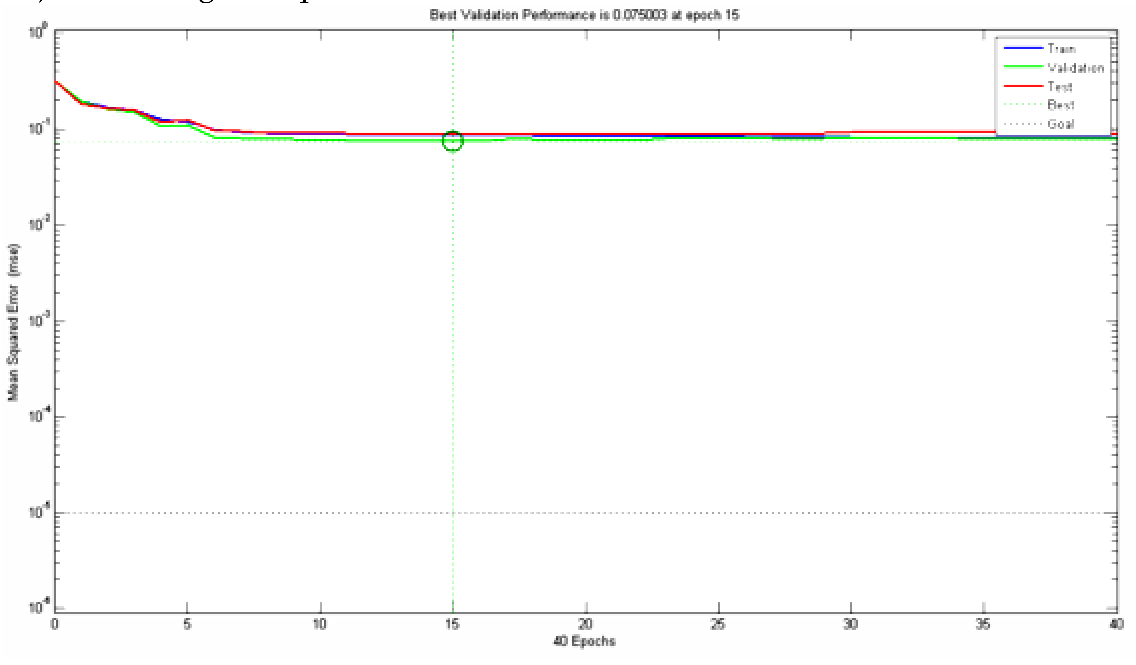
f) BFGS quasi–Newton Backpropagation

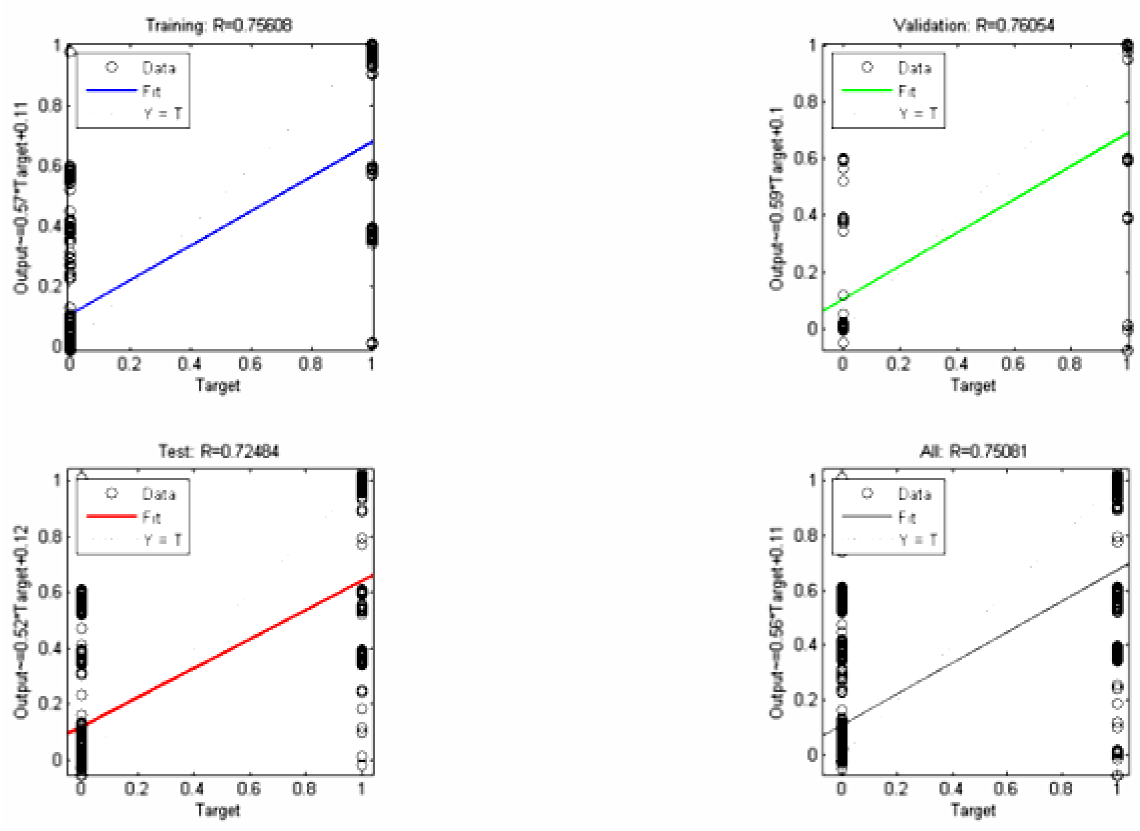




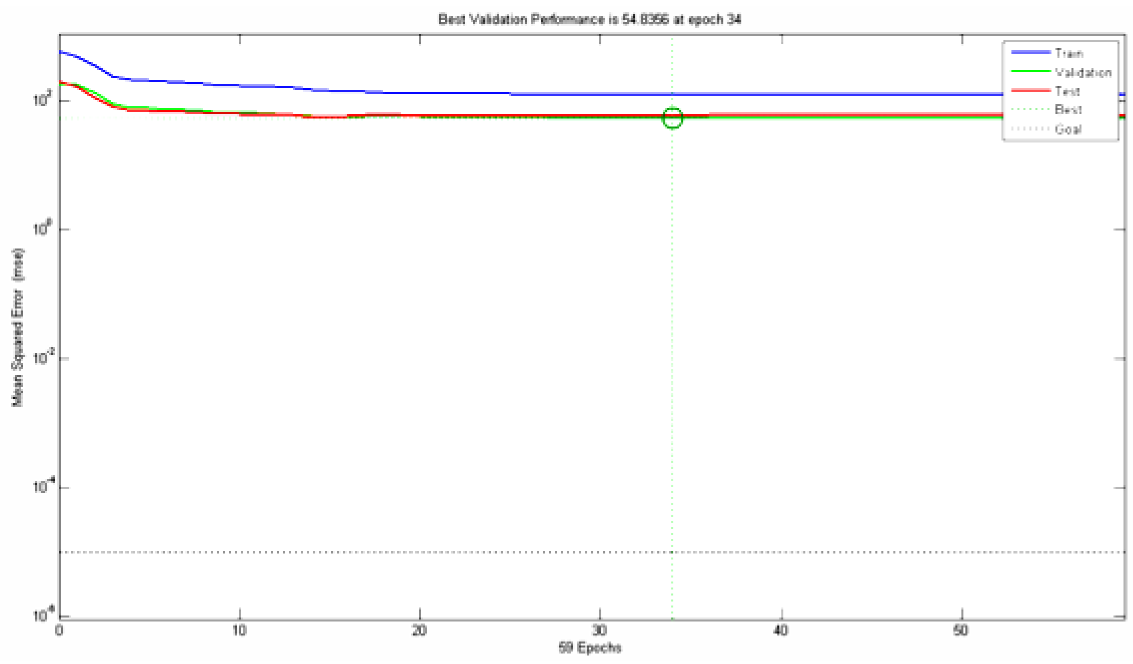
4) [5 2]

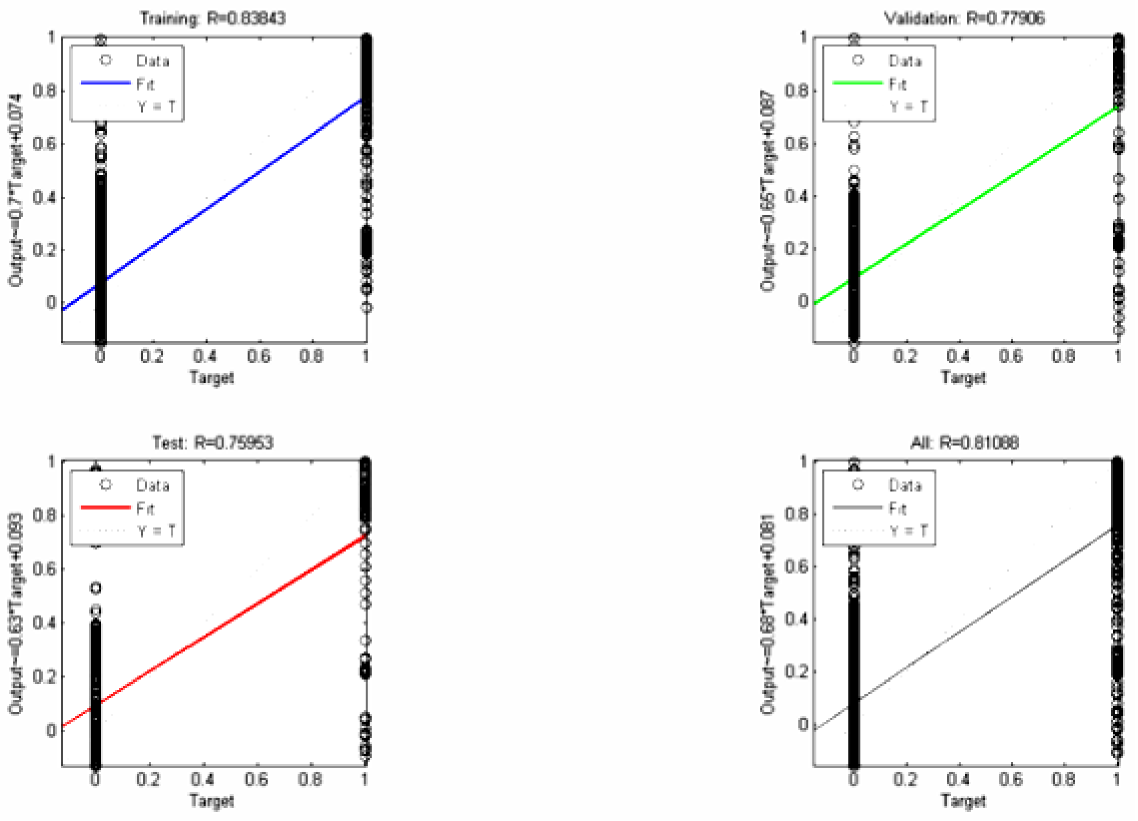
a) Levenberg–Marquardt



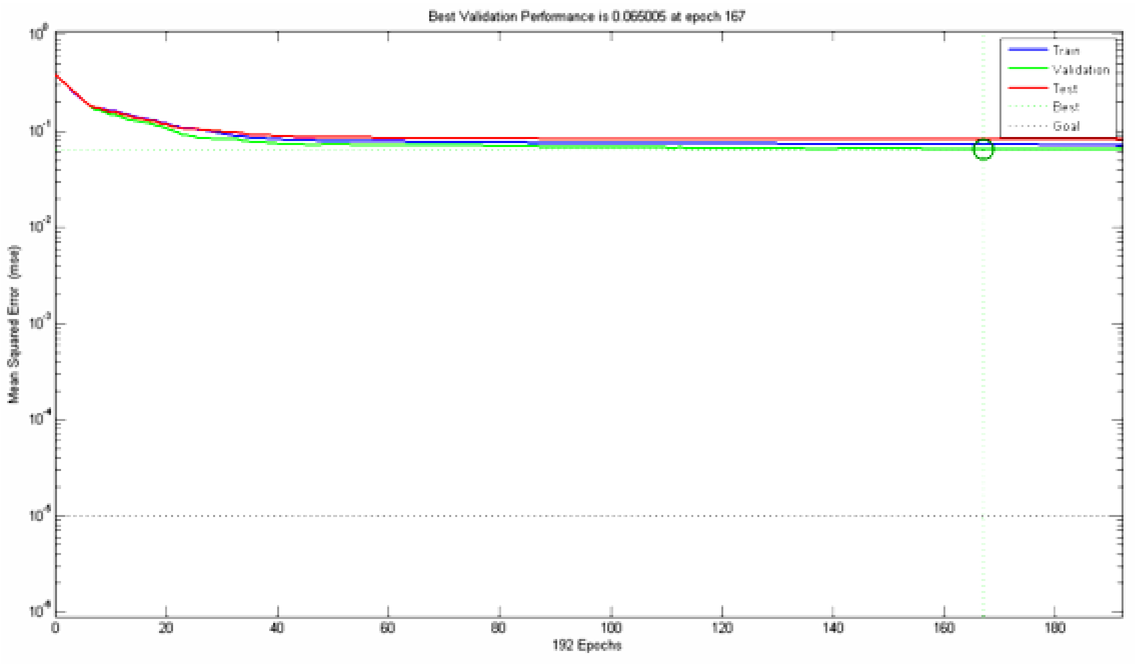


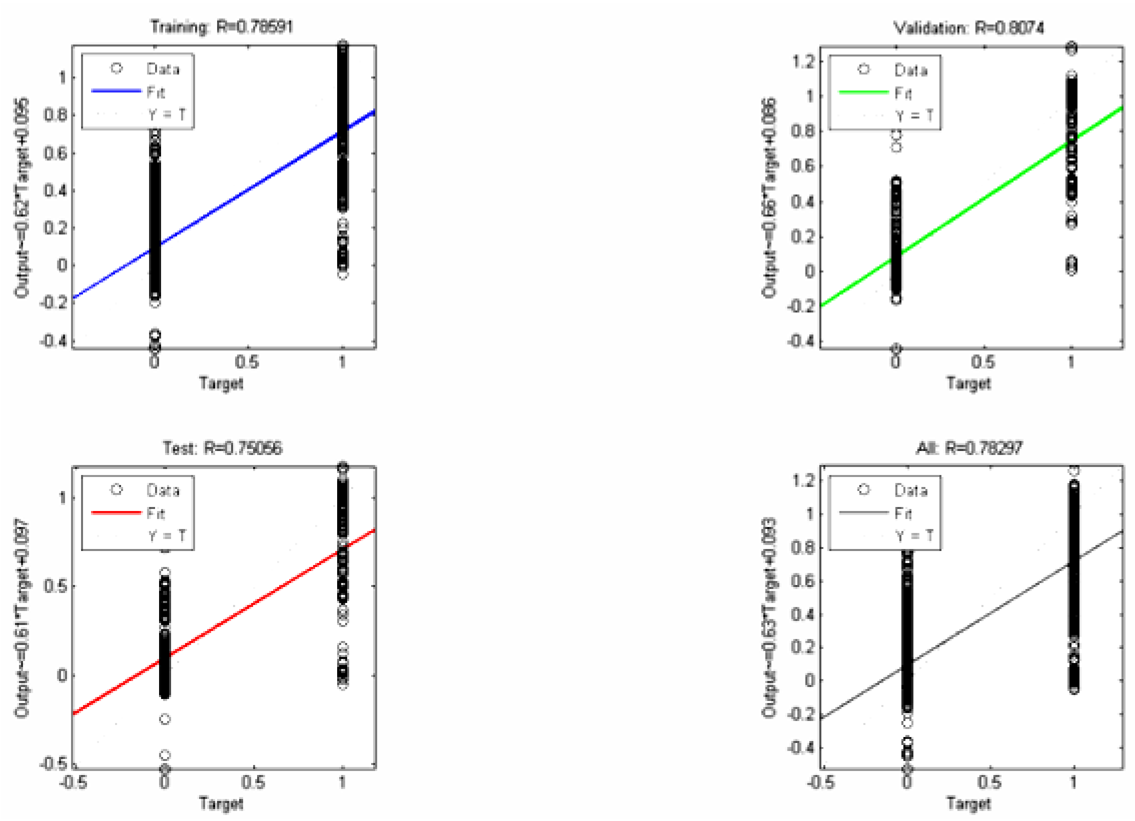
b) Bayesian Regularisation



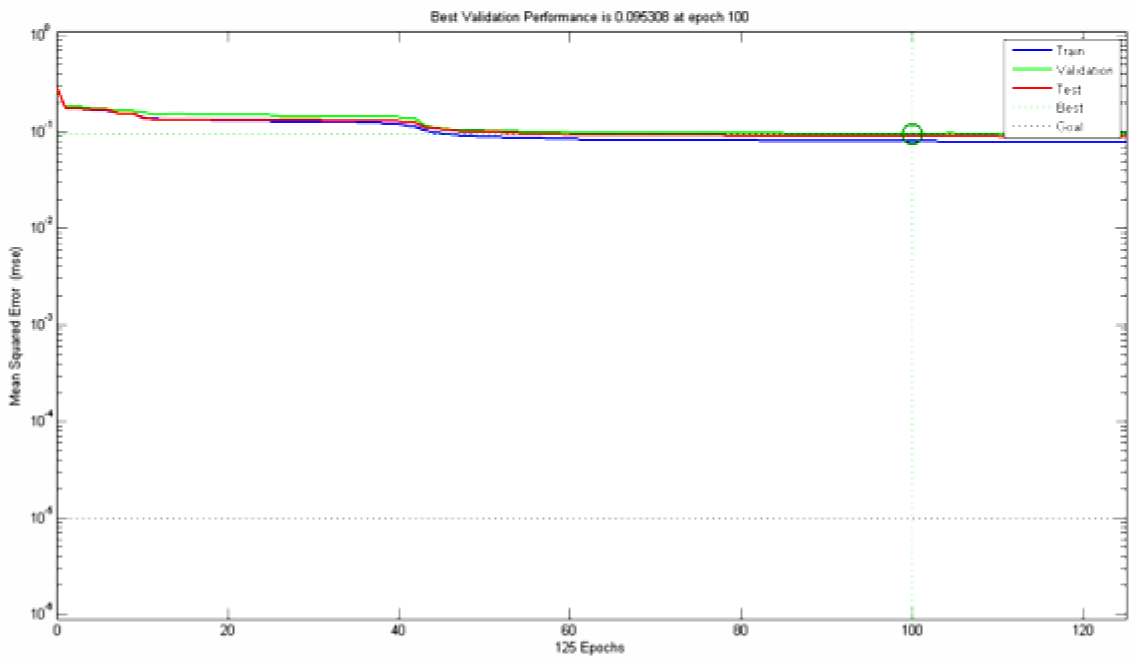


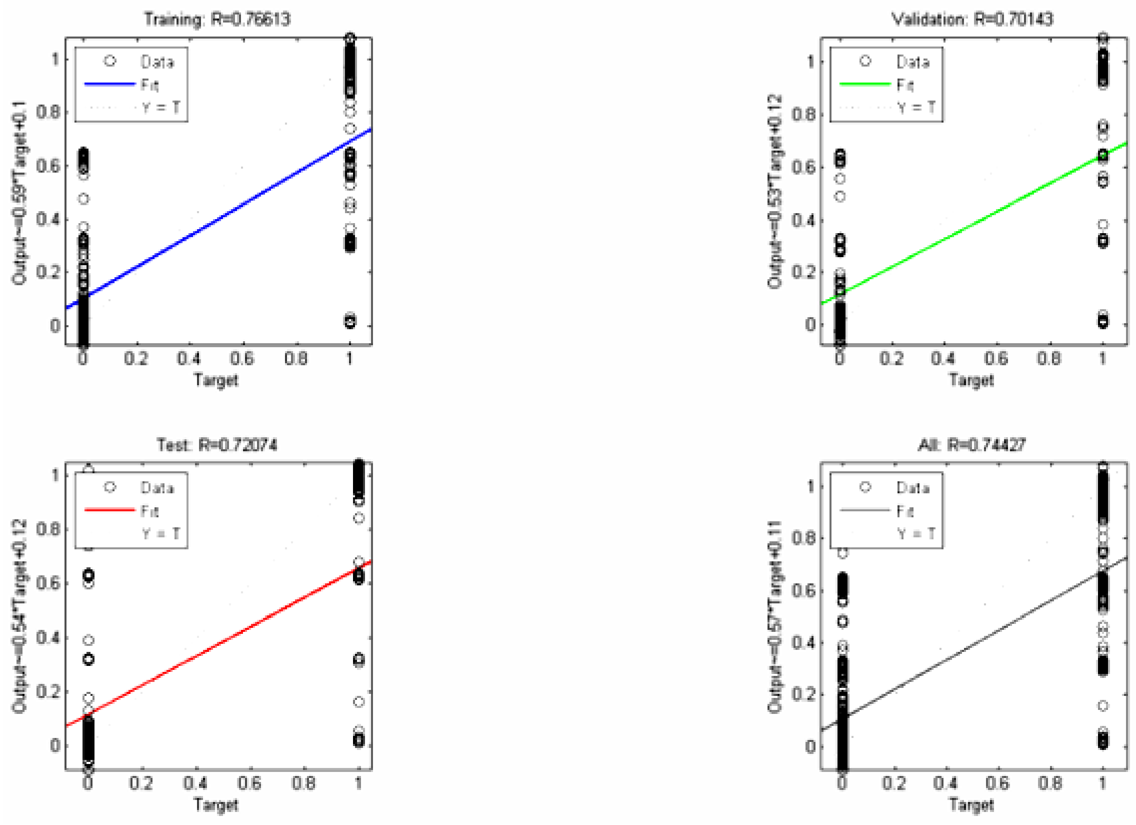
c) Resilient Backpropagation



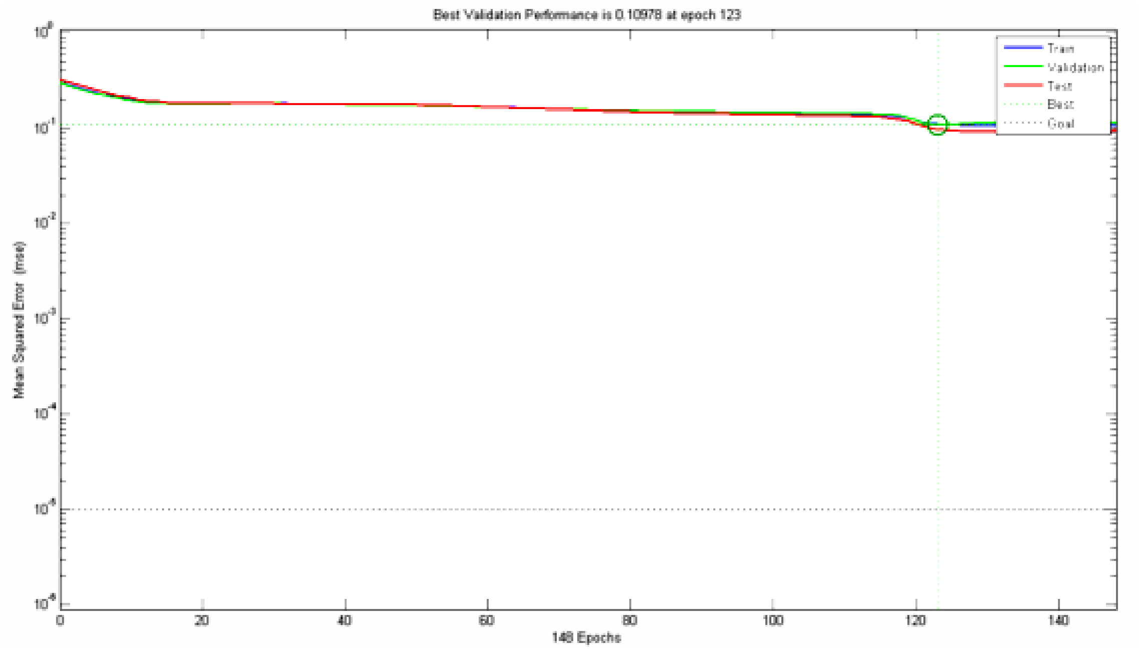


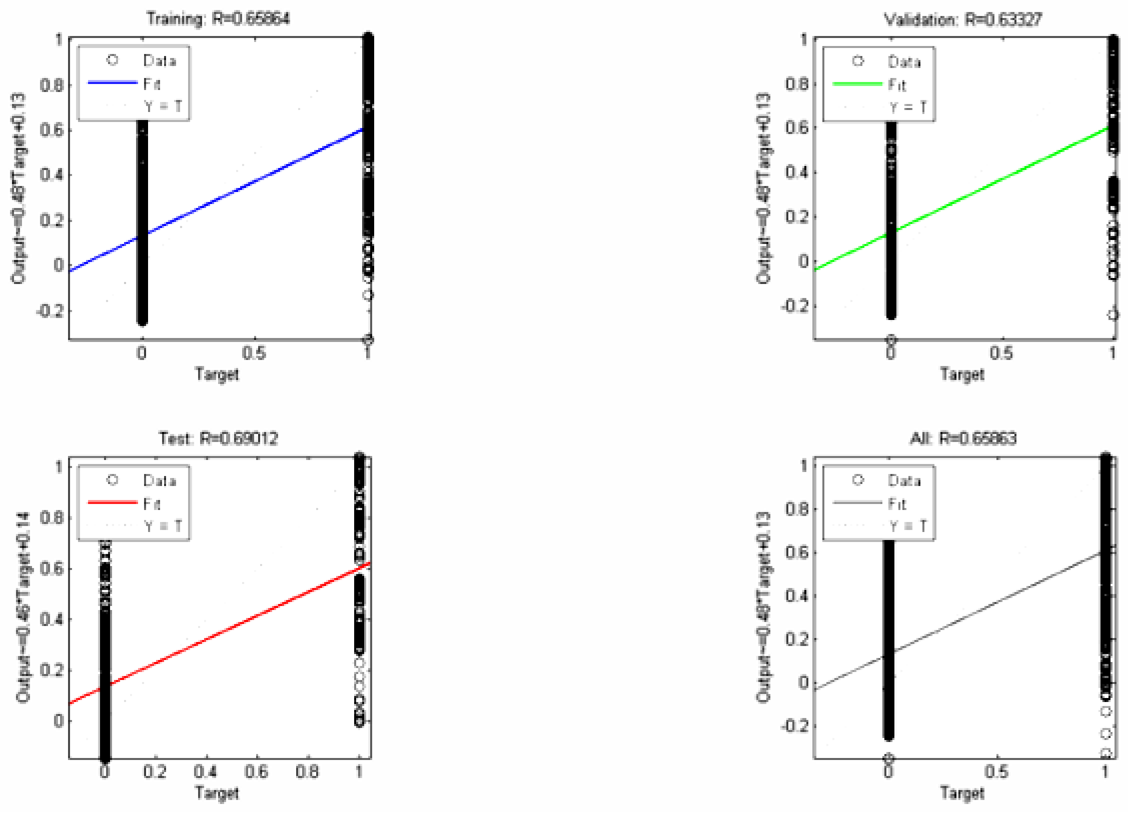
d) Scaled Conjugate Gradient



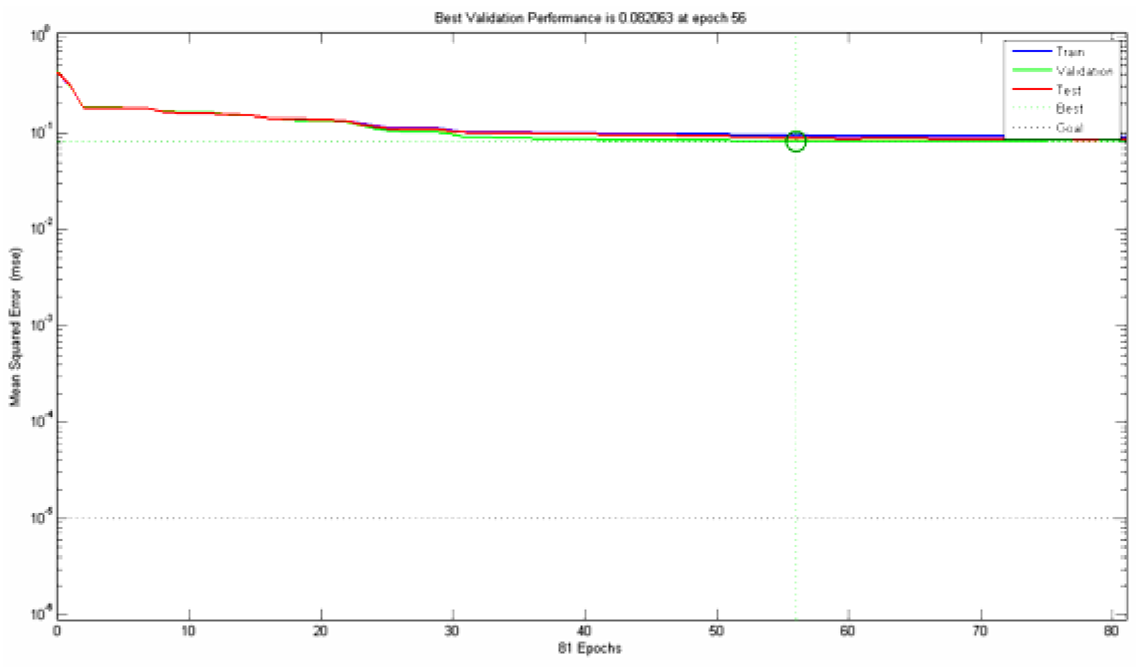


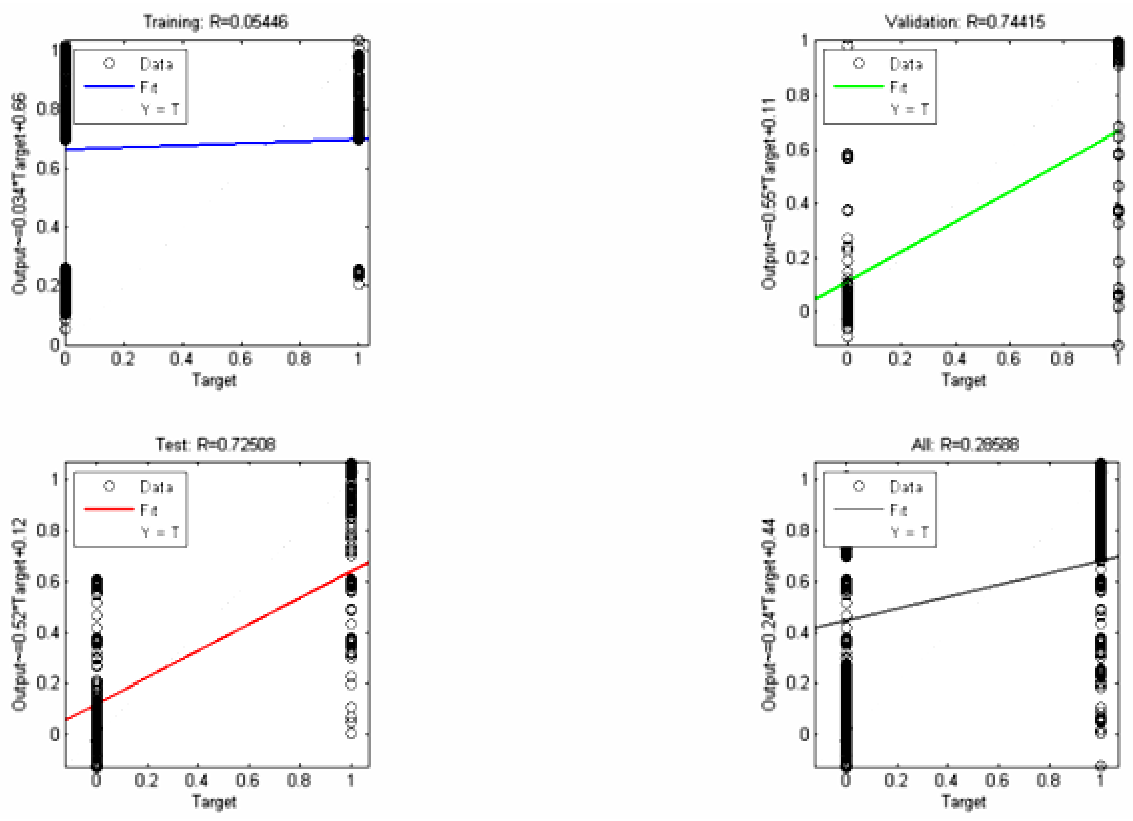
e) Variable Learning Rate Backpropagation





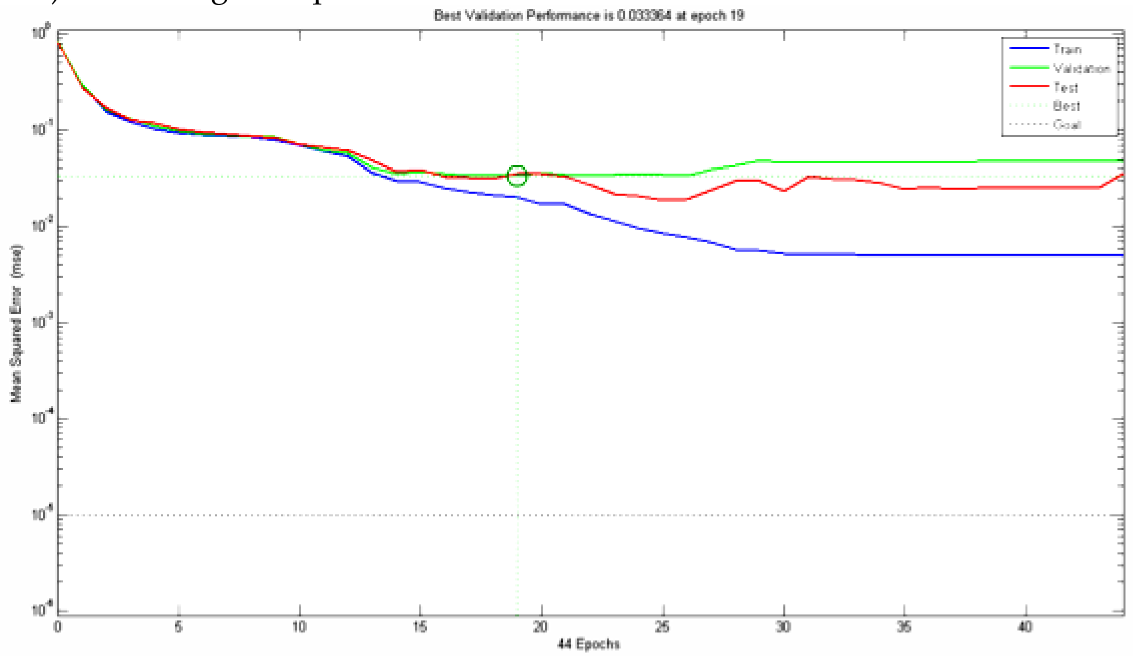
f) BFGS quasi–Newton Backpropagation

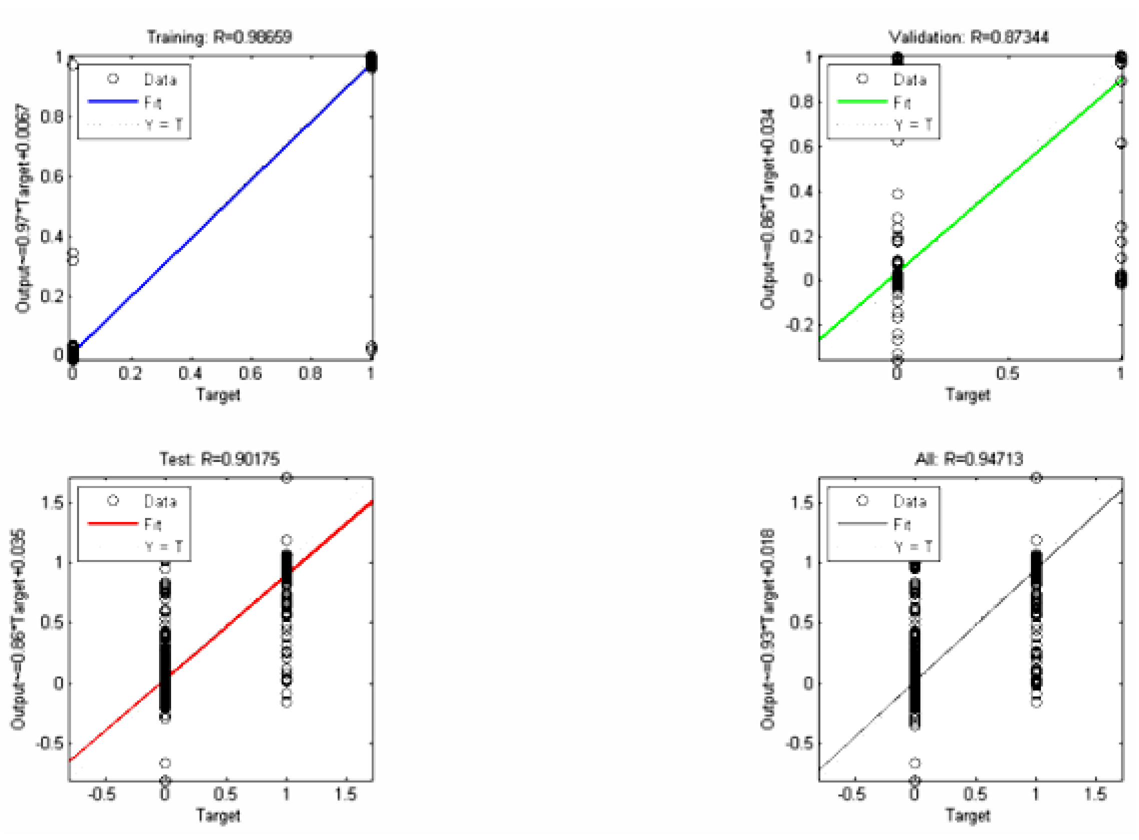




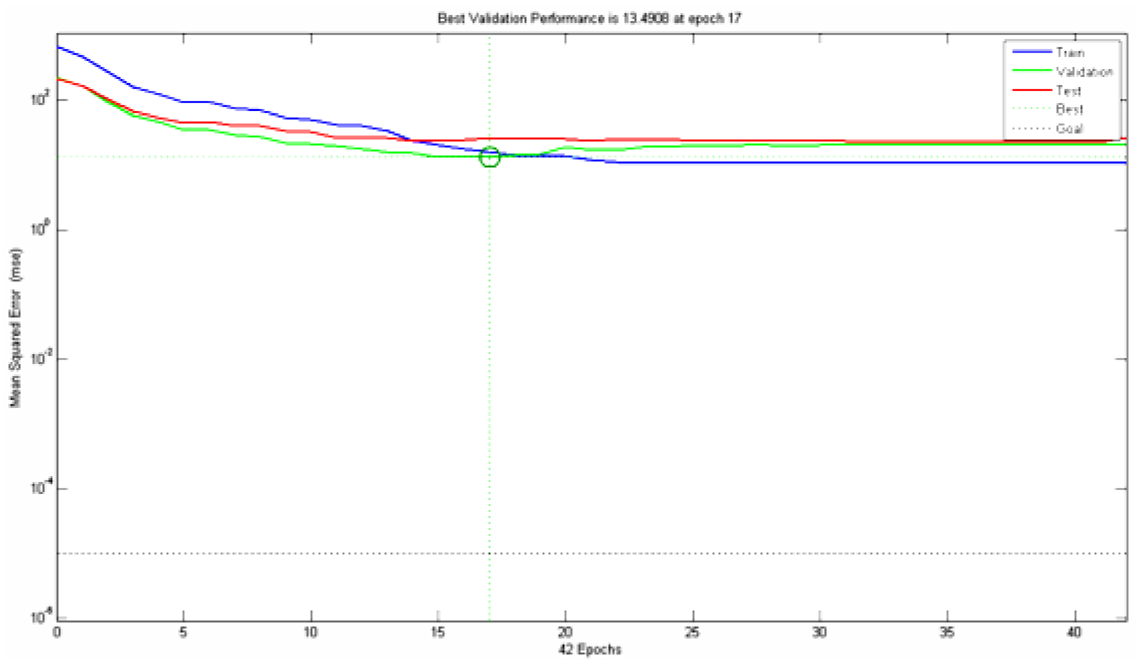
5) [8 4]

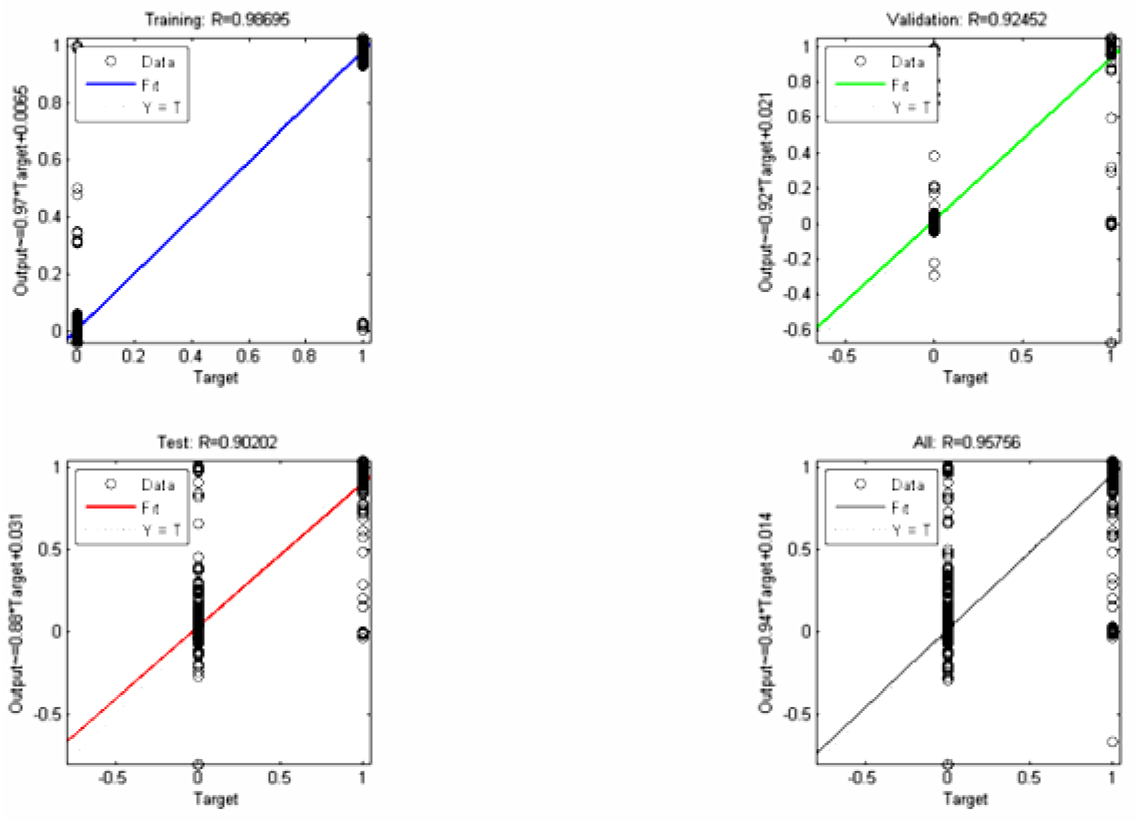
a) Levenberg–Marquardt



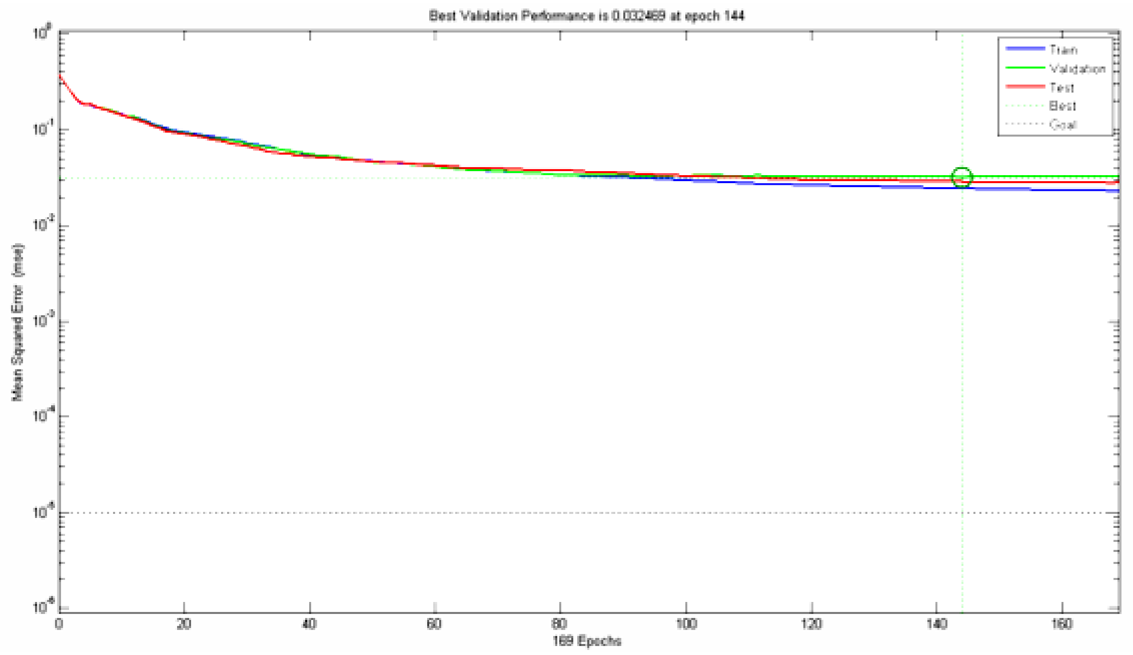


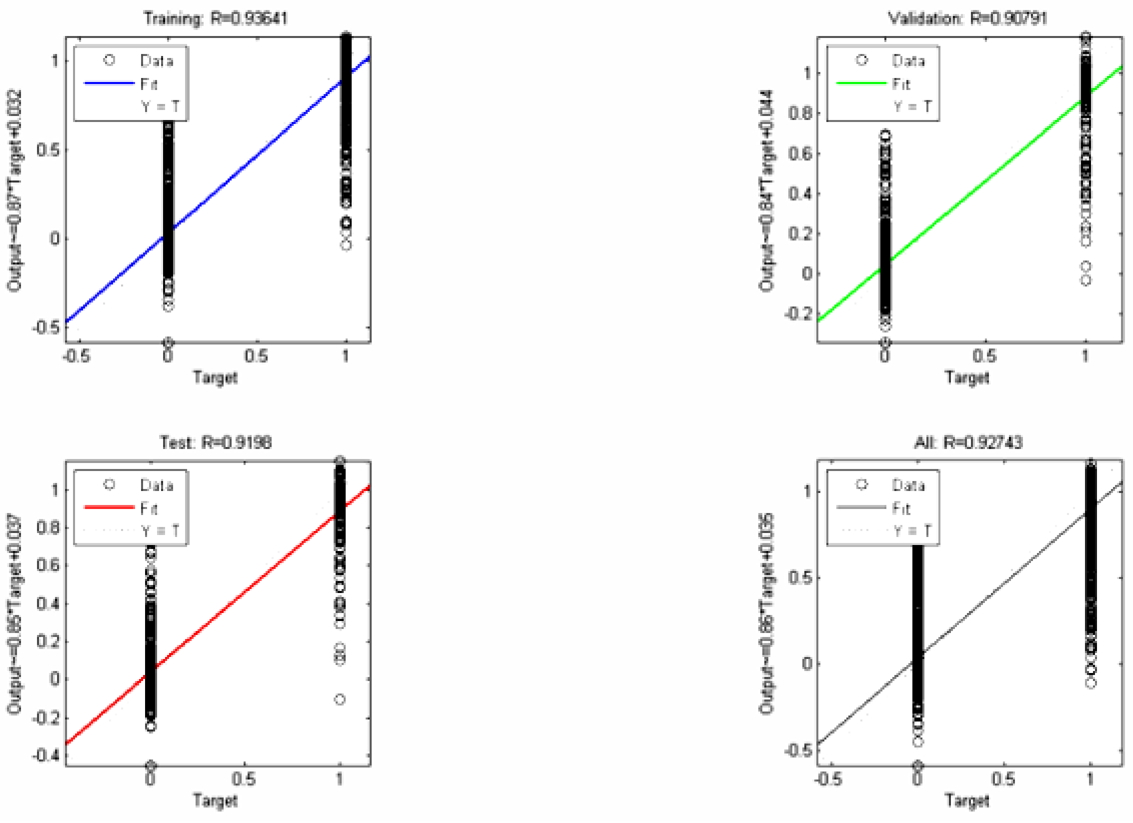
b) Bayesian Regularisation



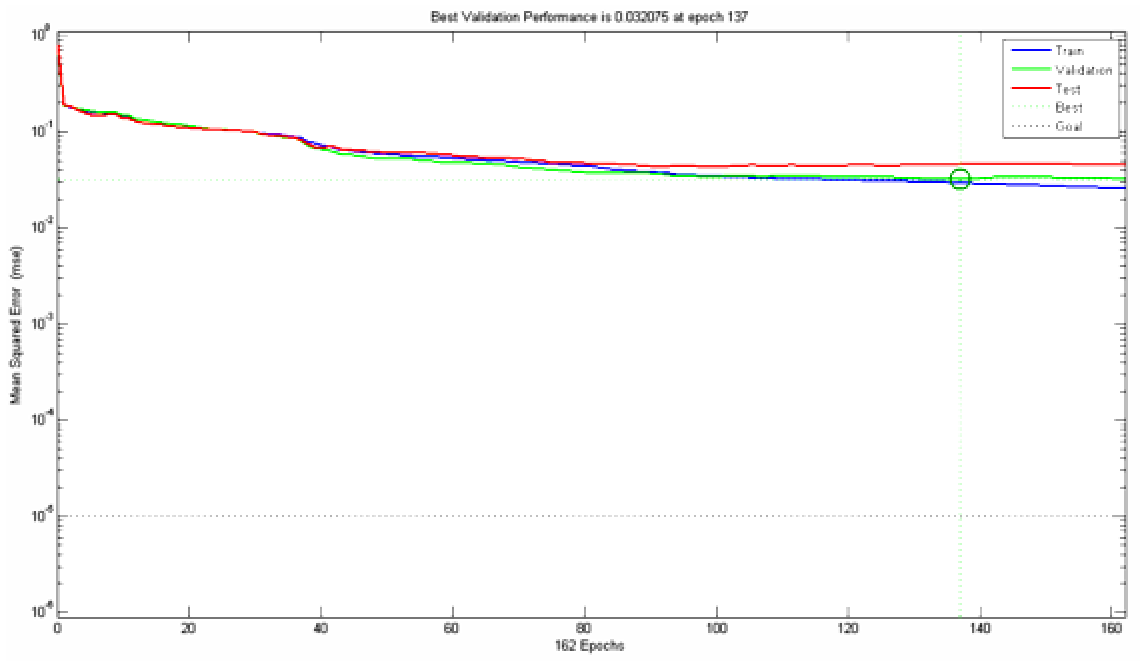


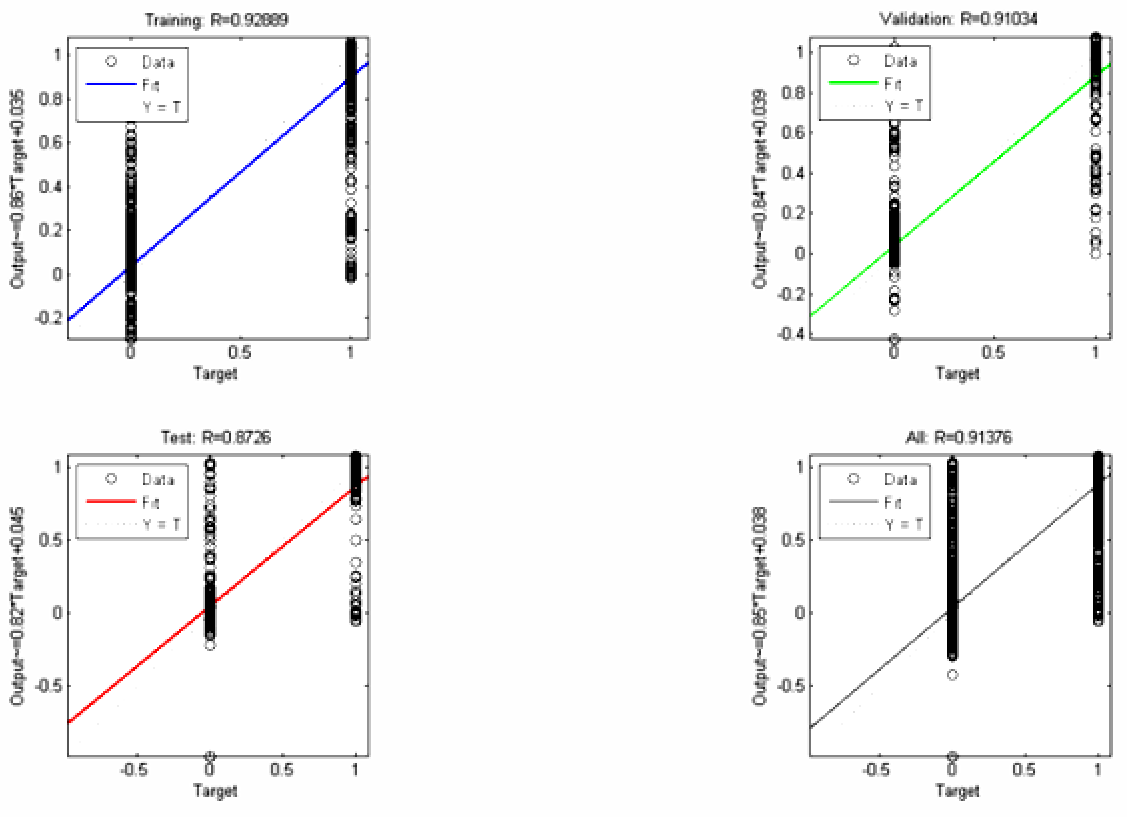
c) Resilient Backpropagation



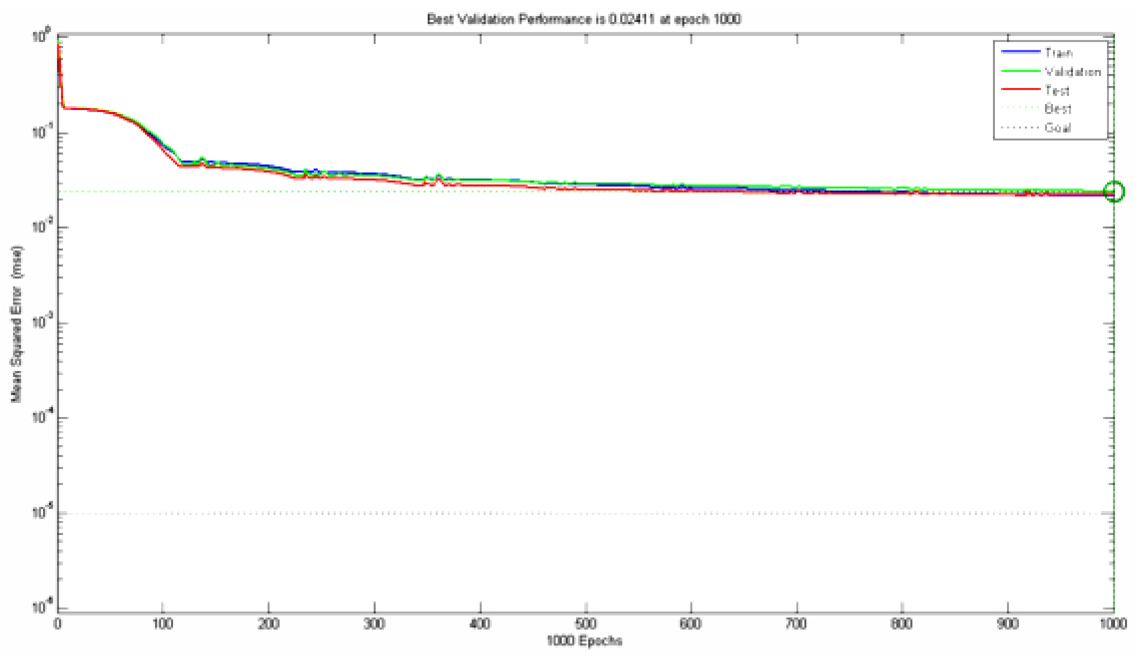


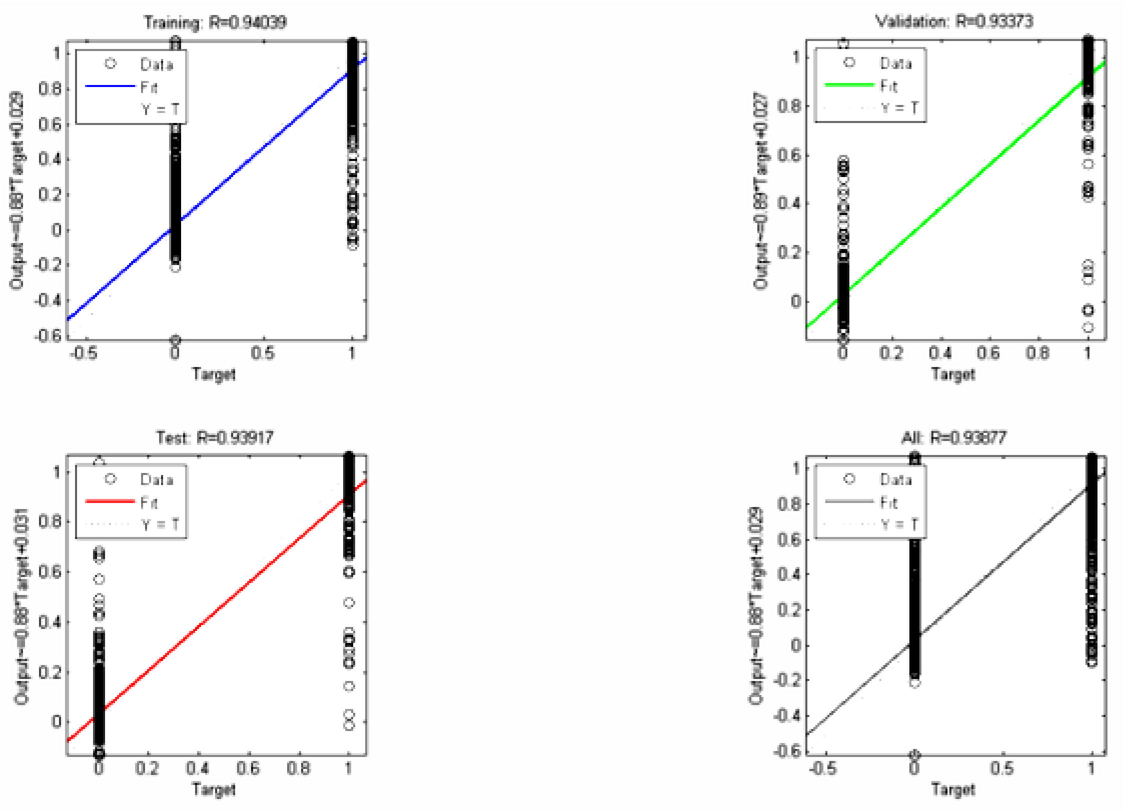
d) Scaled Conjugate Gradient



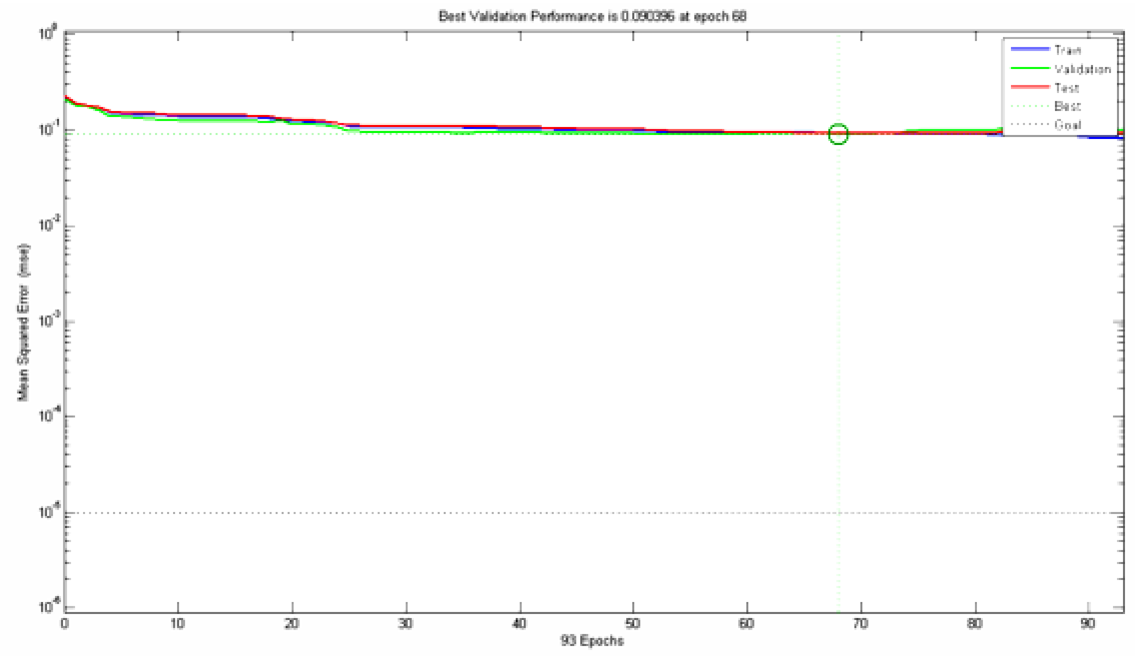


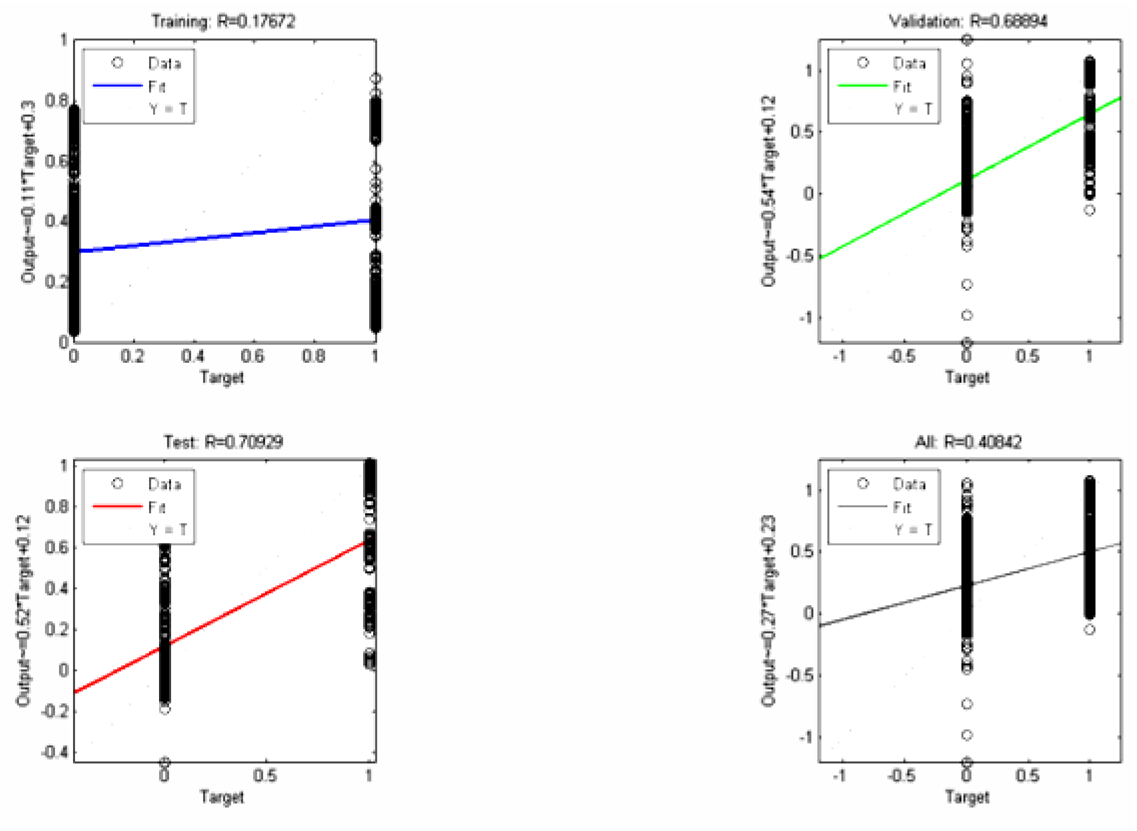
e) Variable Learning Rate Backpropagation





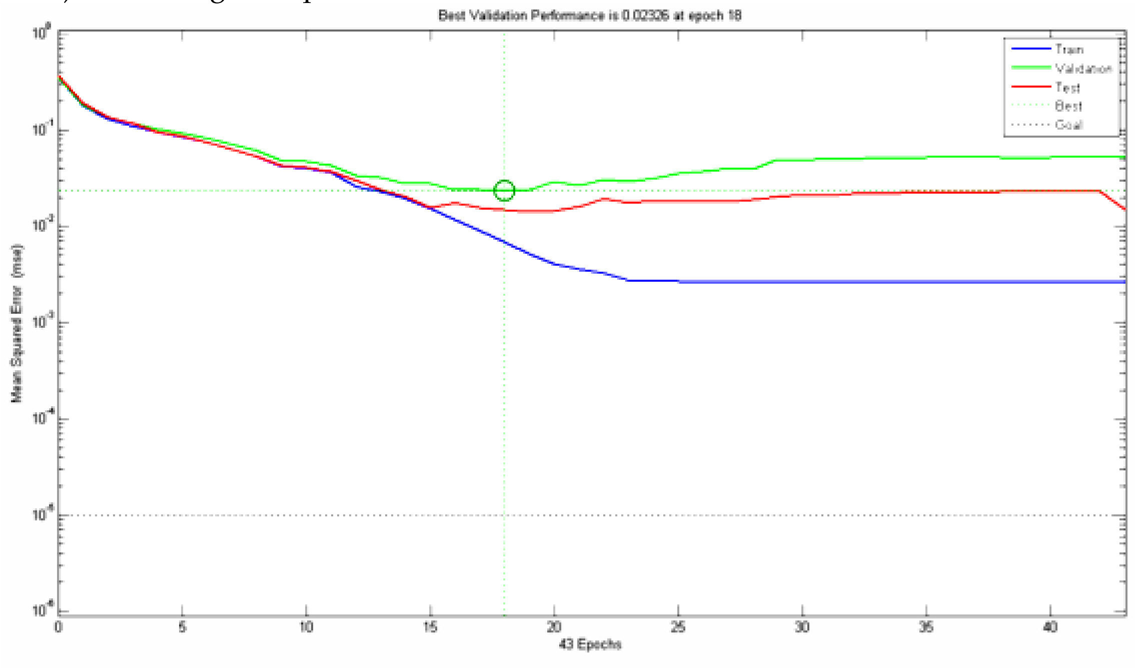
f) BFGS quasi–Newton Backpropagation

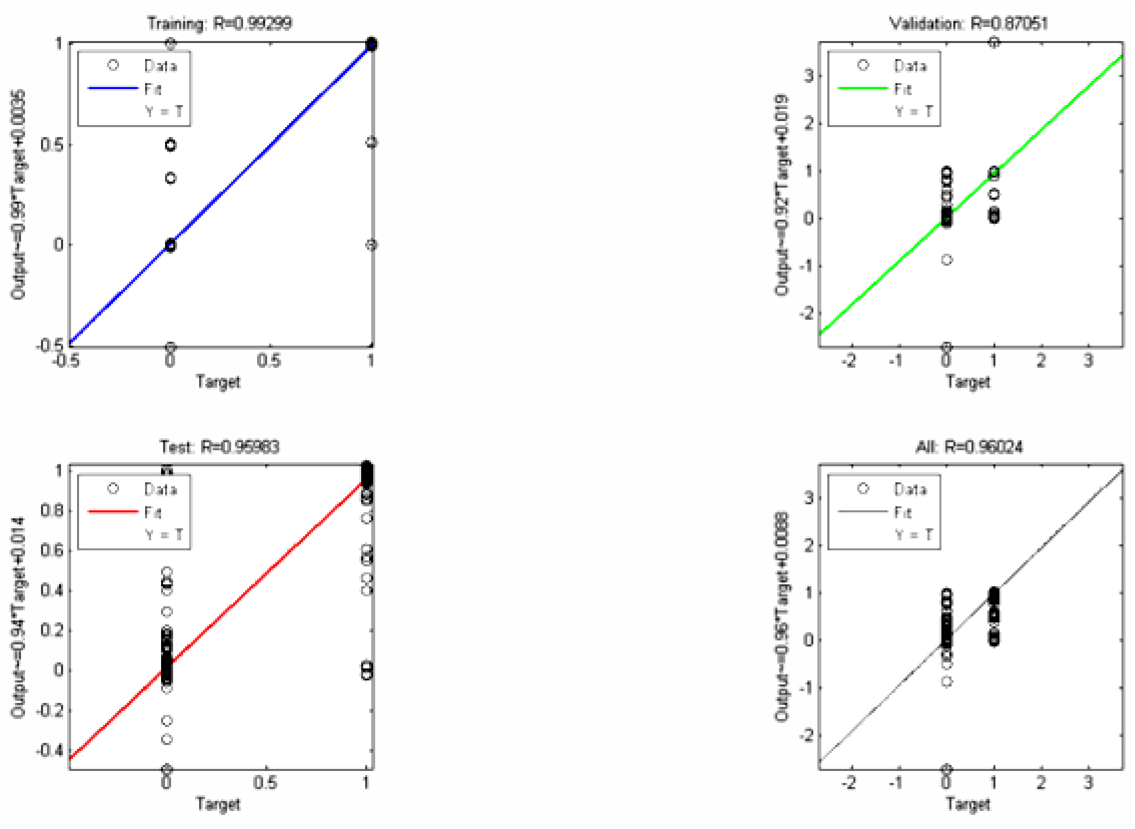




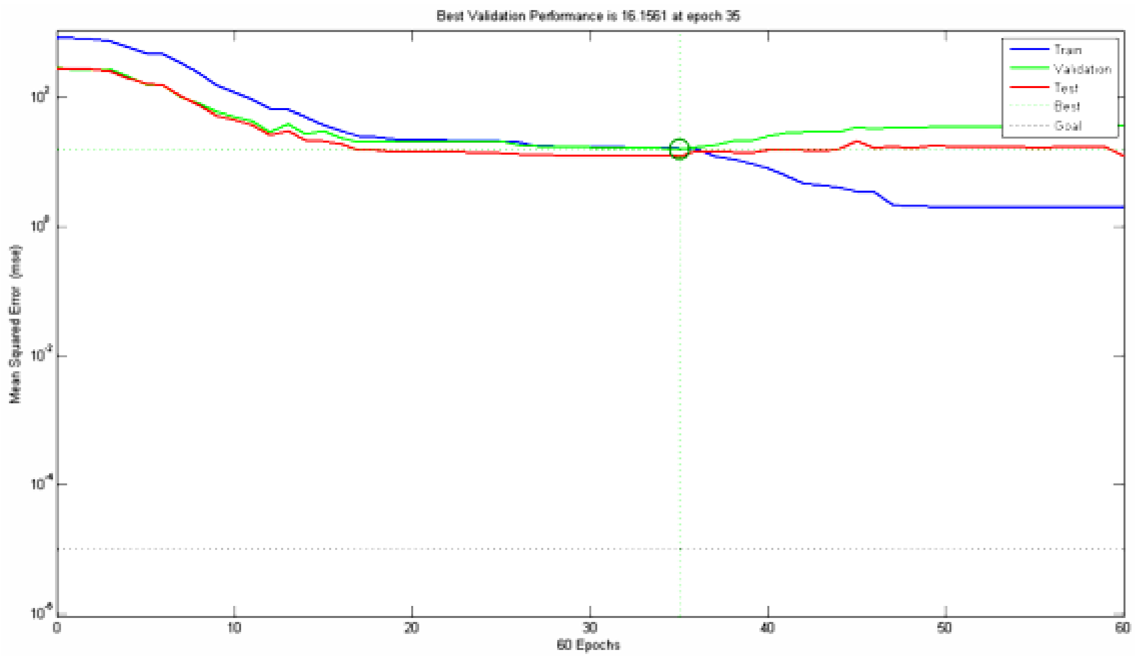
5) [10 4]

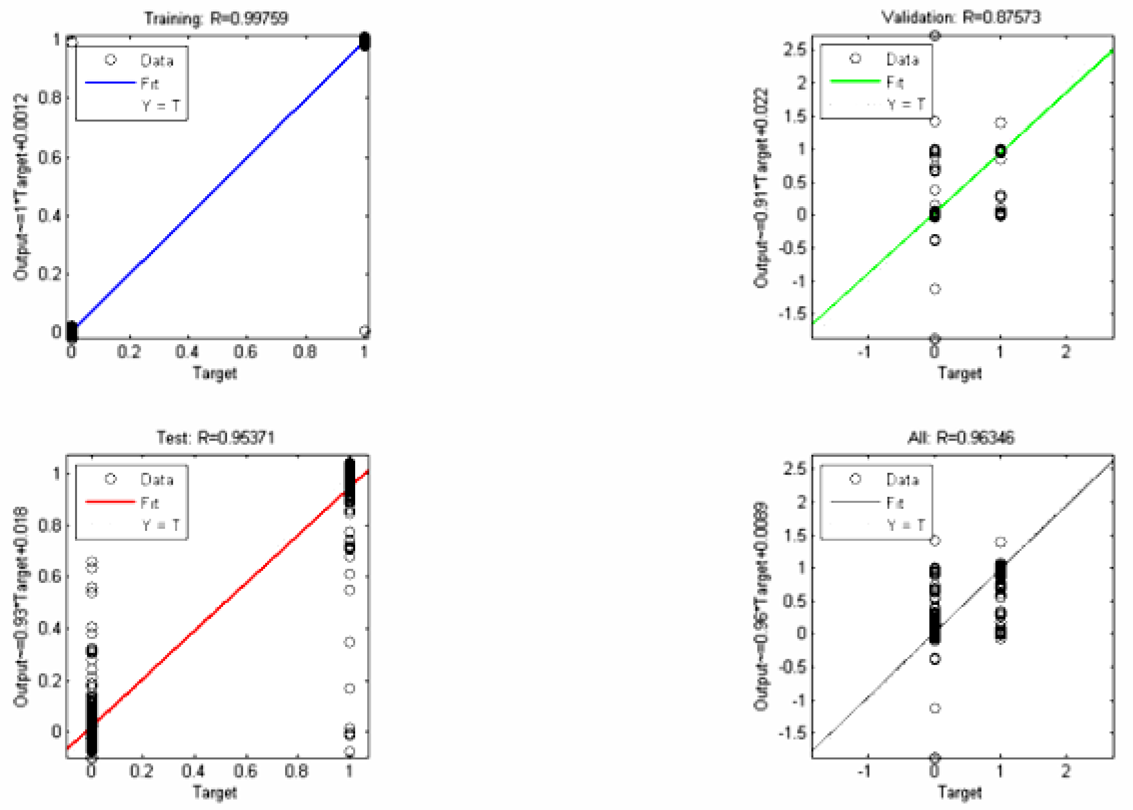
a) Levenberg–Marquardt



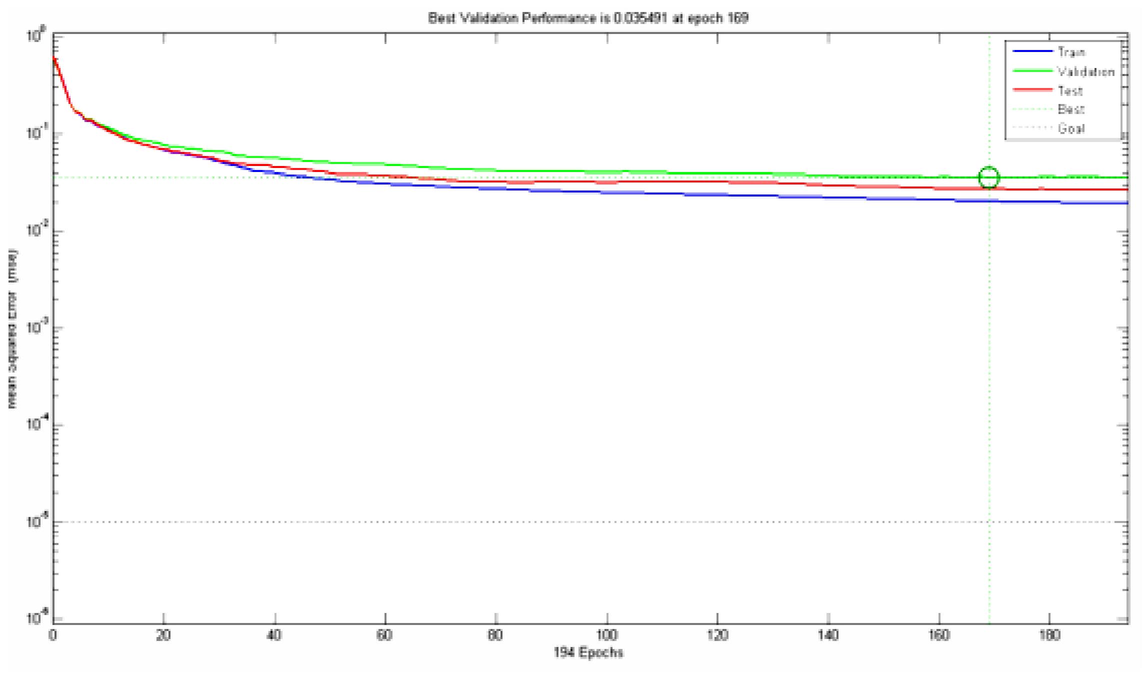


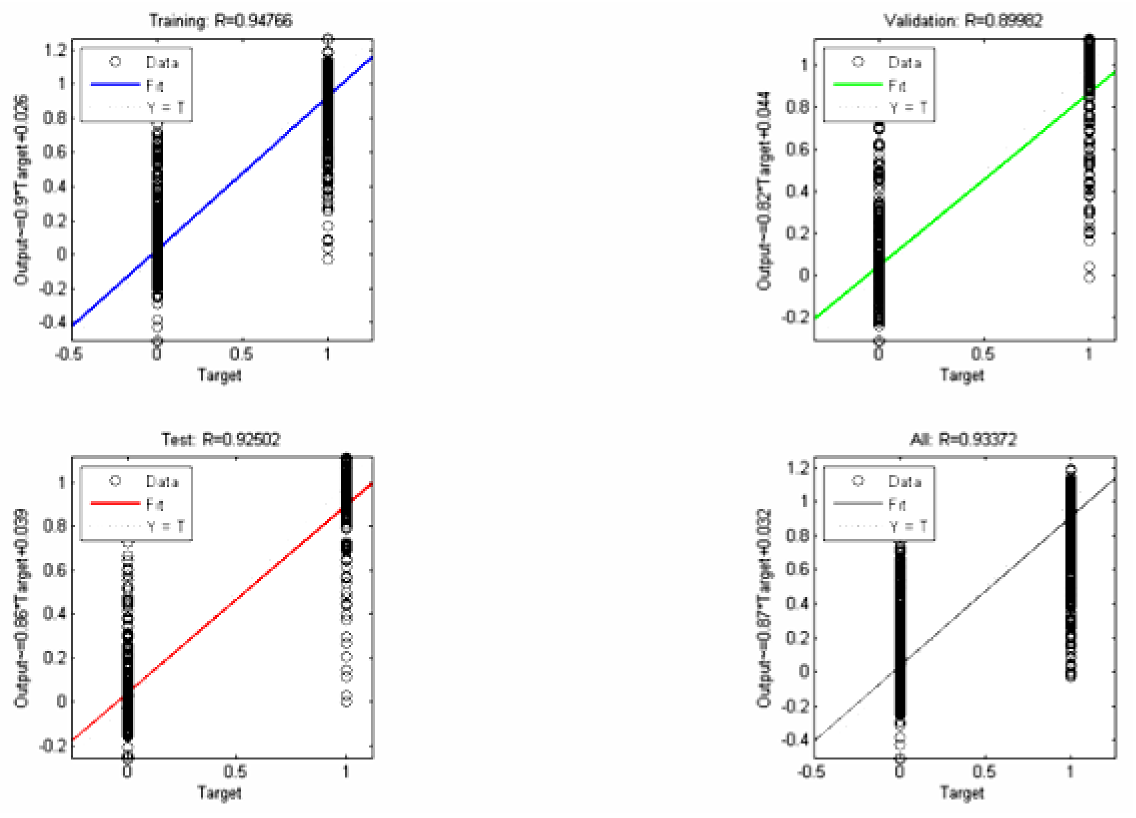
b) Bayesian Regularisation



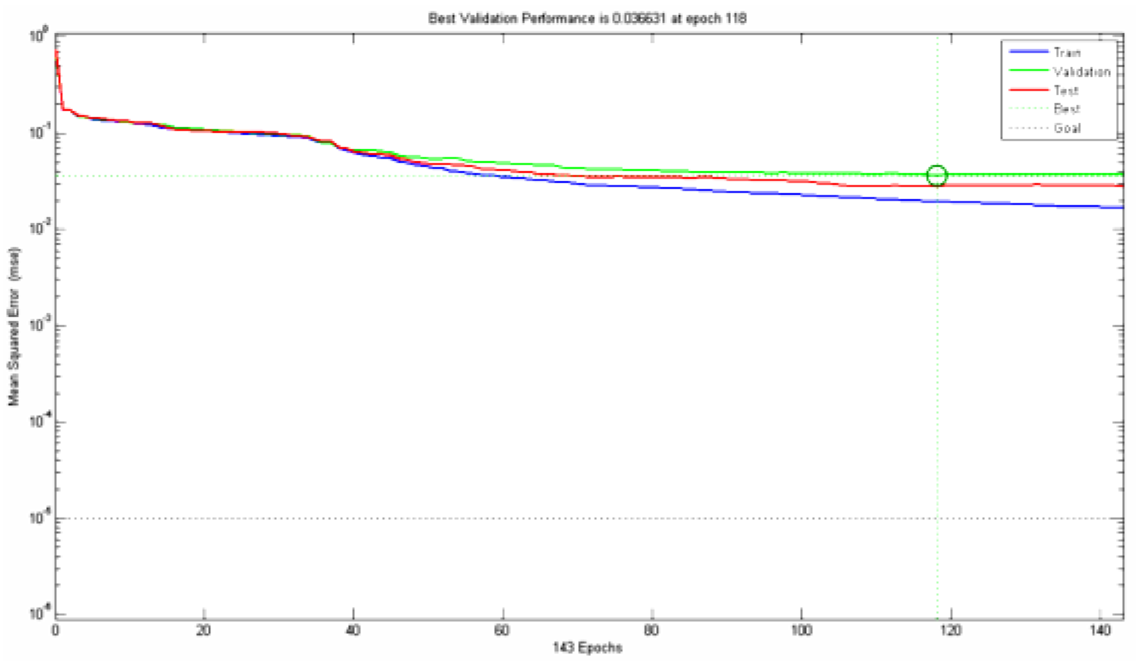


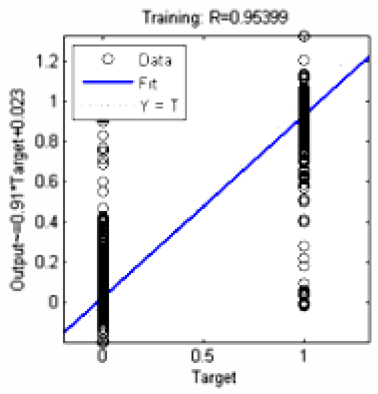
c) Resilient Backpropagation

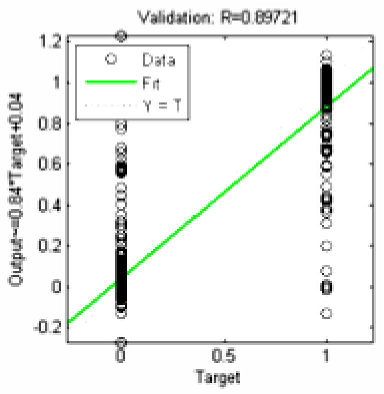


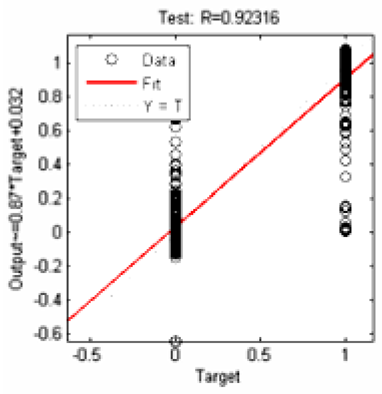


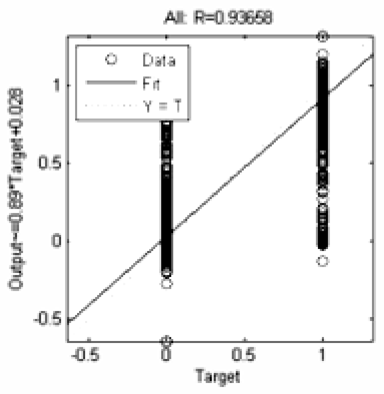
d) Scaled Conjugate Gradient



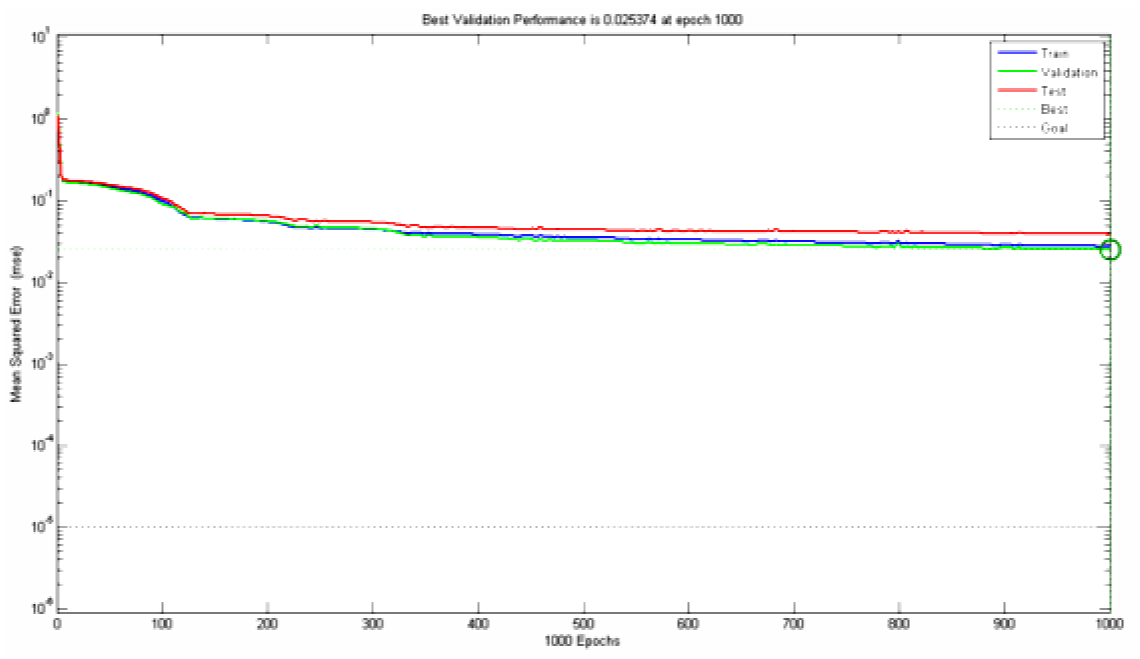


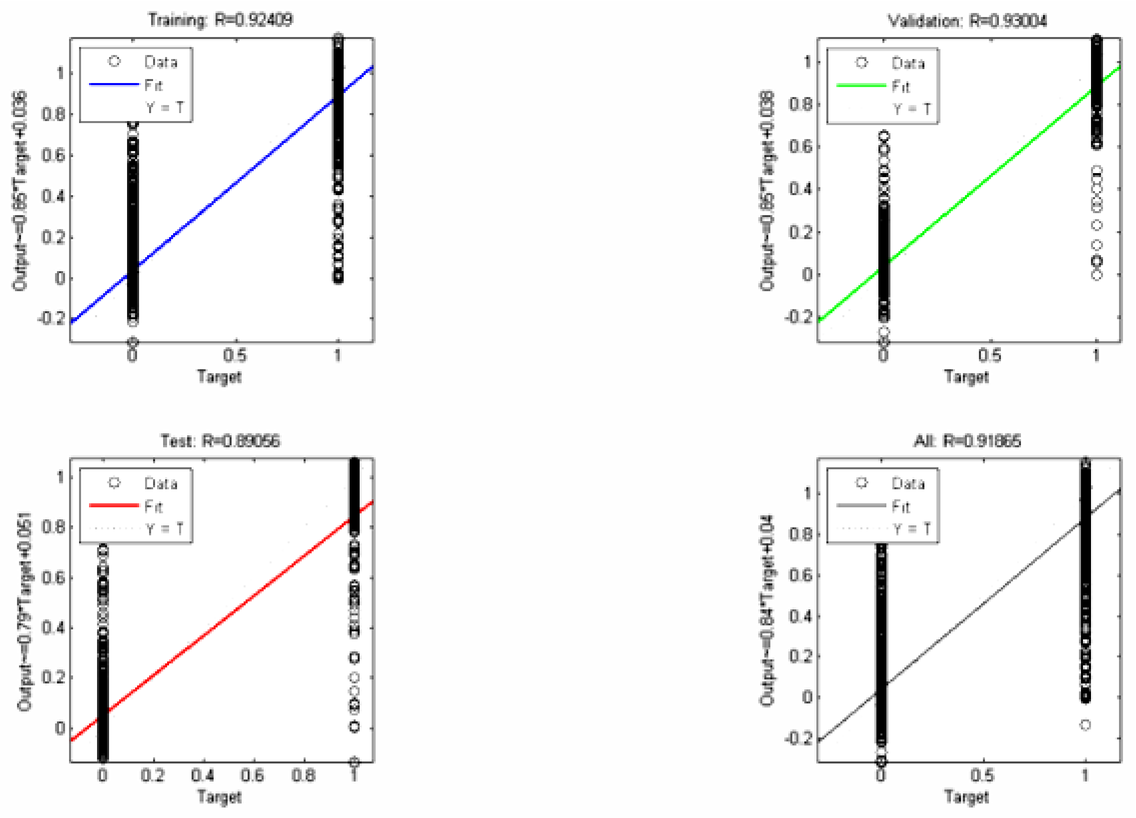




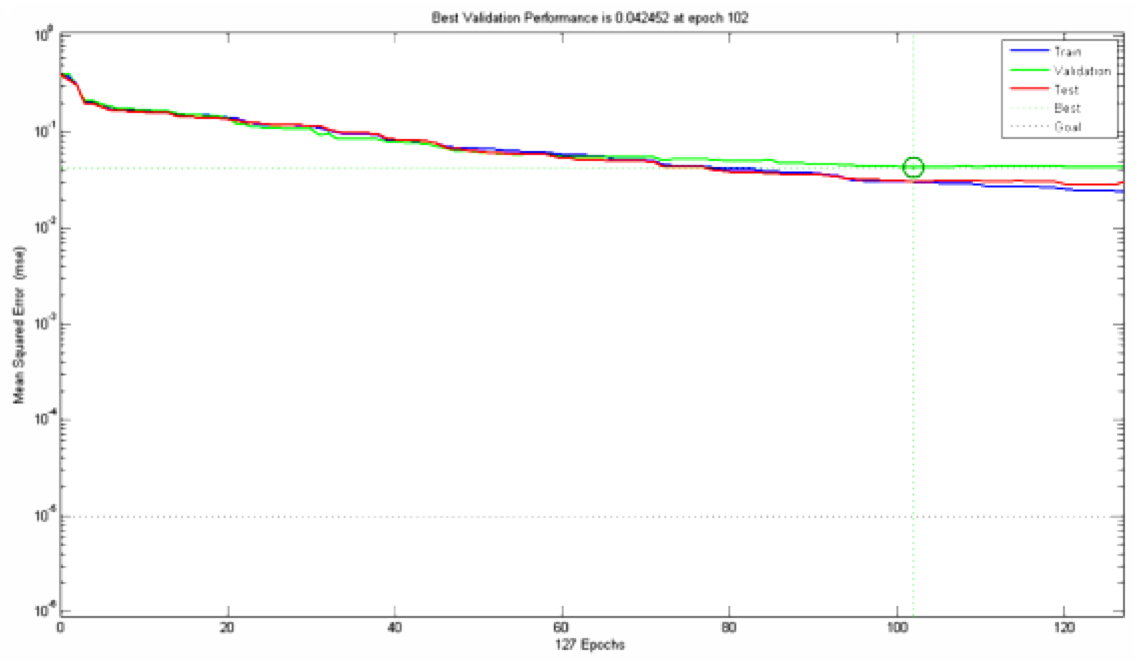


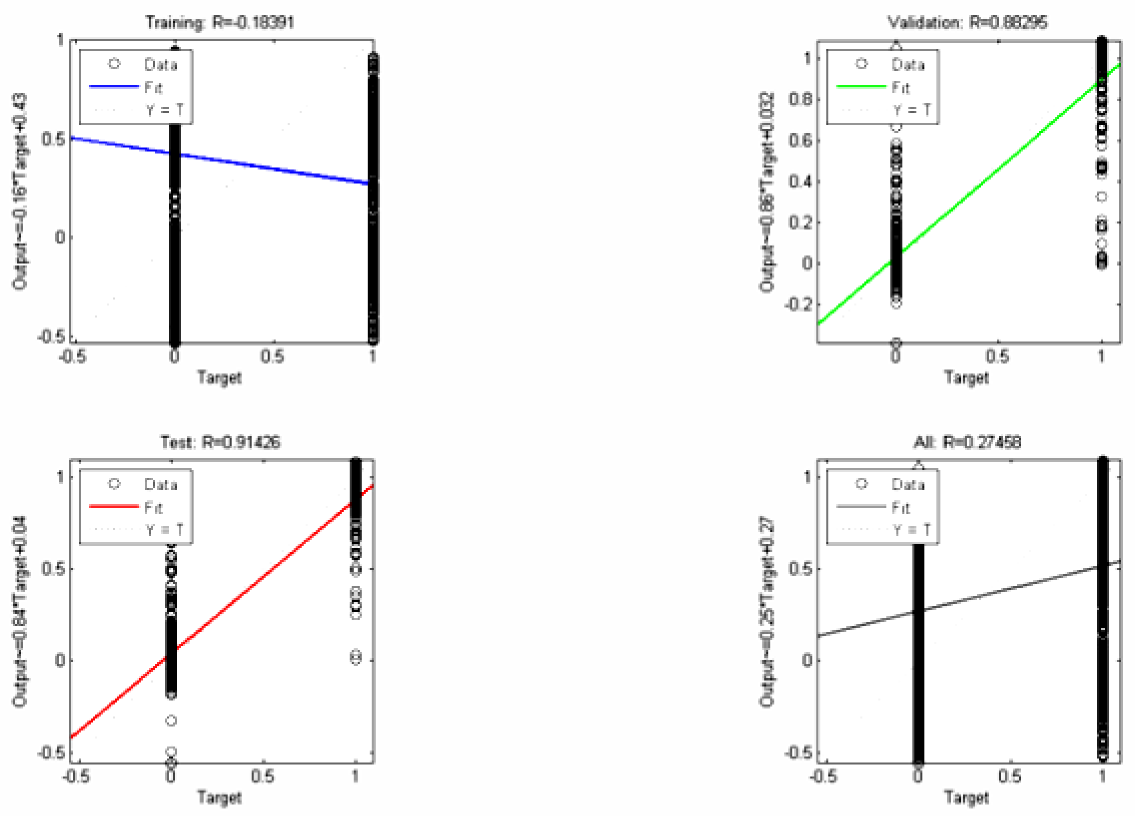
e) Variable Learning Rate Backpropagation





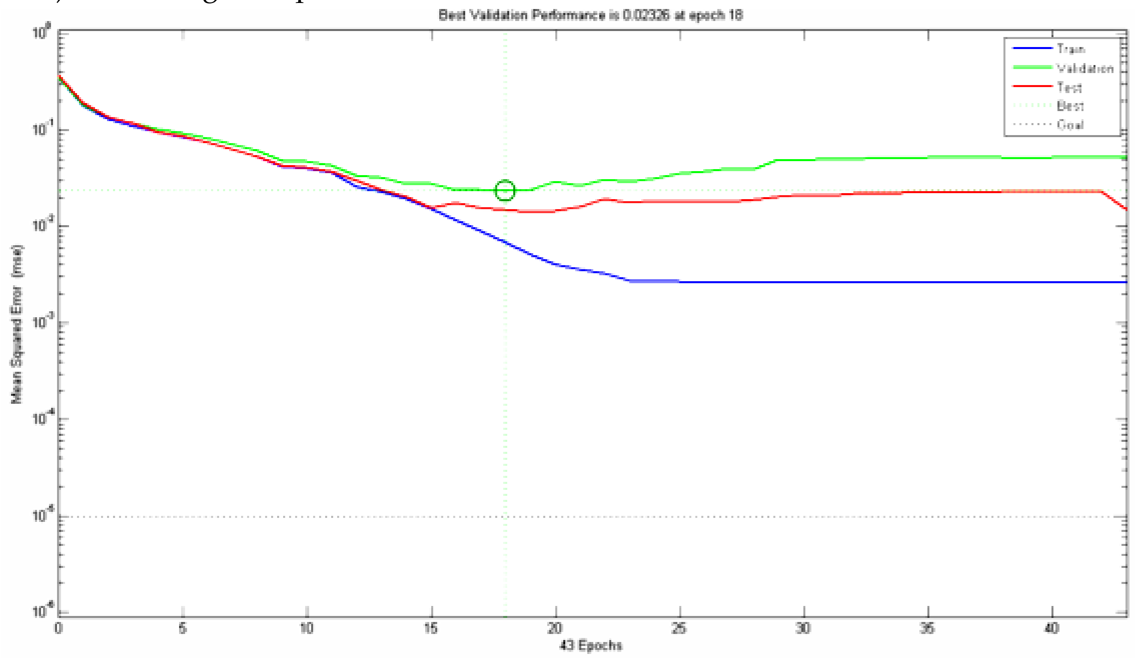
f) BFGS quasi–Newton Backpropagation

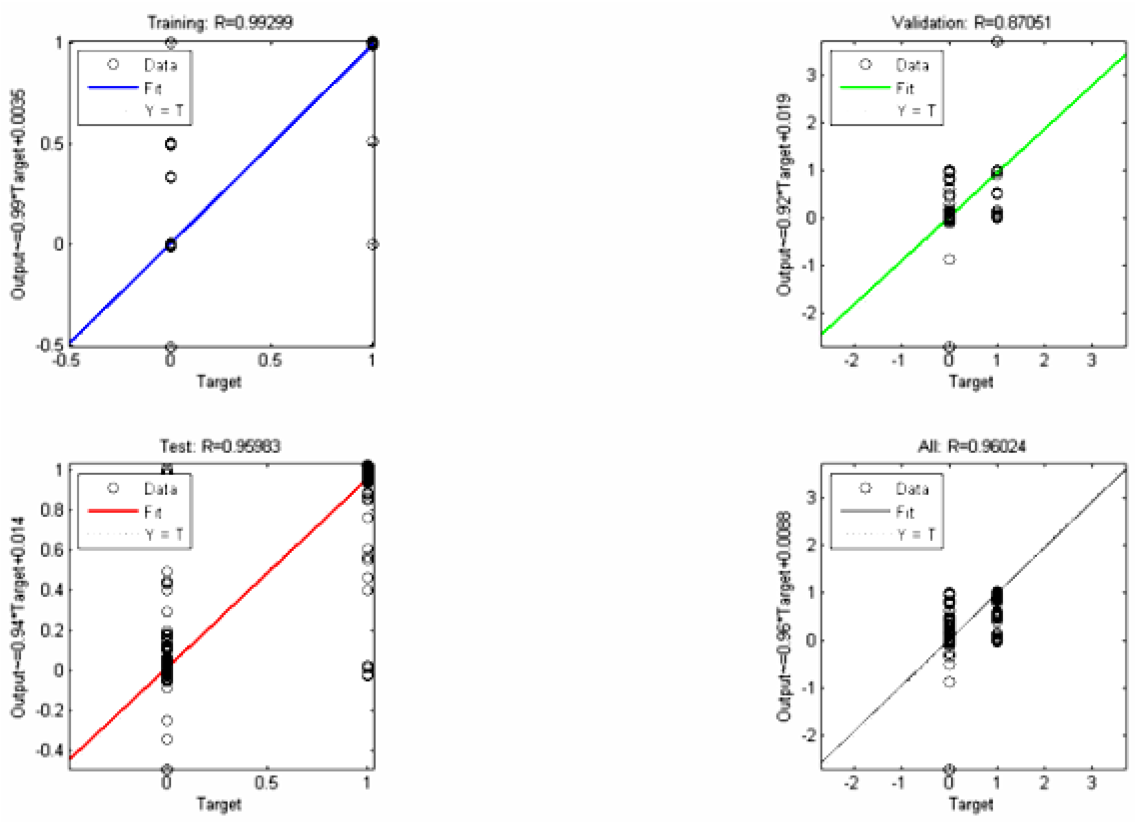




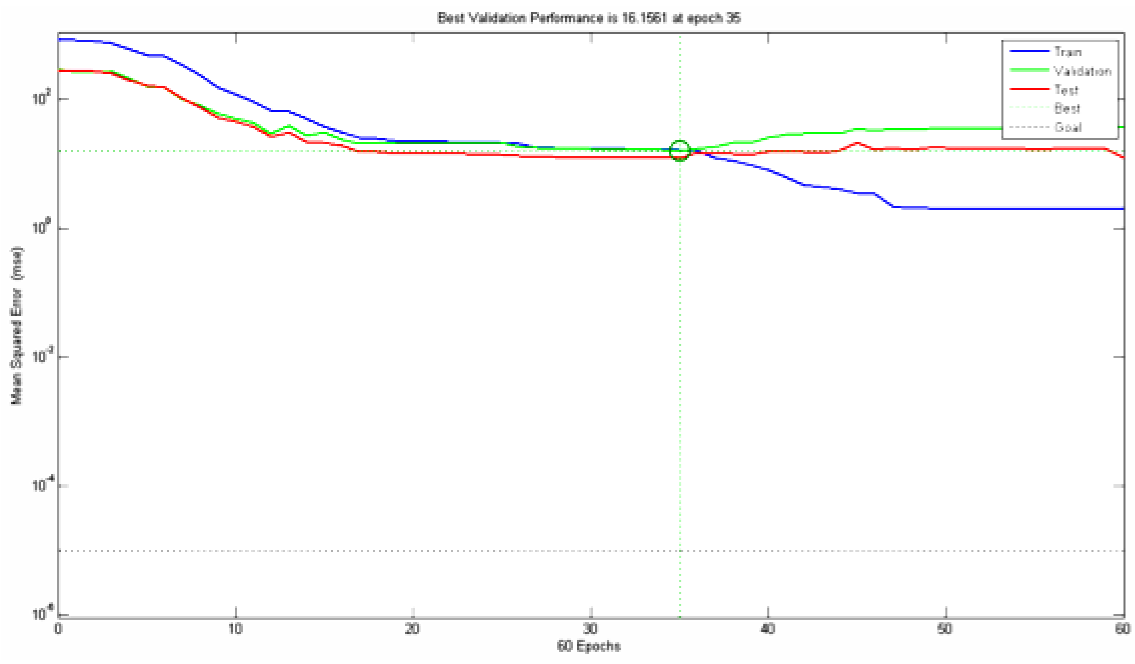
6) [10 4]

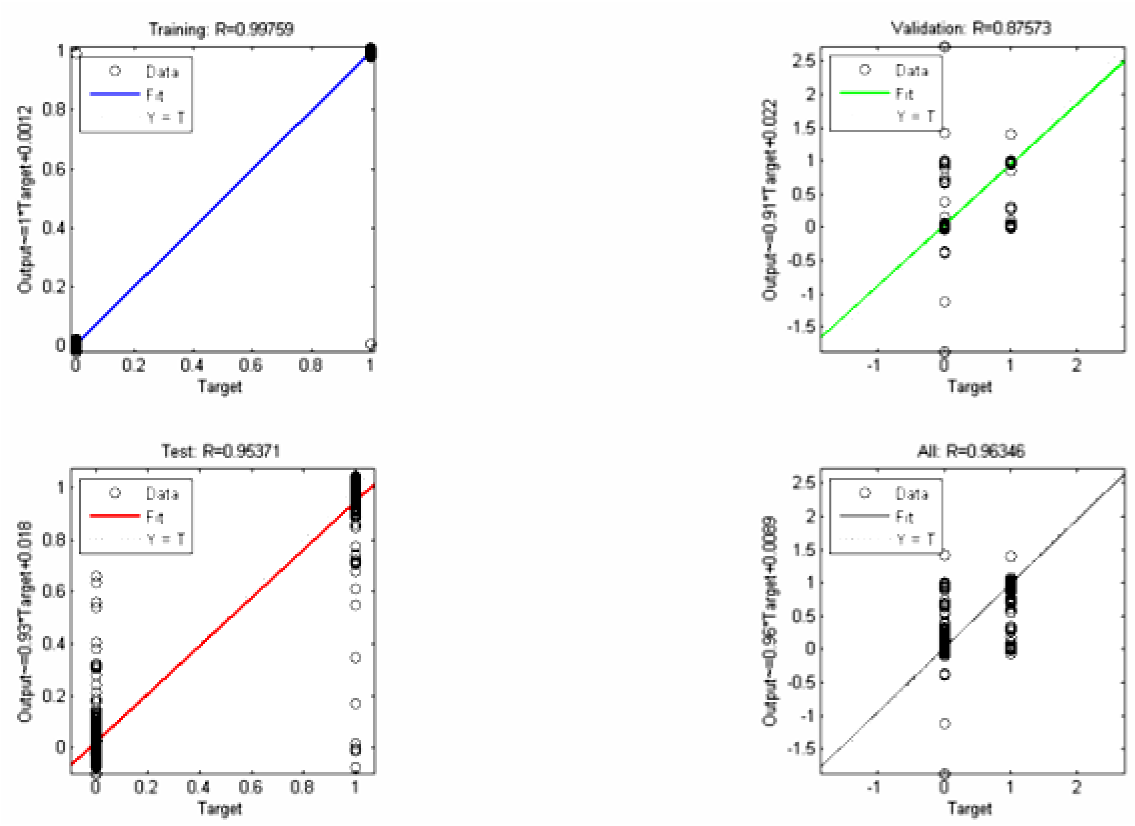
a) Levenberg–Marquardt



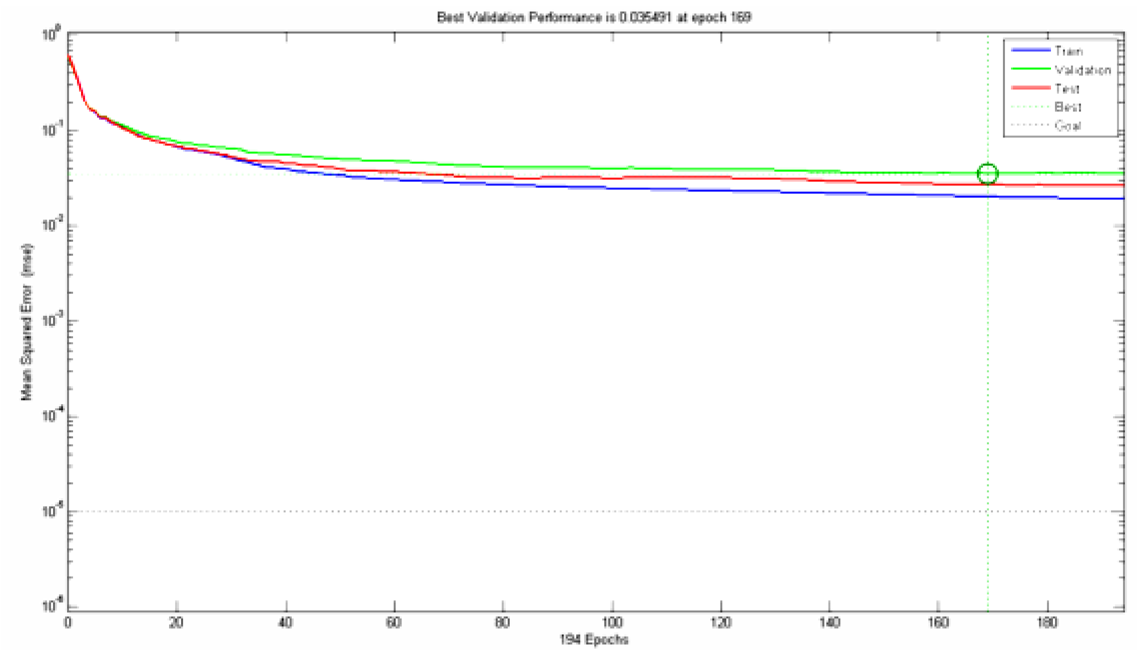


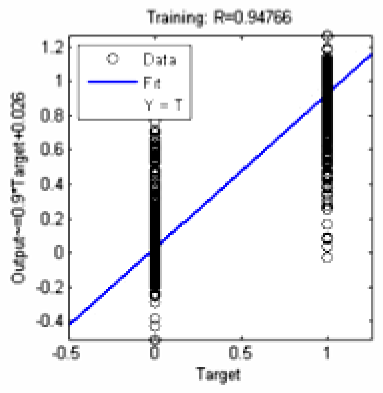
b) Bayesian Regularisation

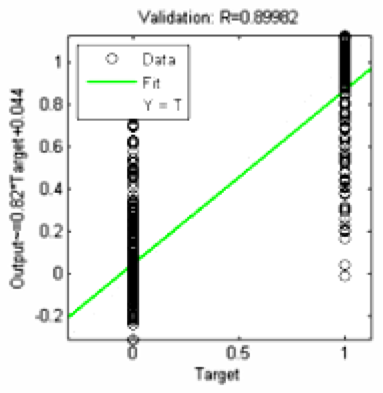


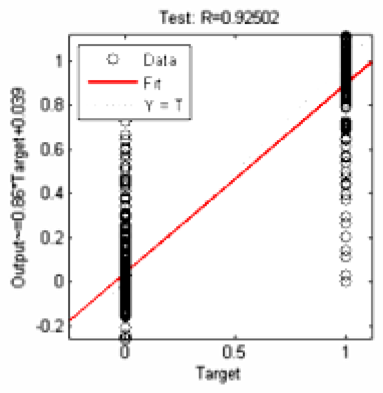


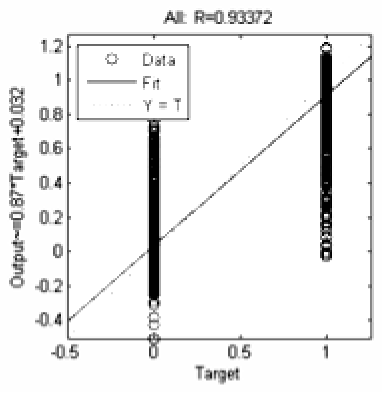
c) Resilient Backpropagation



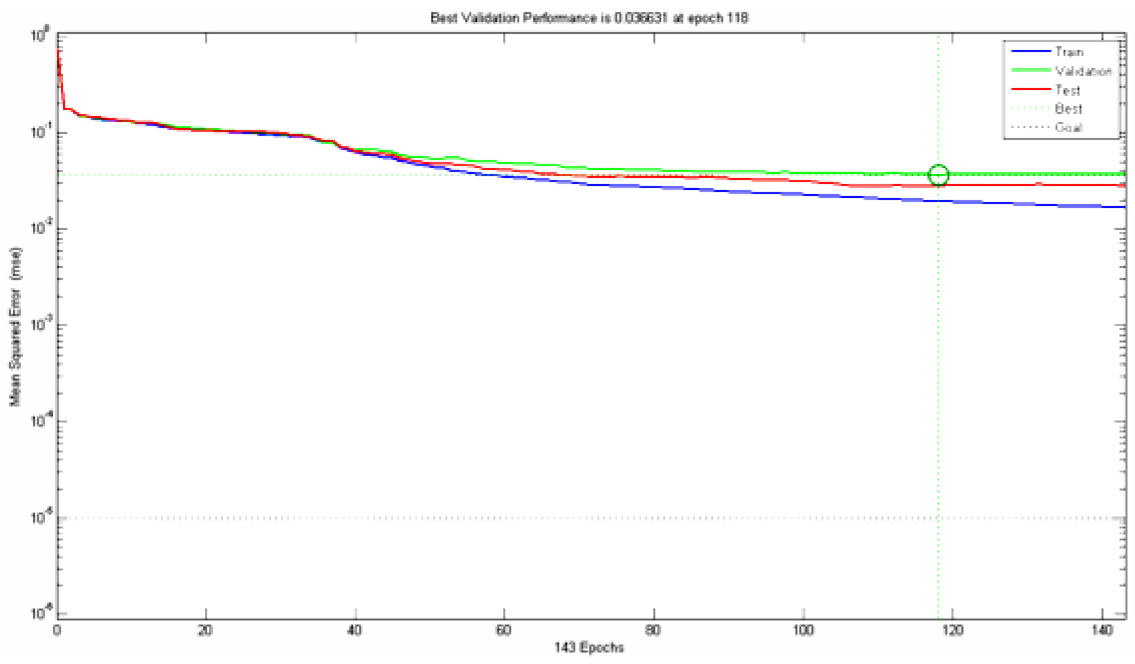


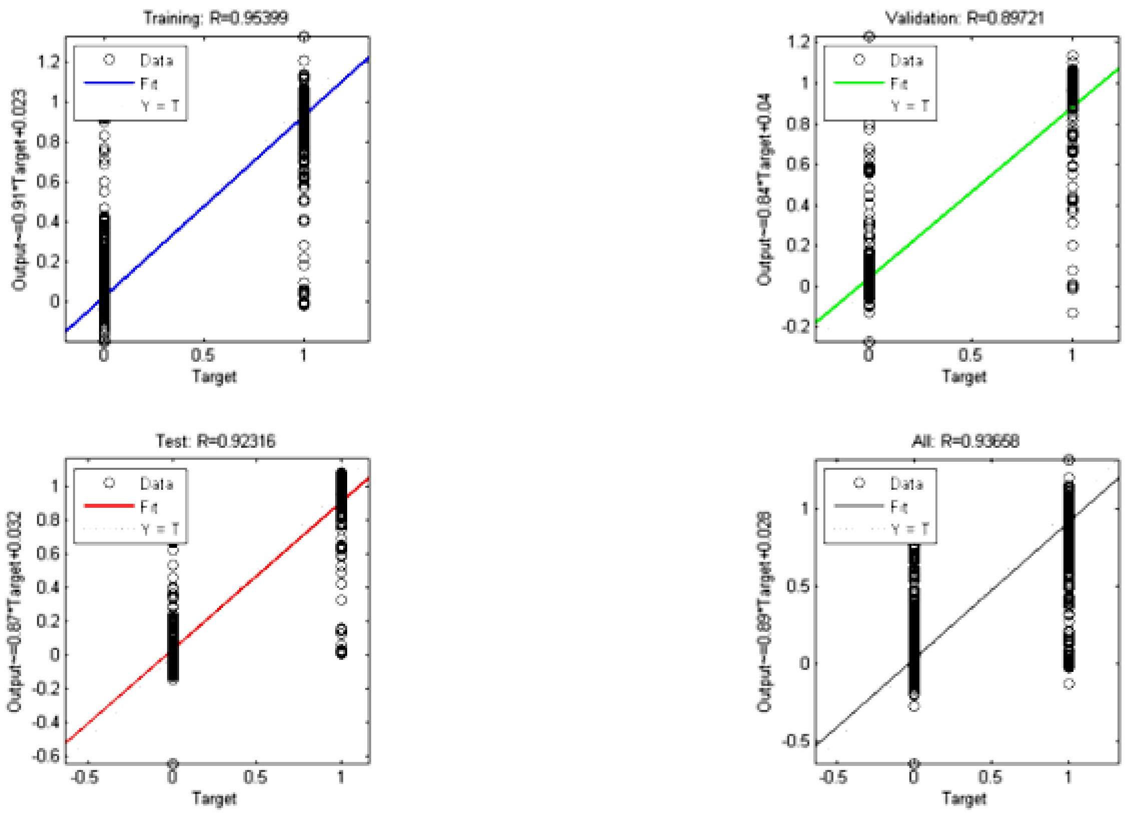




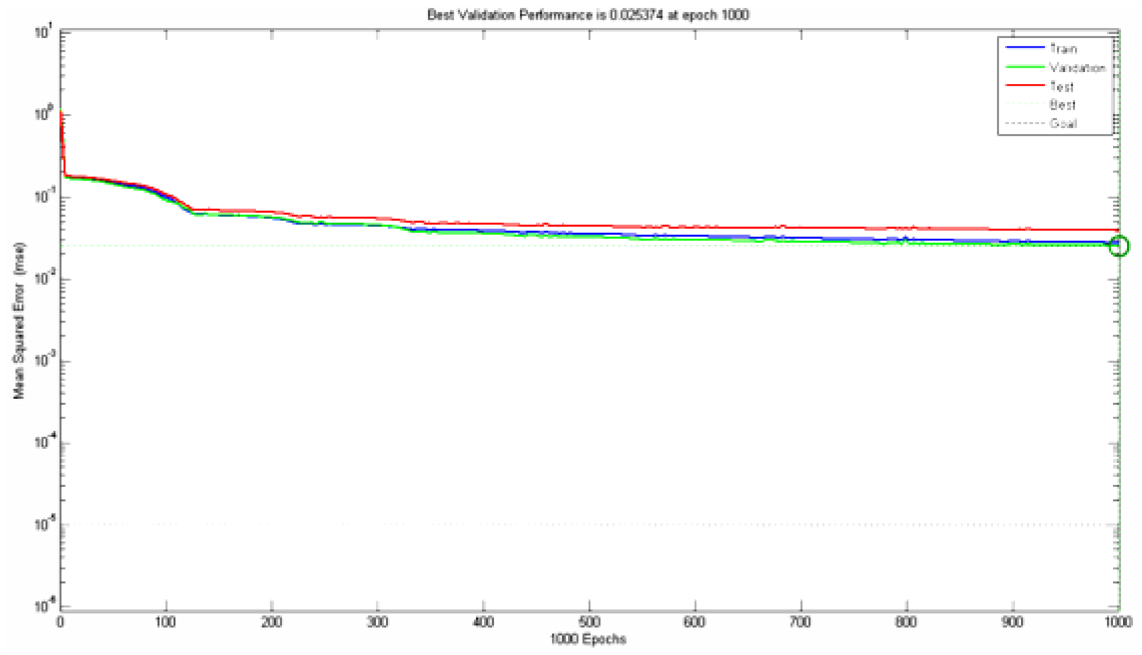


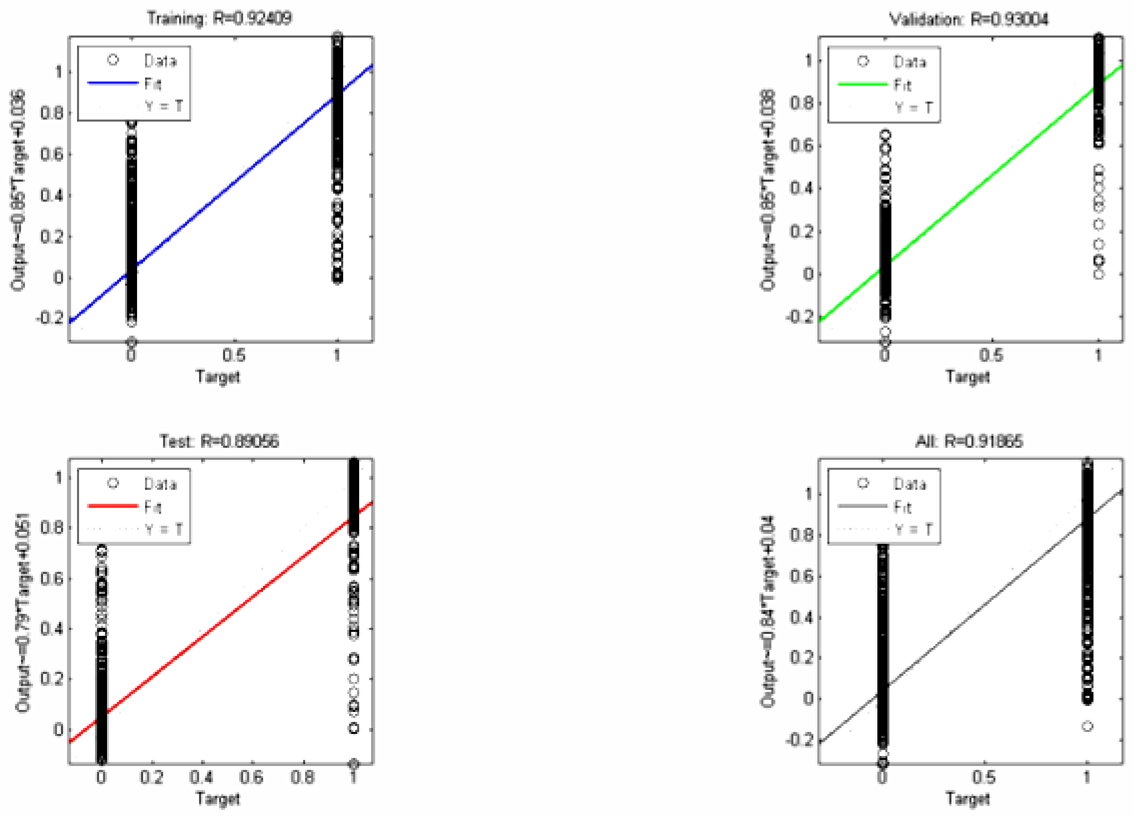
d) Scaled Conjugate Gradient



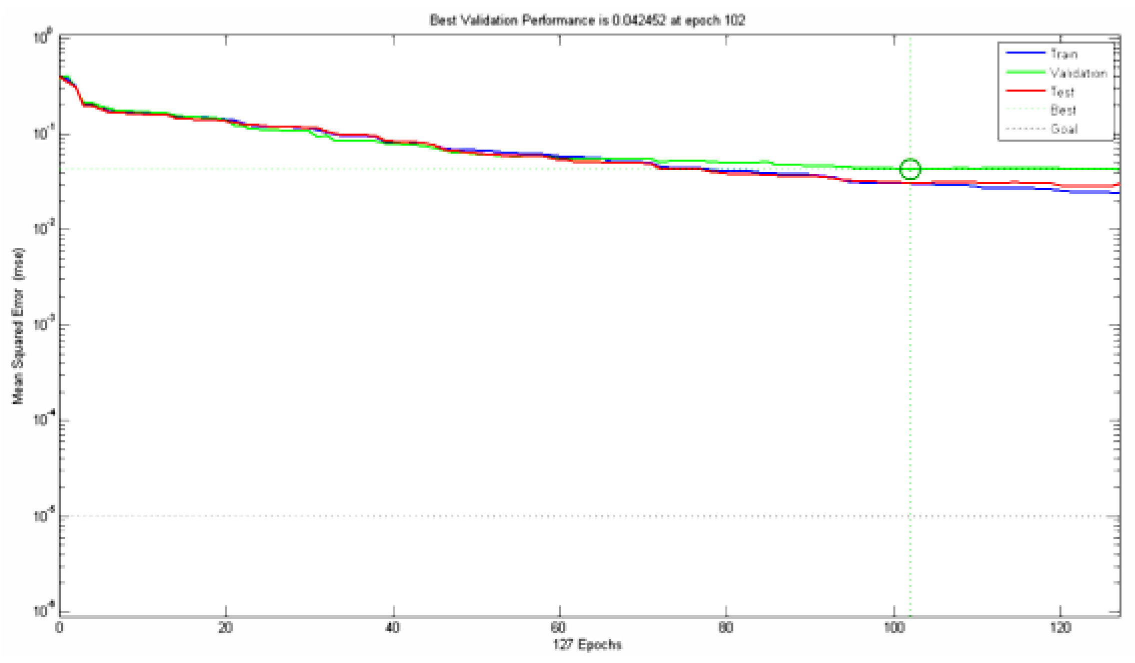


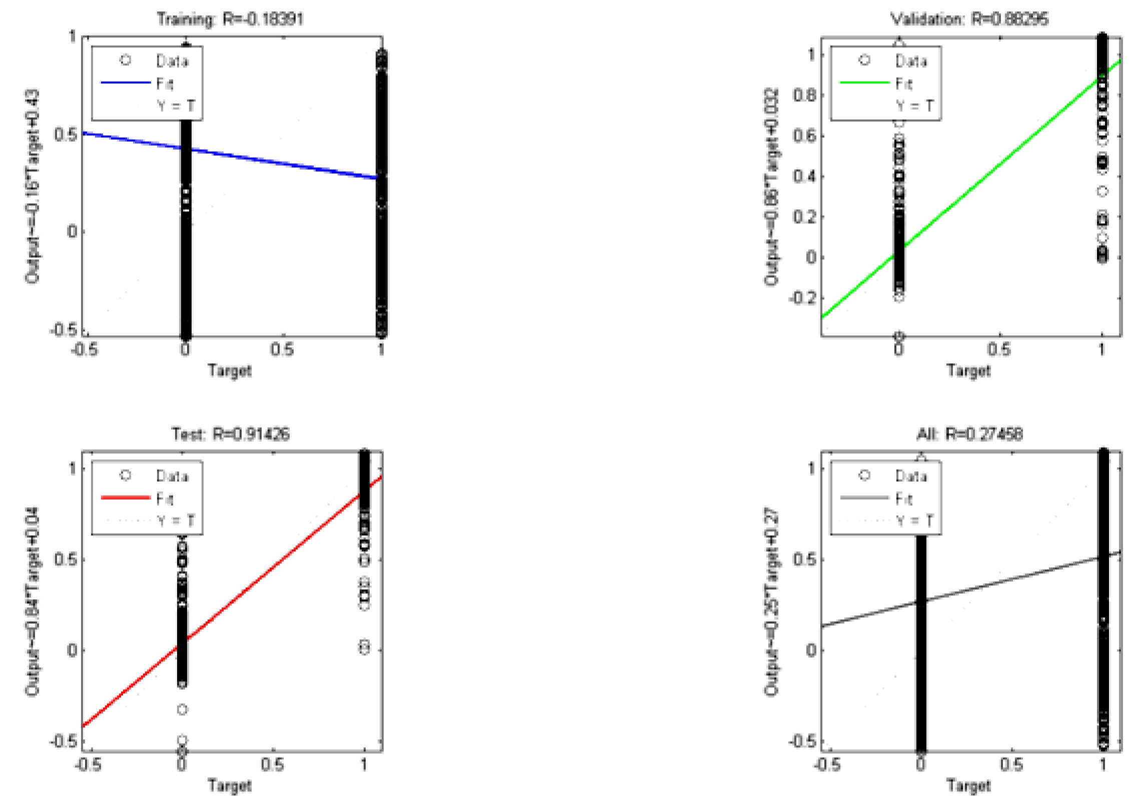
e) Variable Learning Rate Backpropagation





f) BFGS quasi–Newton Backpropagation

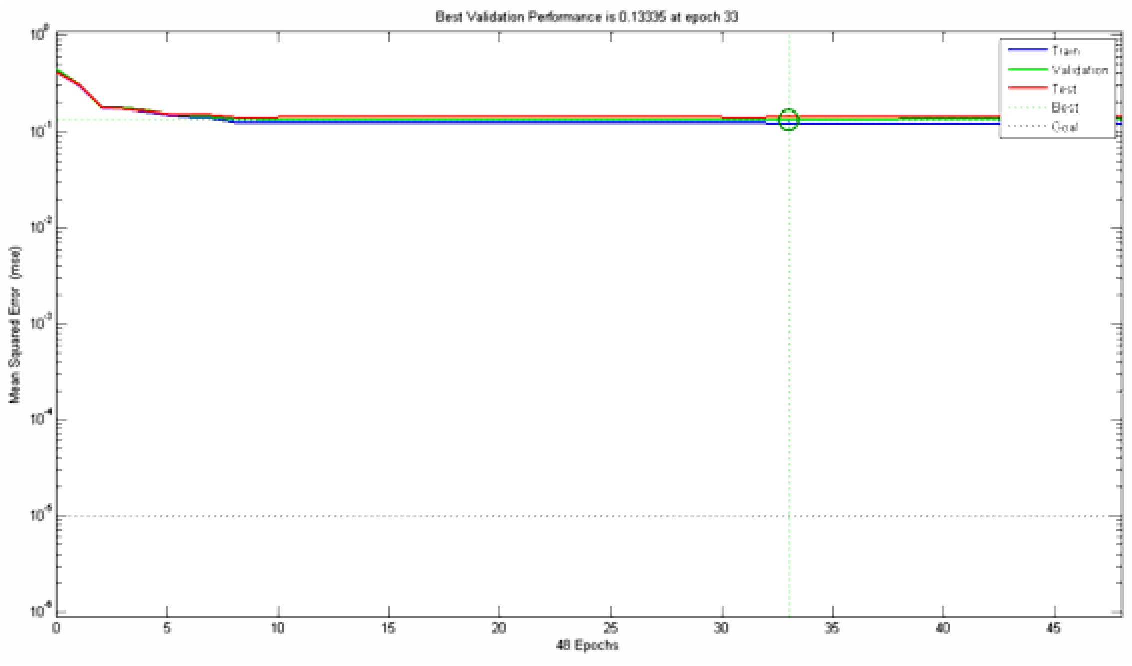


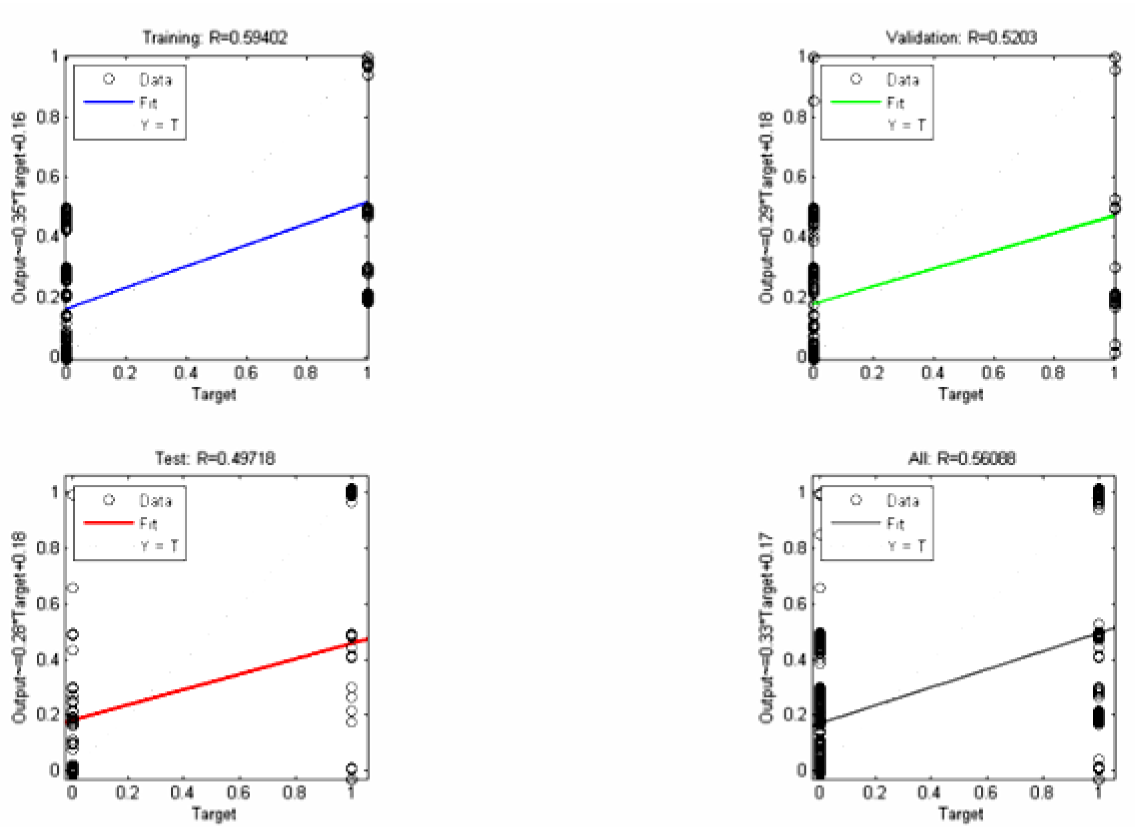


3. Three hidden layer

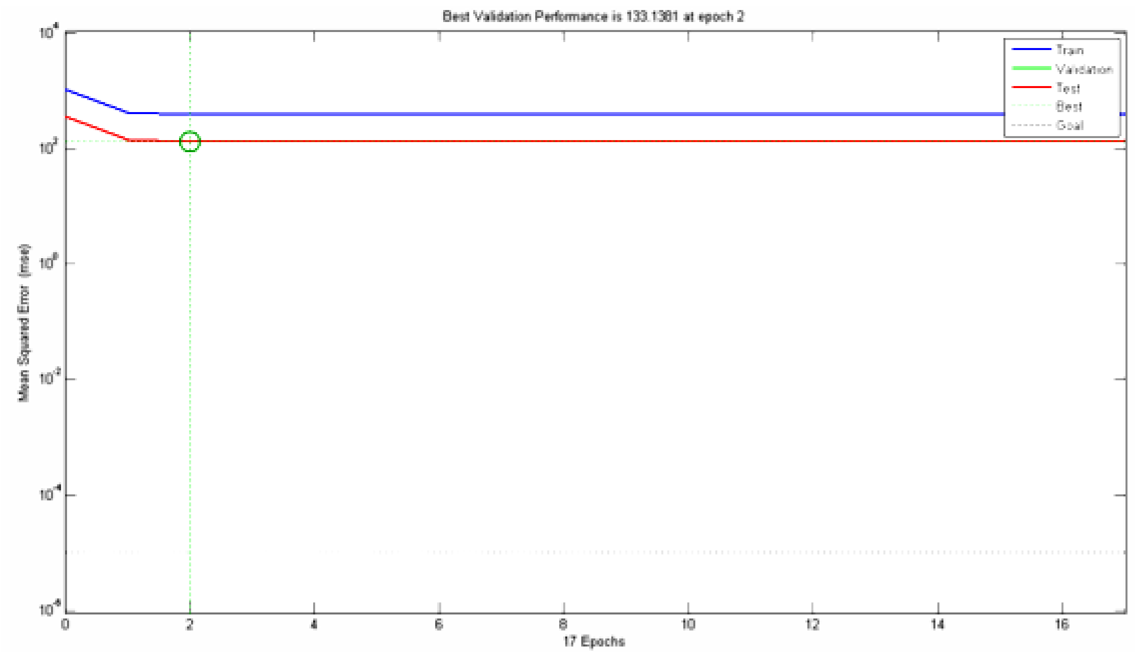
1) [2 2 2]

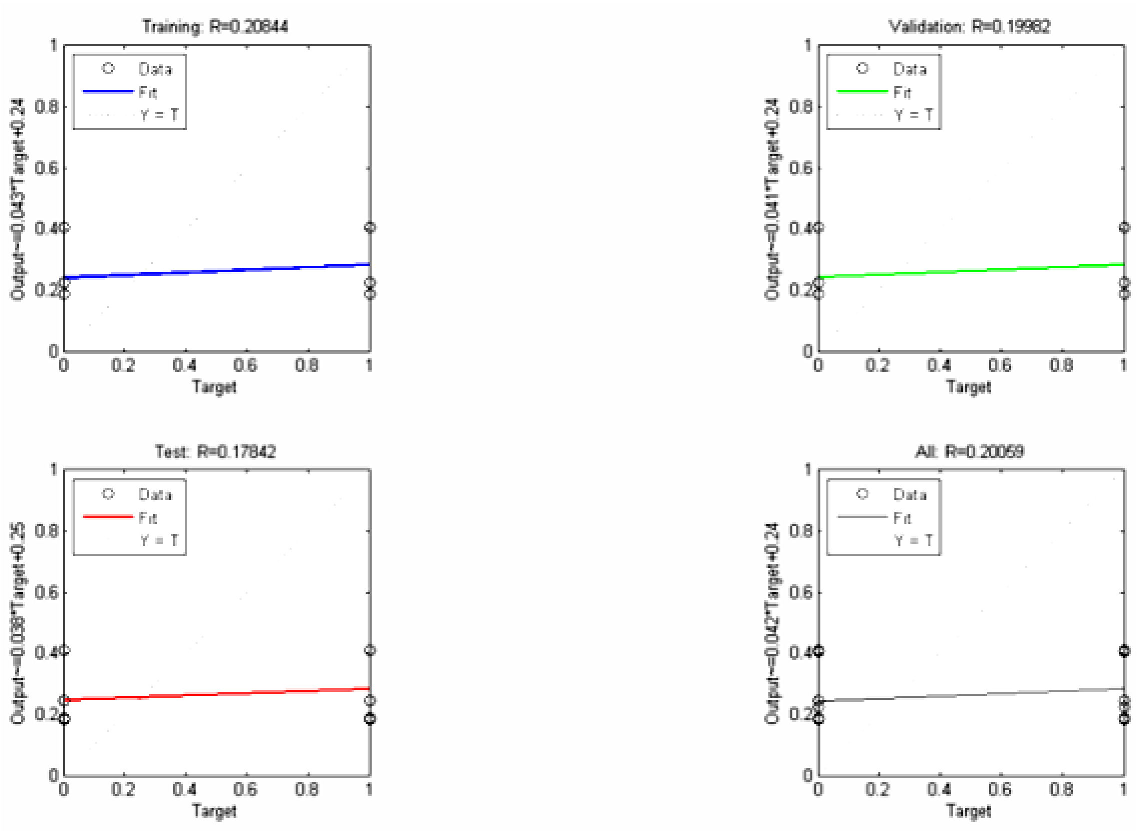
a) Levenberg–Marquardt



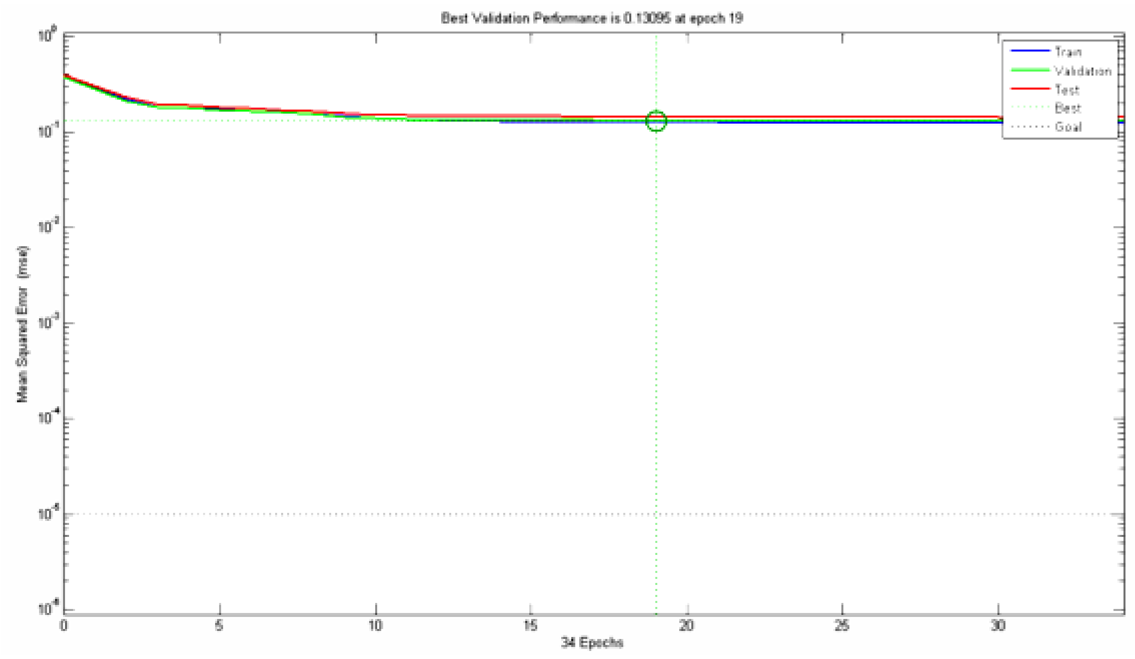


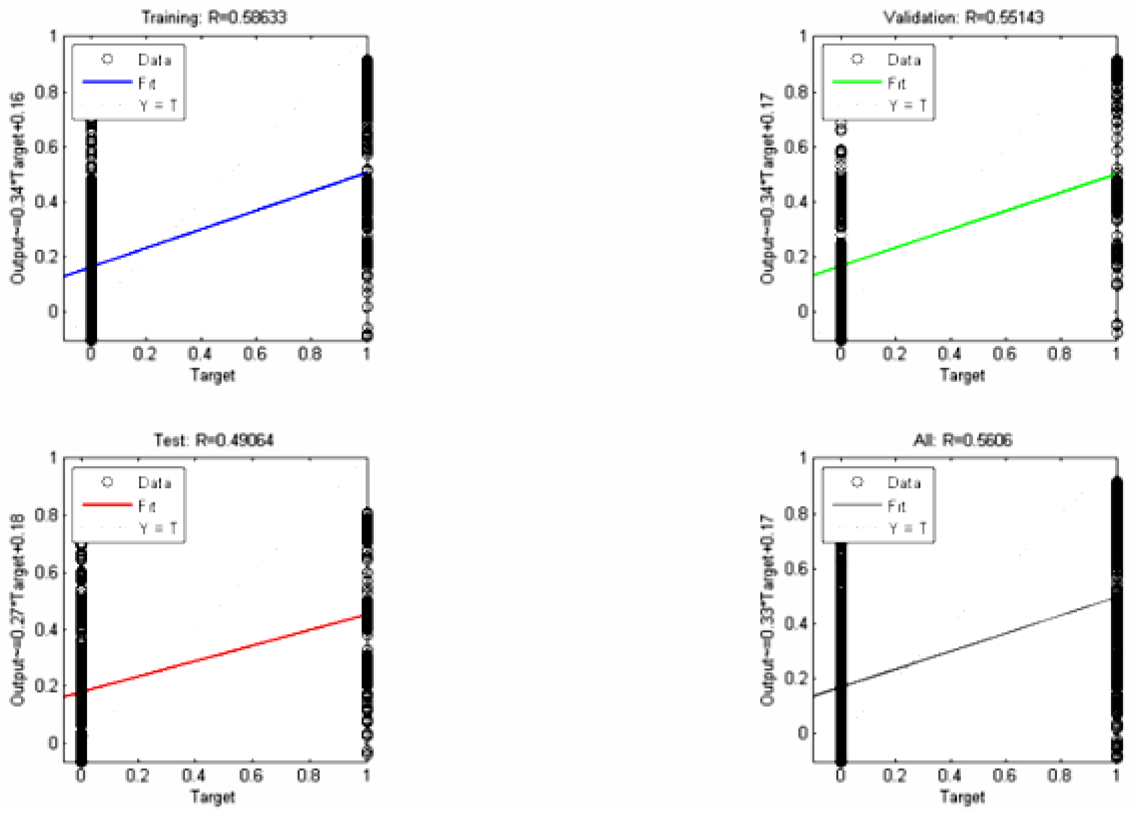
b) Bayesian Regularisation



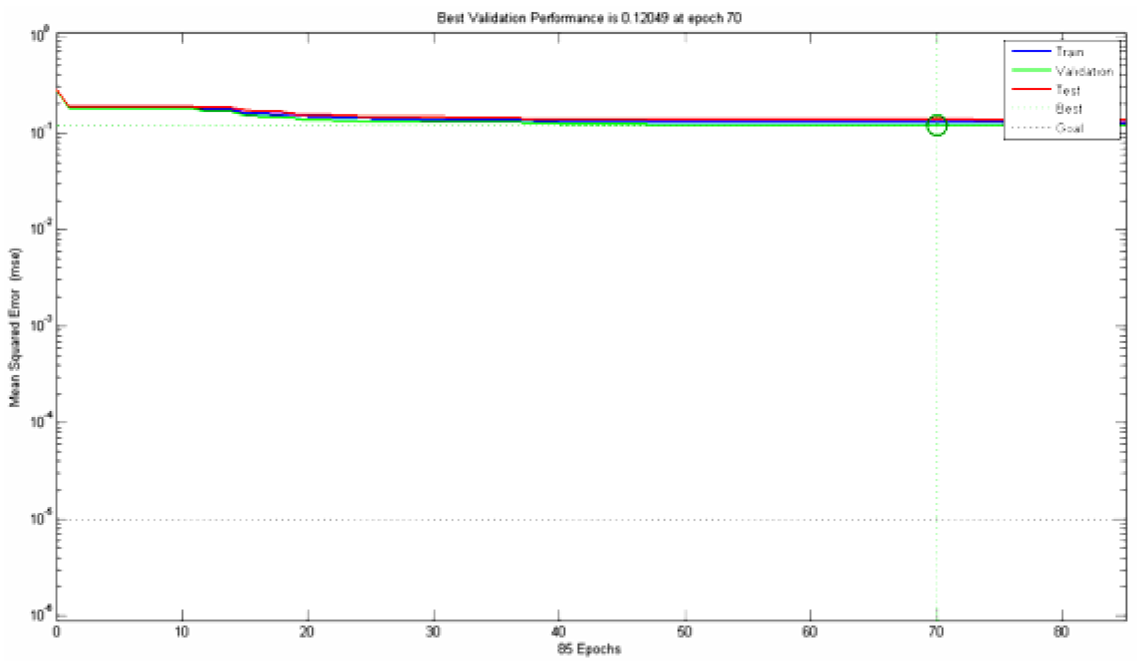


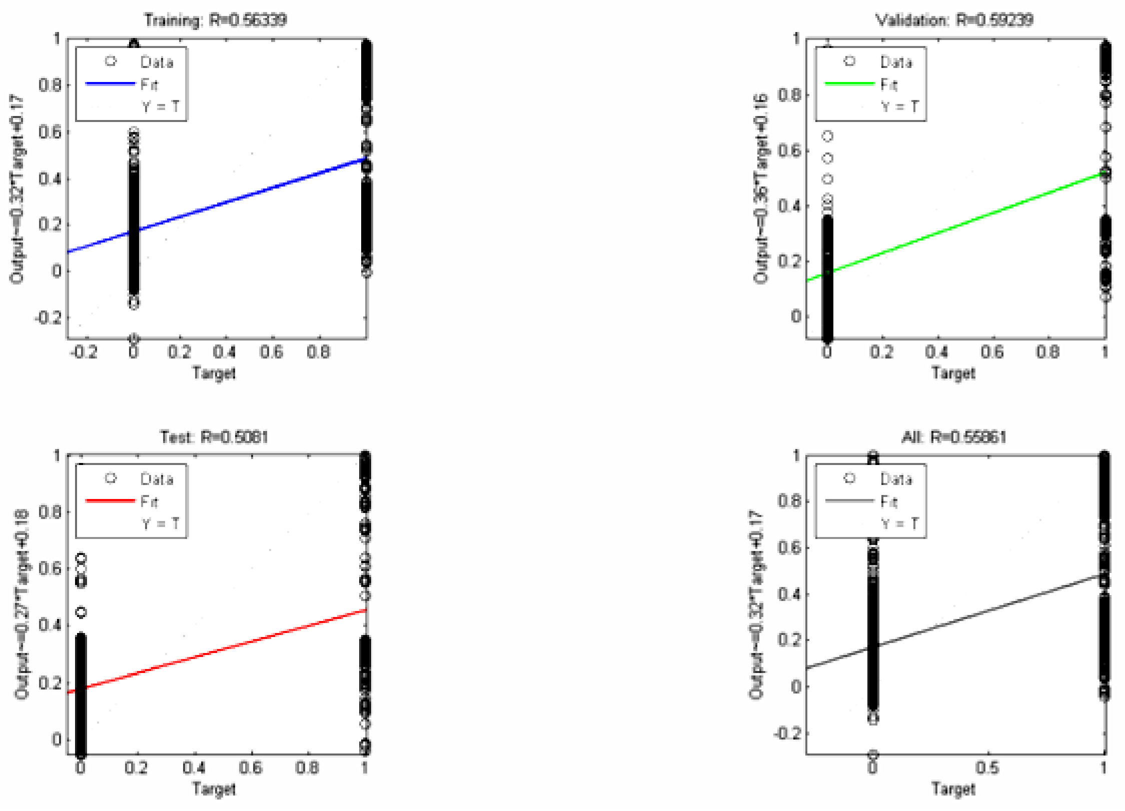
c) Resilient Backpropagation



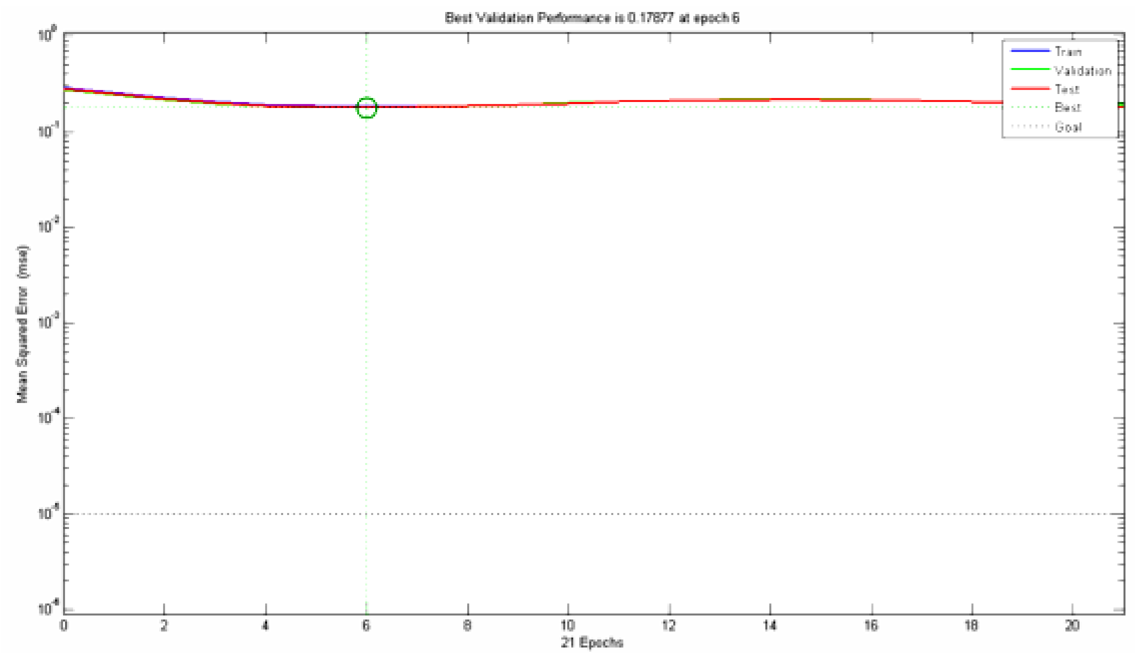


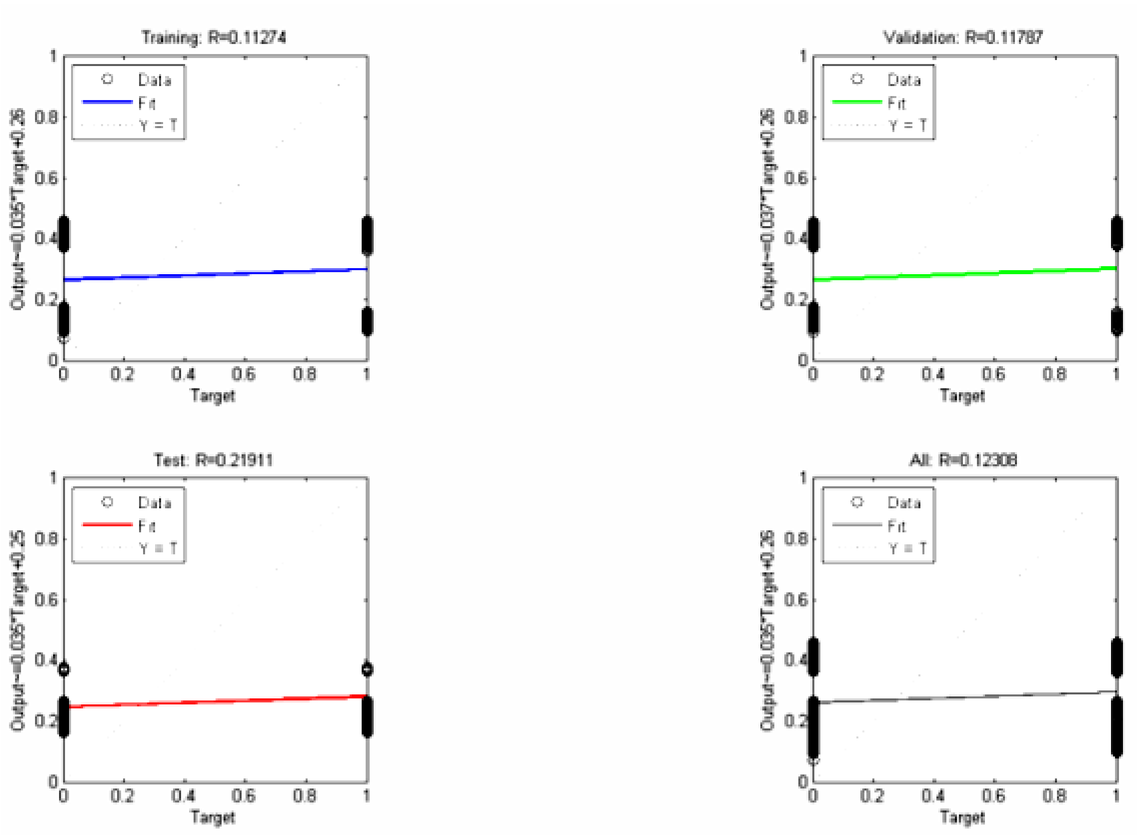
d) Scaled Conjugate Gradient



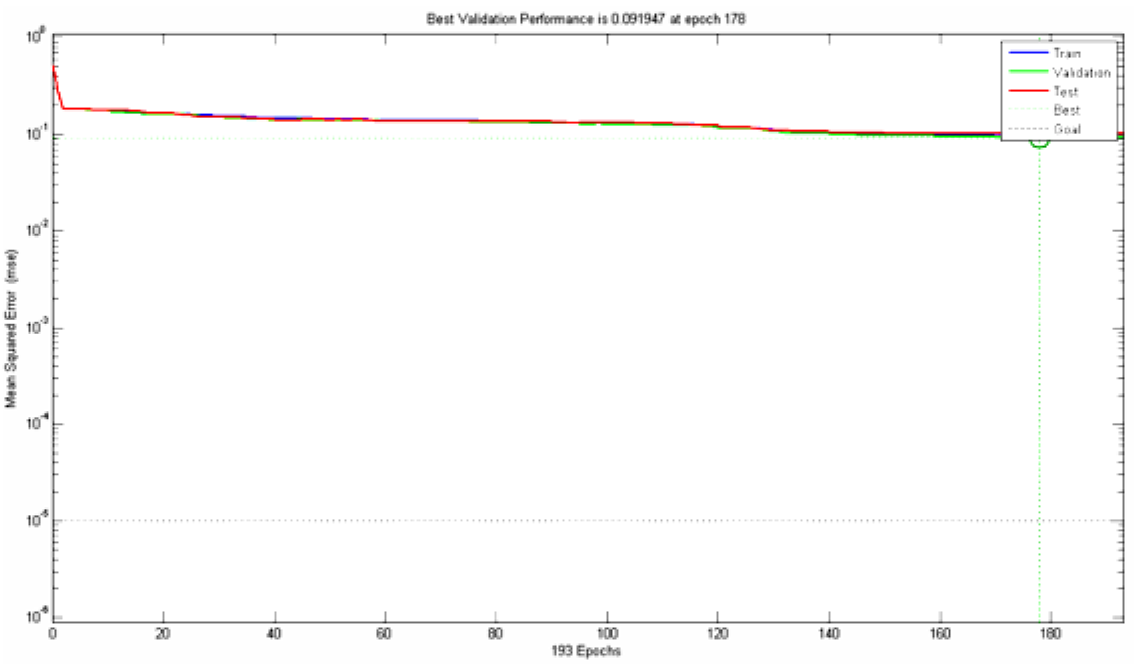


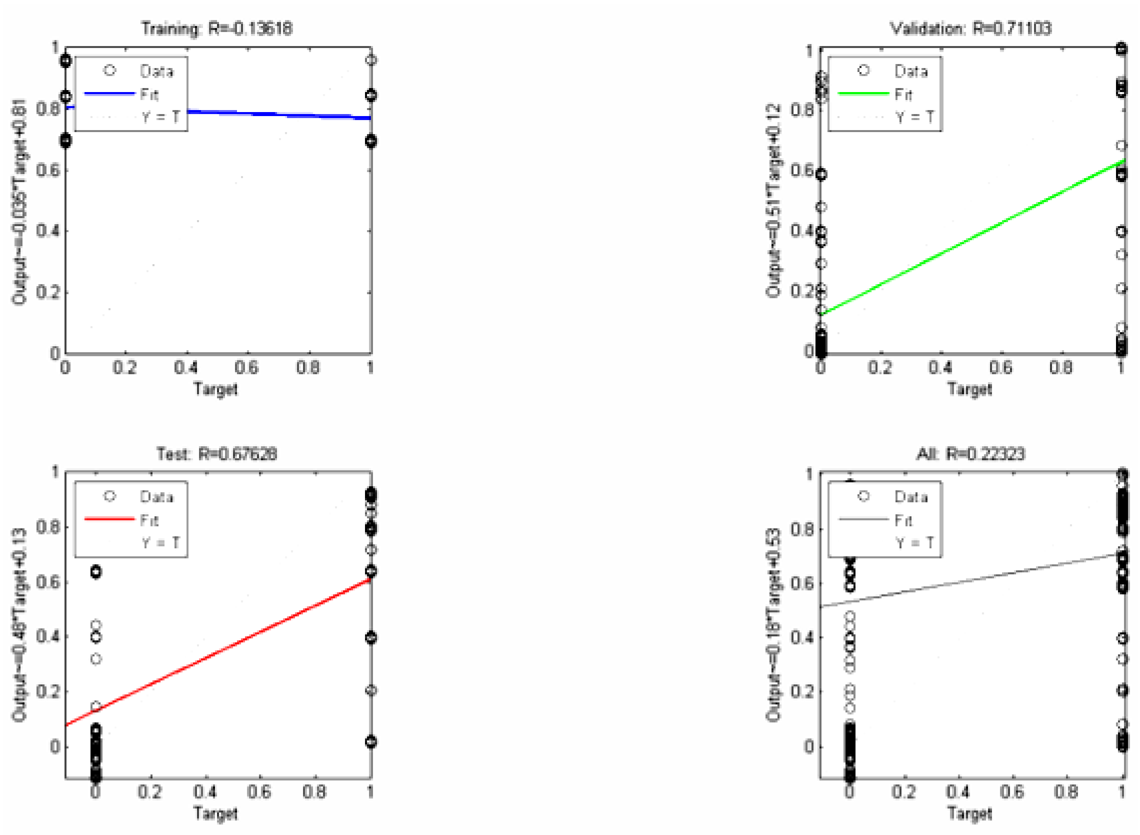
e) Variable Learning Rate Backpropagation





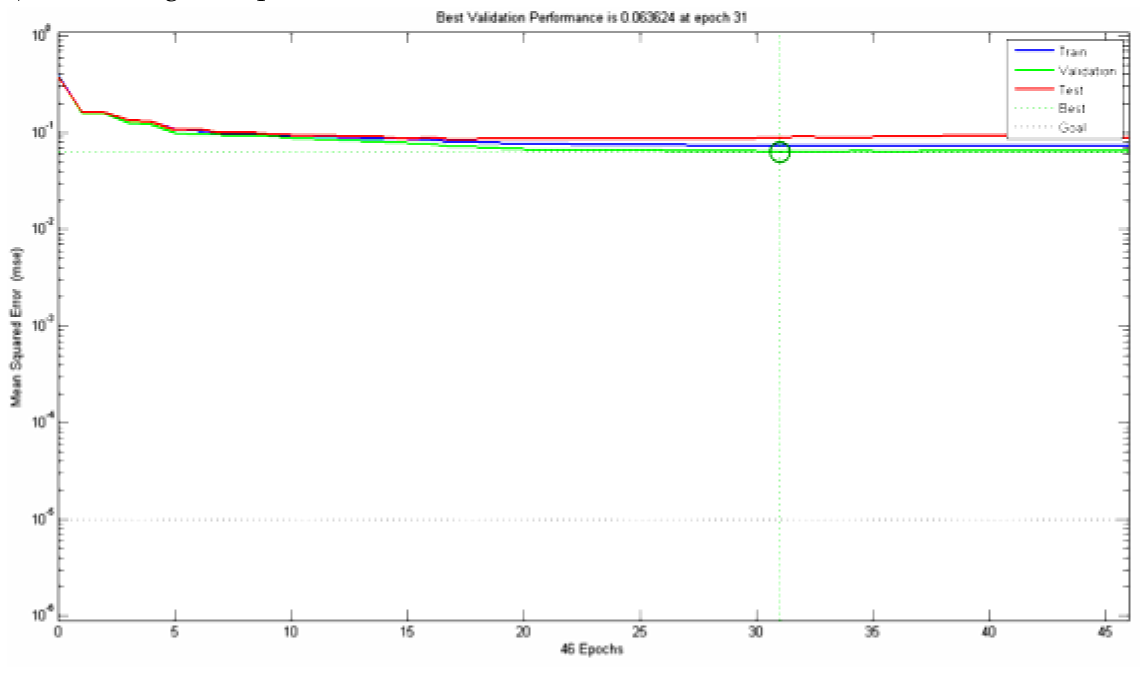
f) BFGS quasi–Newton Backpropagation

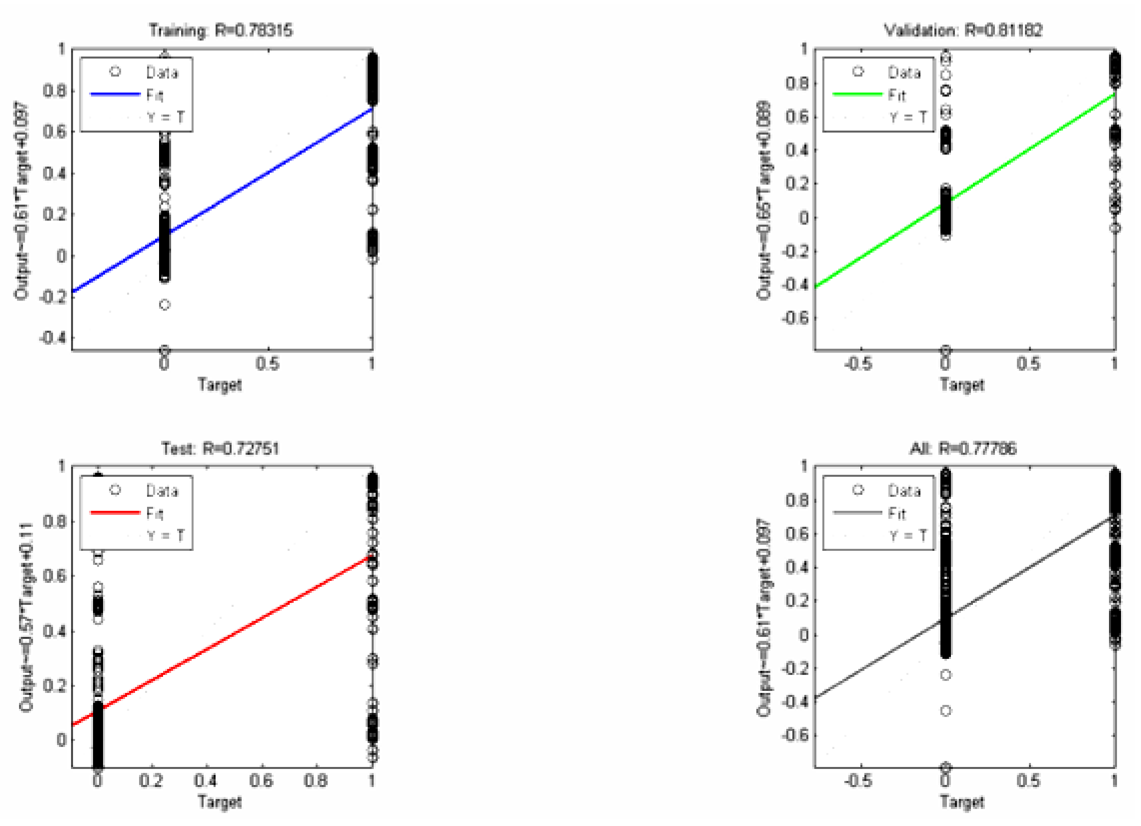




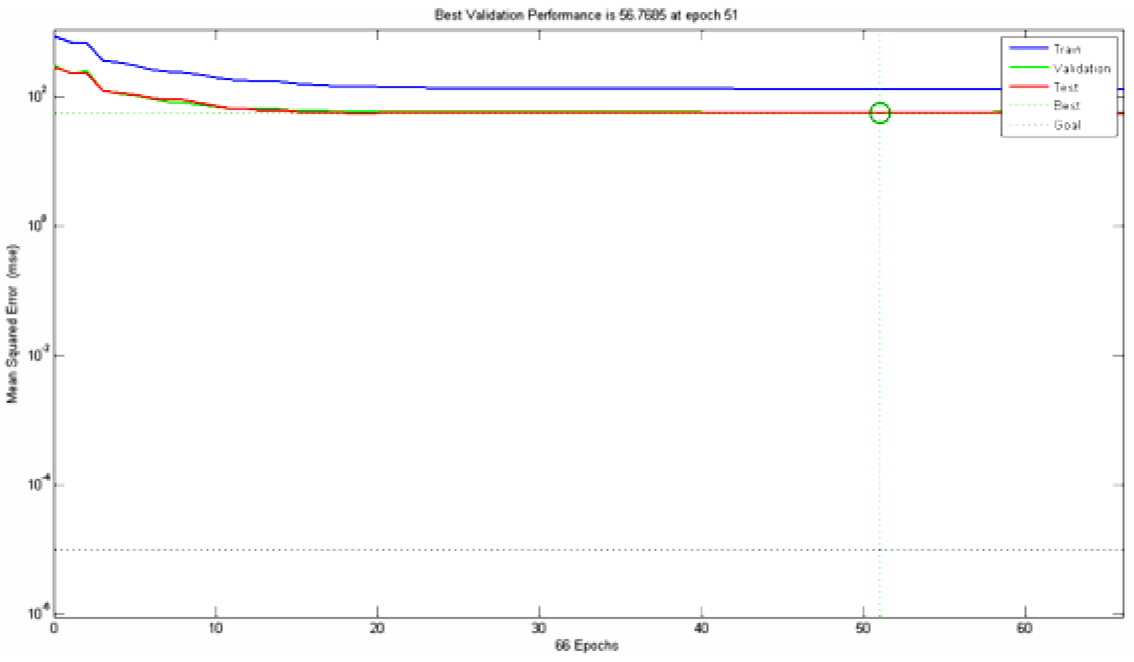
2) [3 2 2]

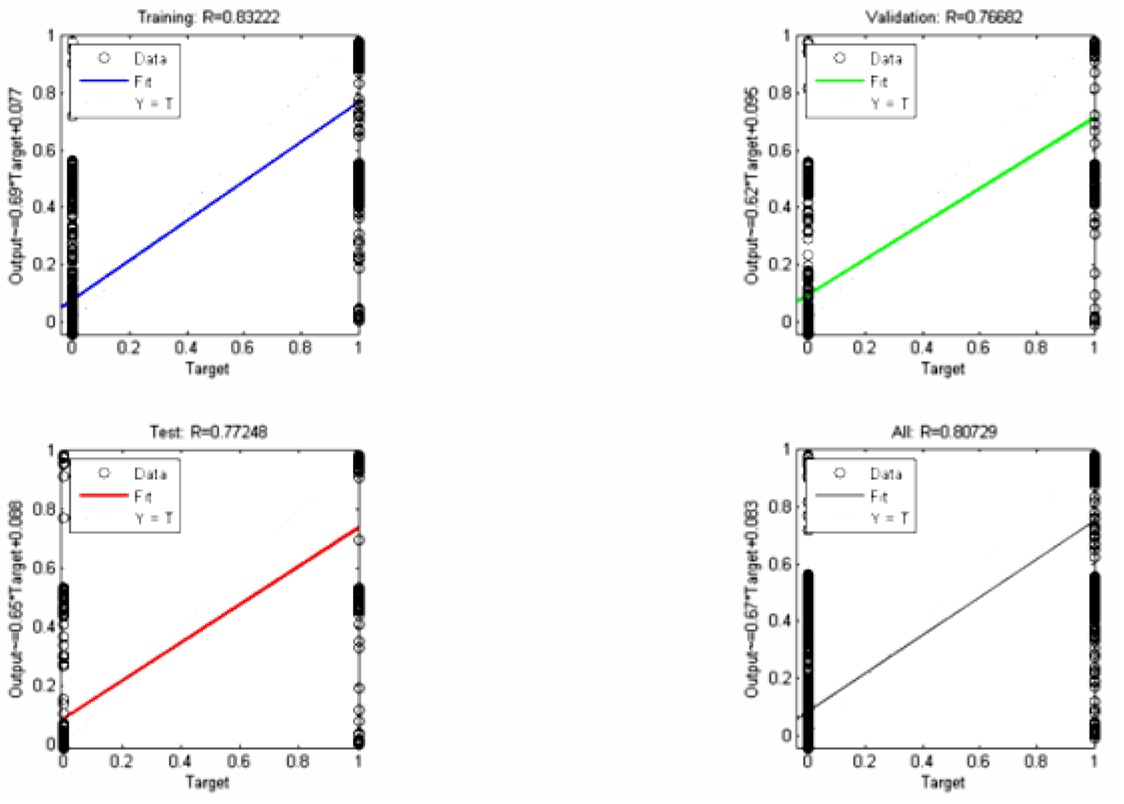
a) Levenberg–Marquardt



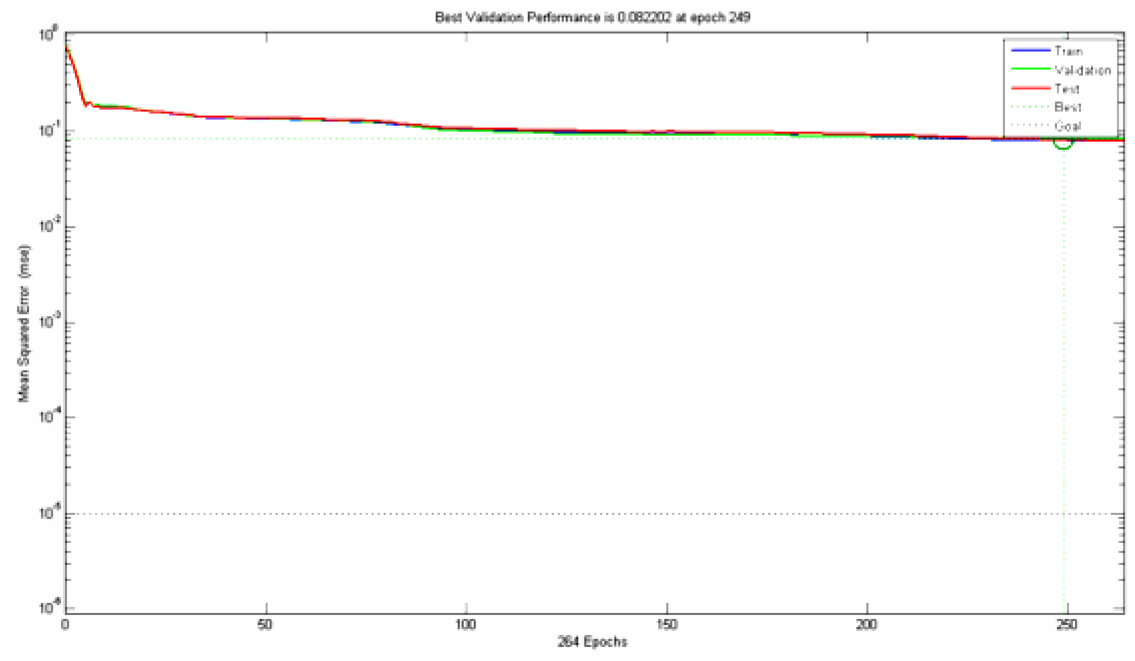


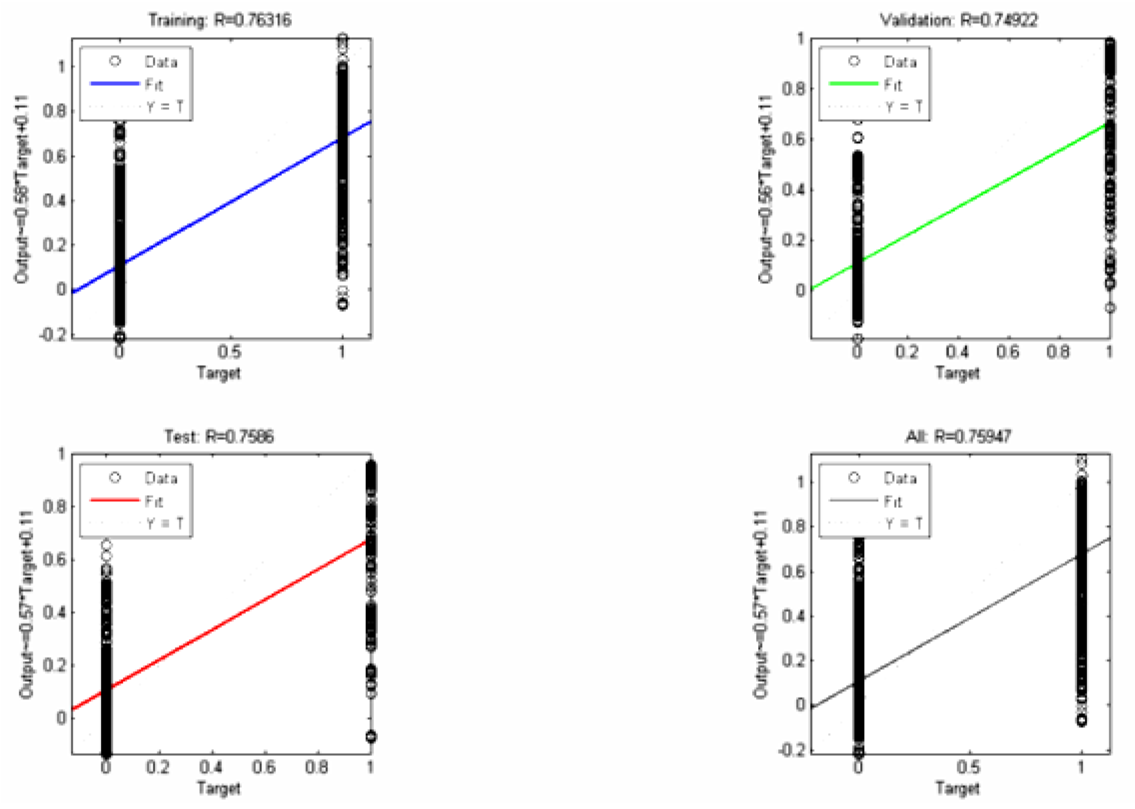
b) Bayesian Regularisation



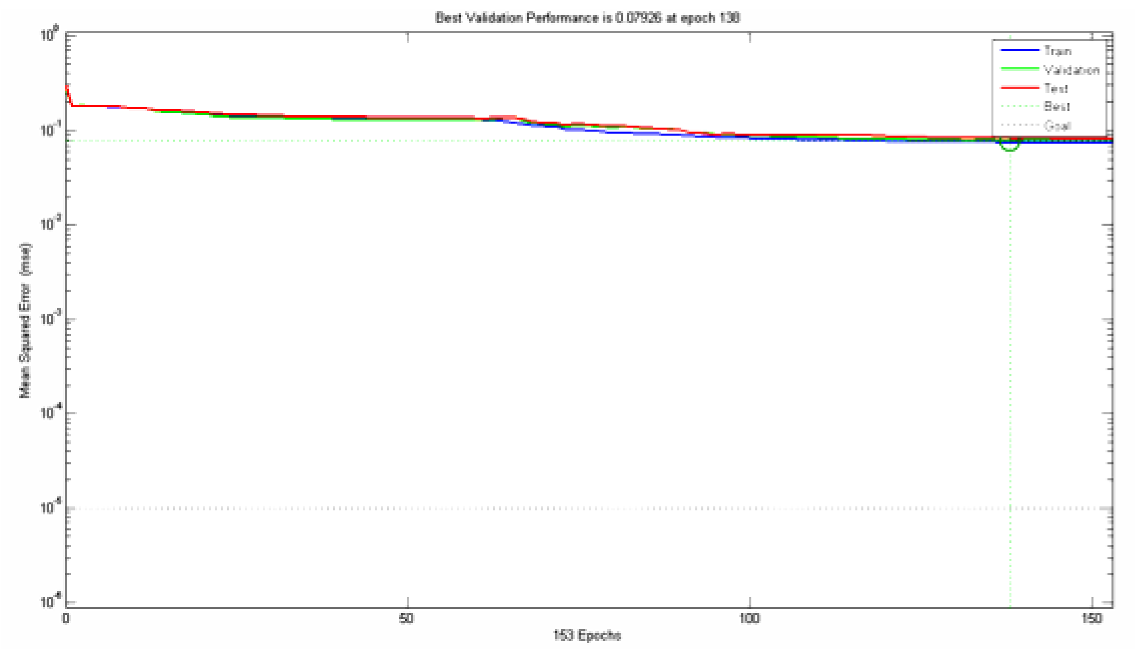


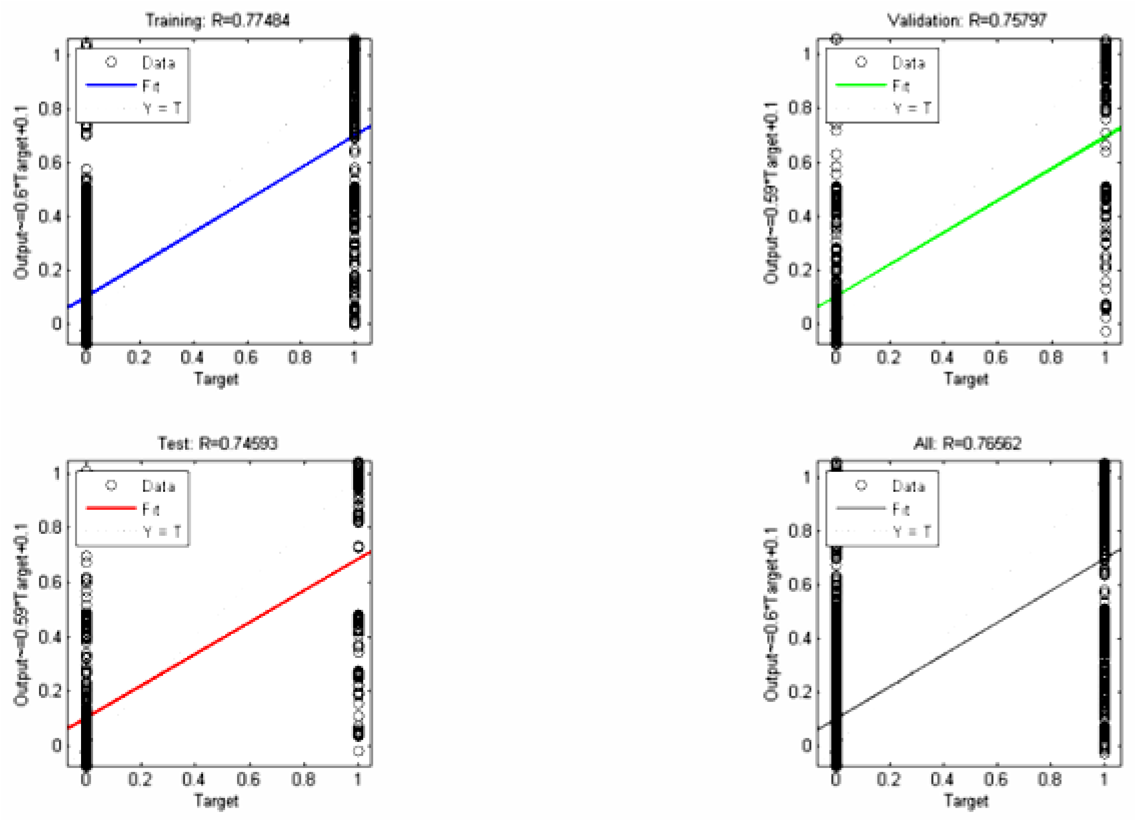
c) Resilient Backpropagation



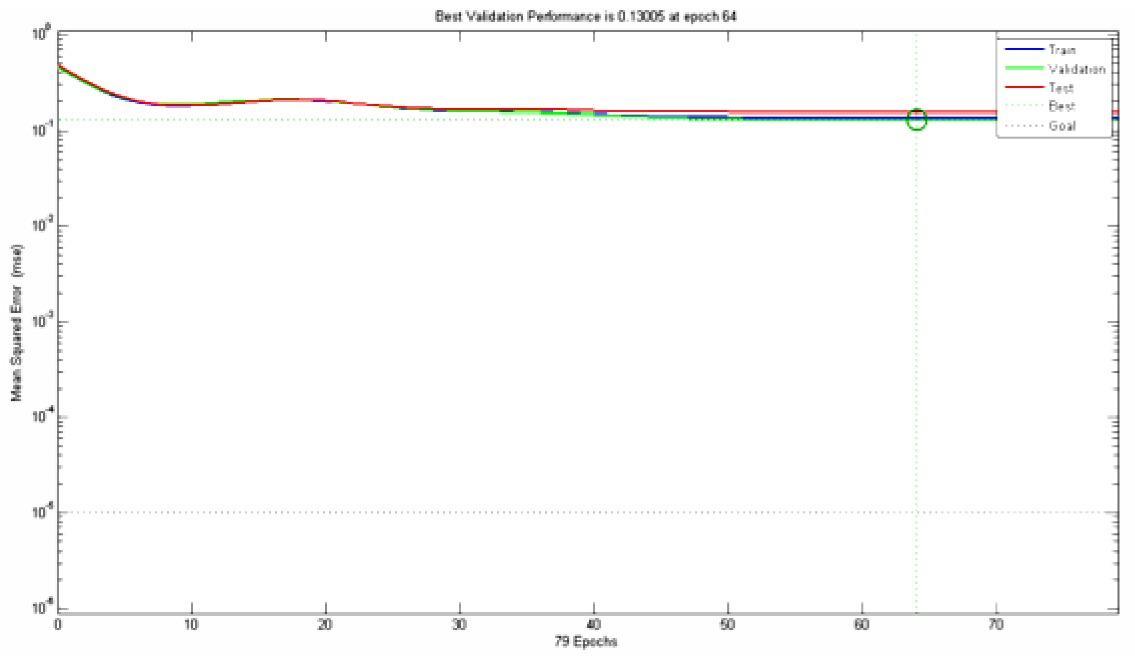


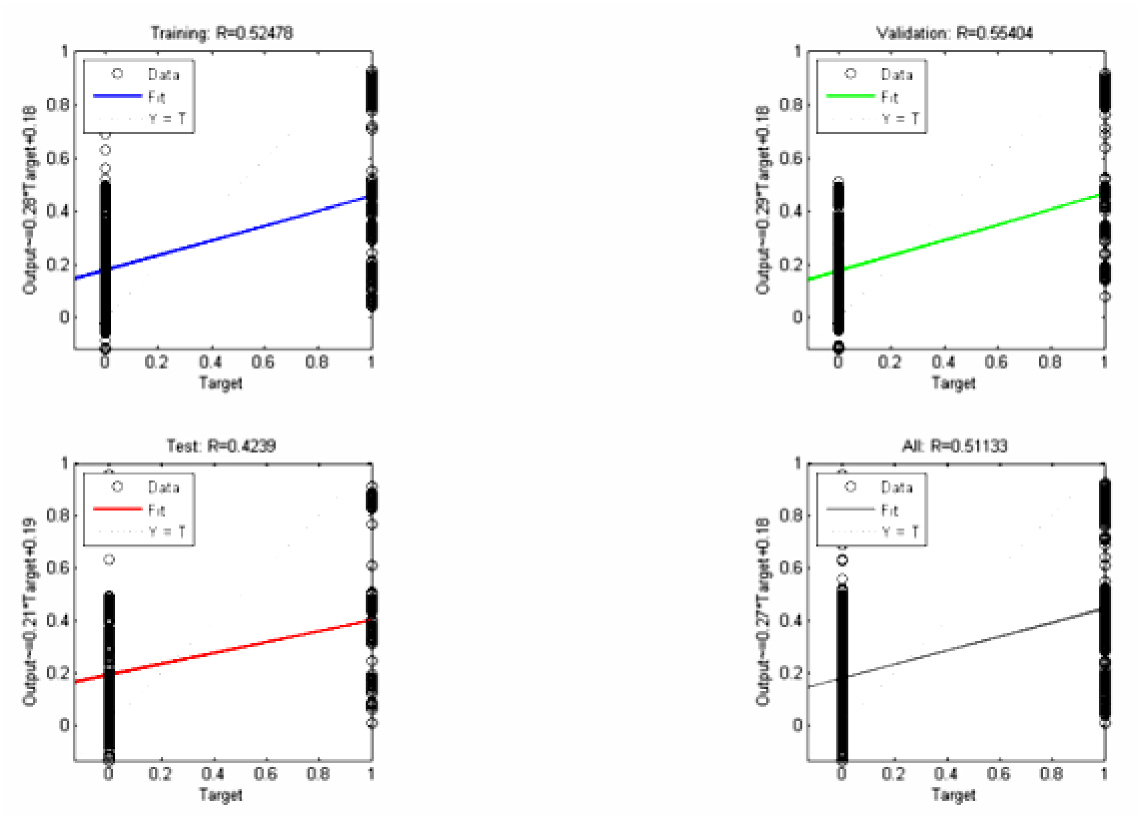
d) Scaled Conjugate Gradient



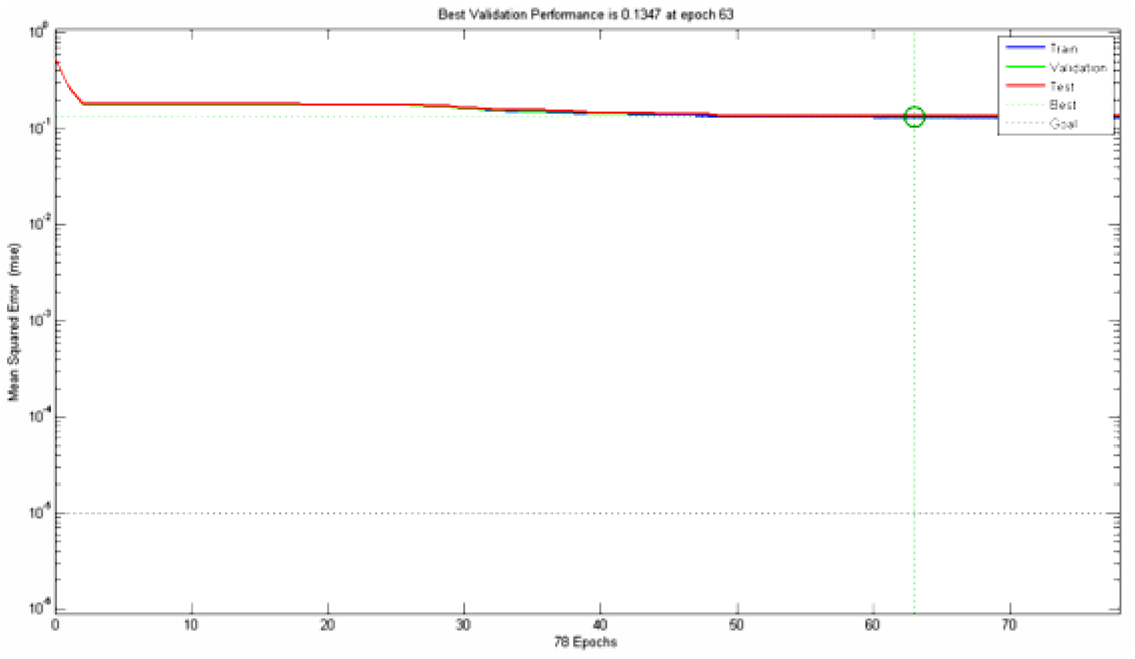


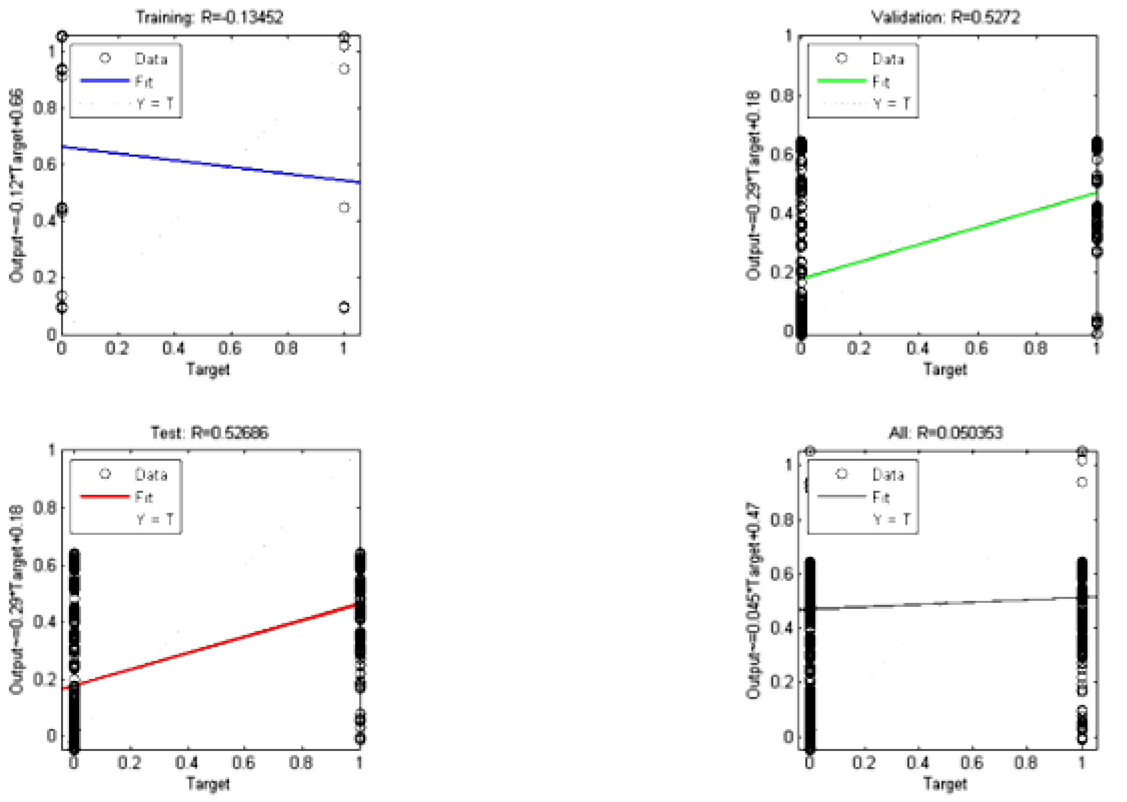
e) Variable Learning Rate Backpropagation





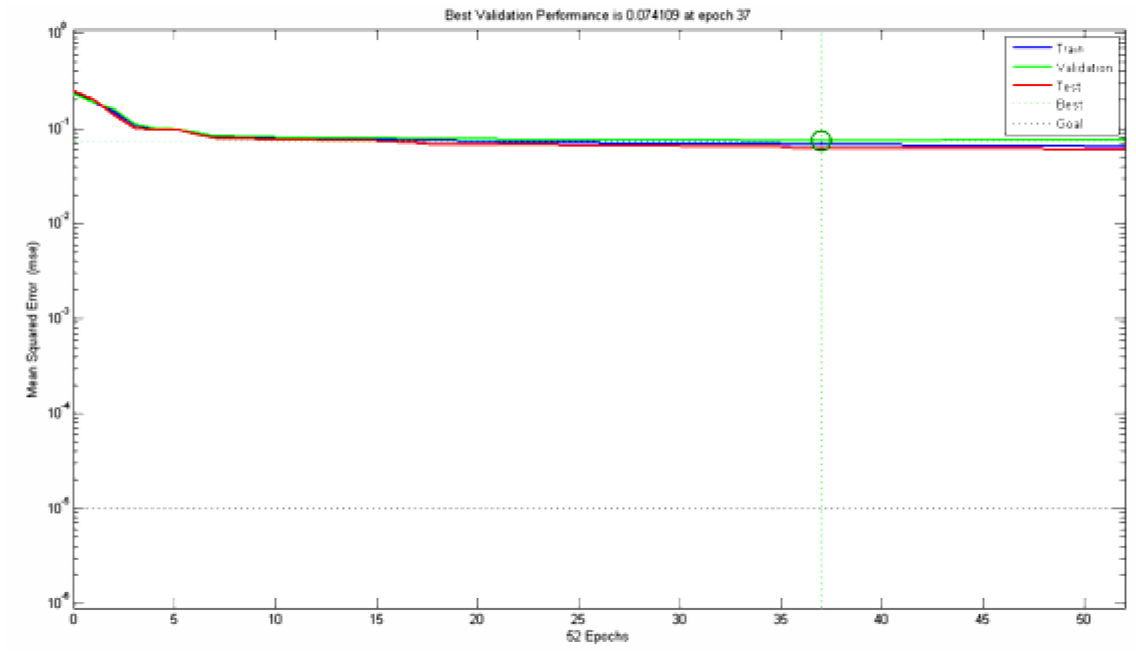
f) BFGS quasi–Newton Backpropagation

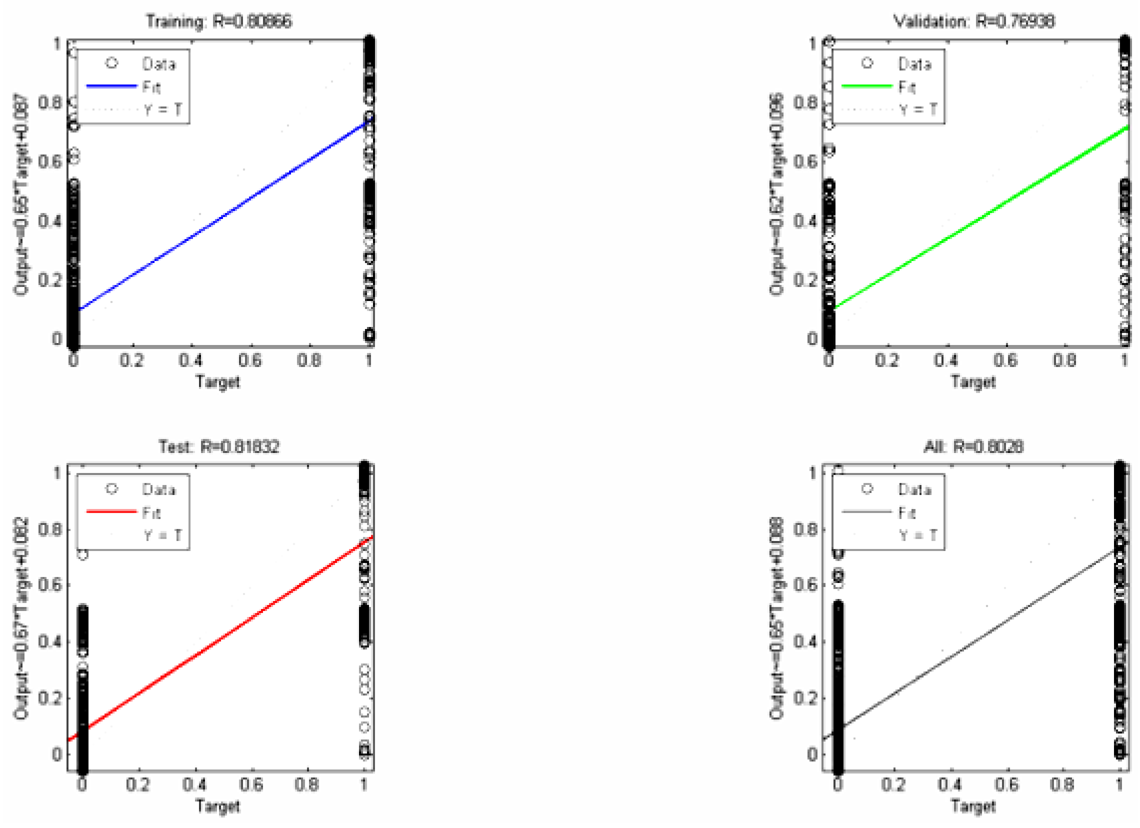




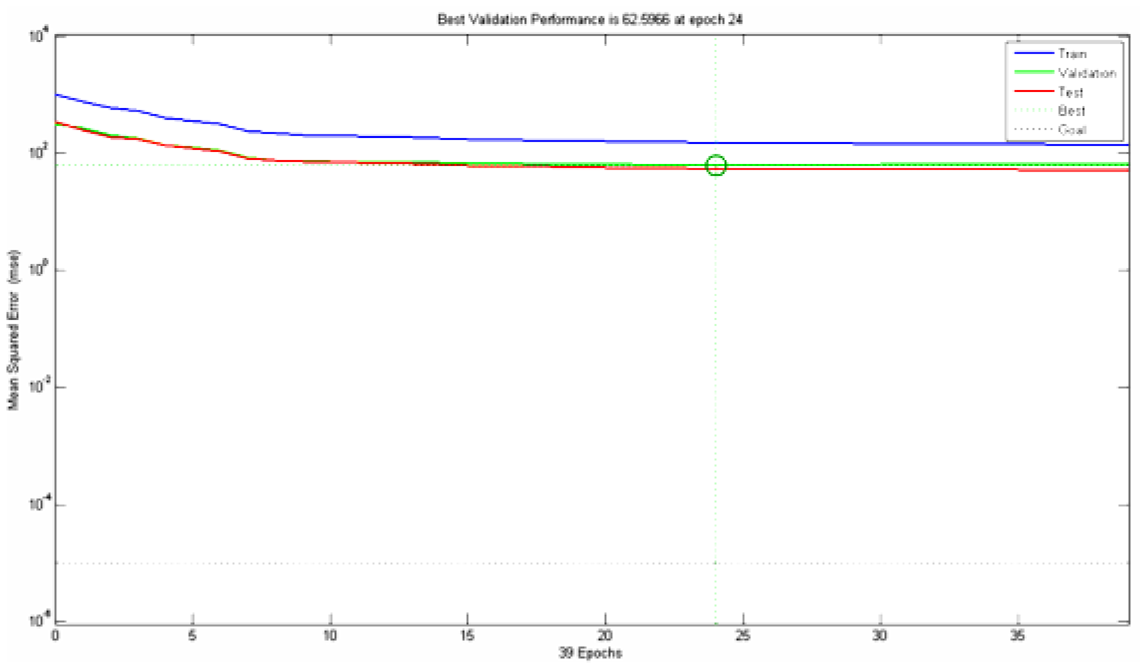
3) [4 2 2]

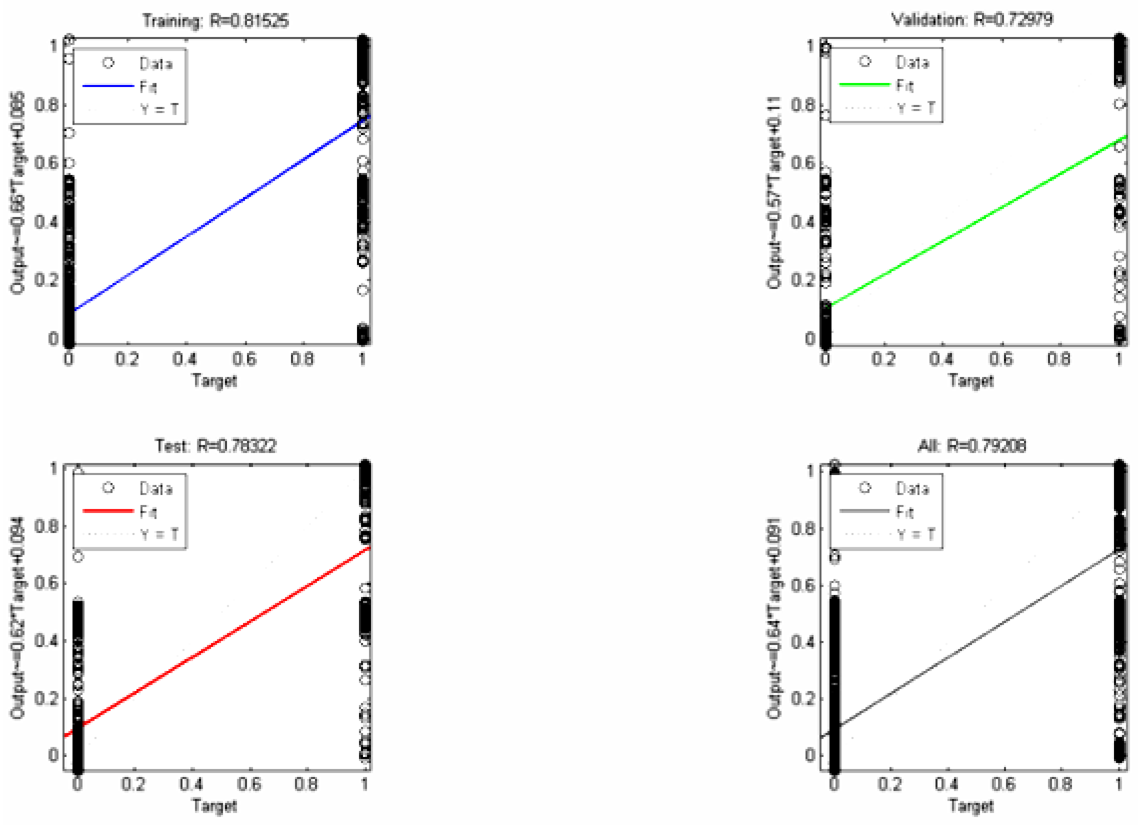
a) Levenberg–Marquardt



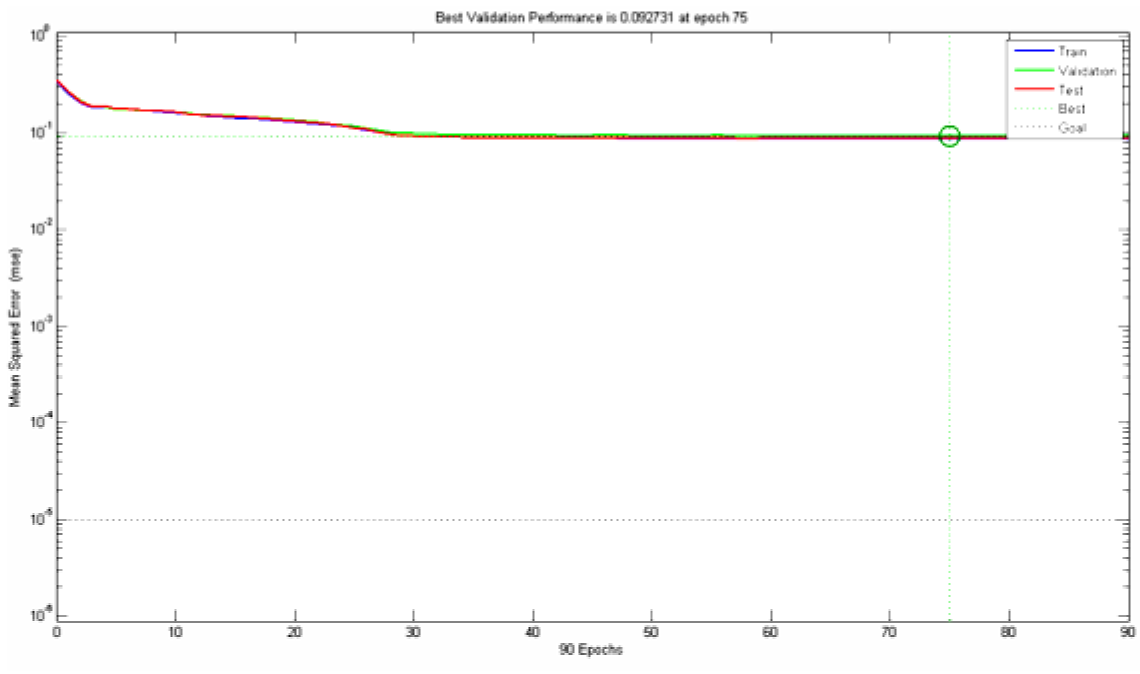


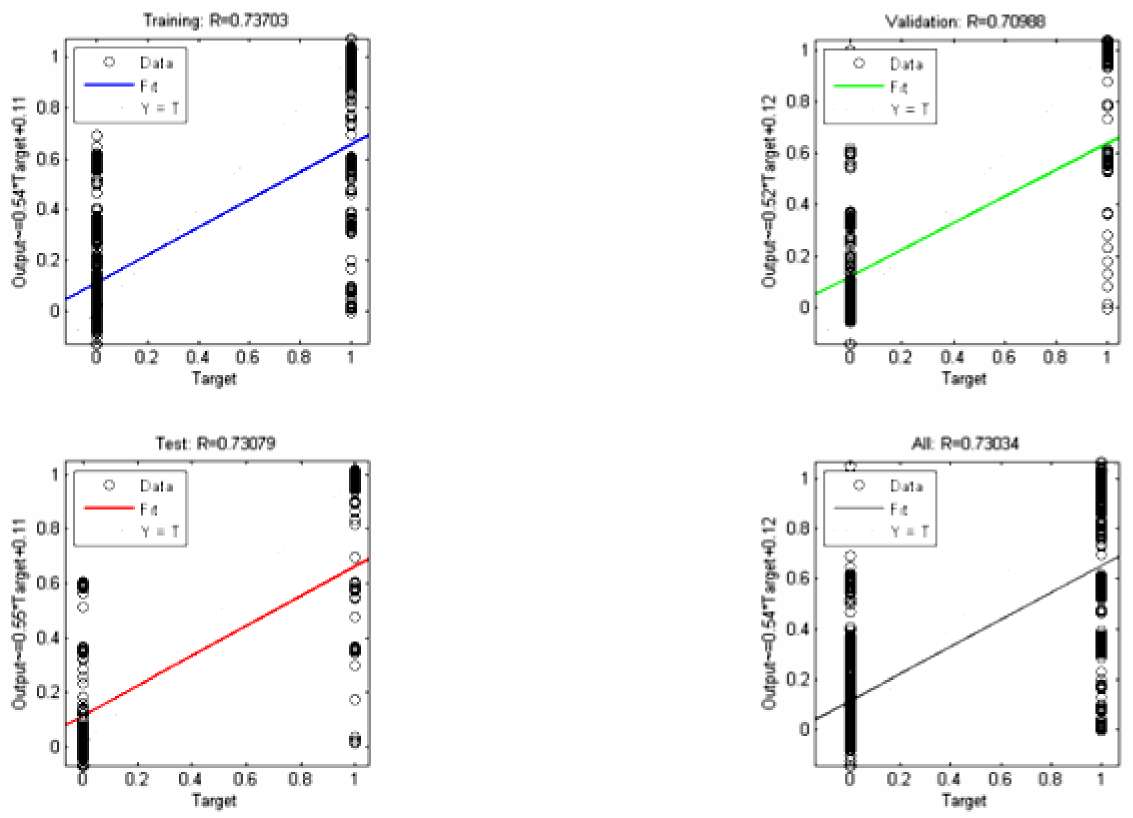
b) Bayesian Regularisation



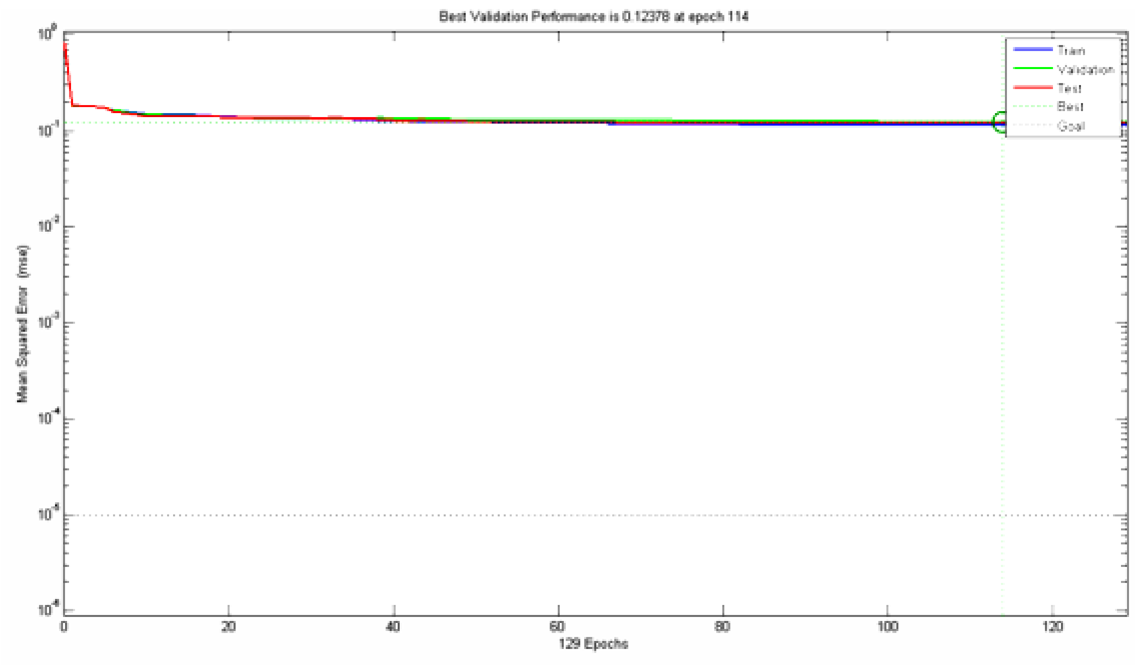


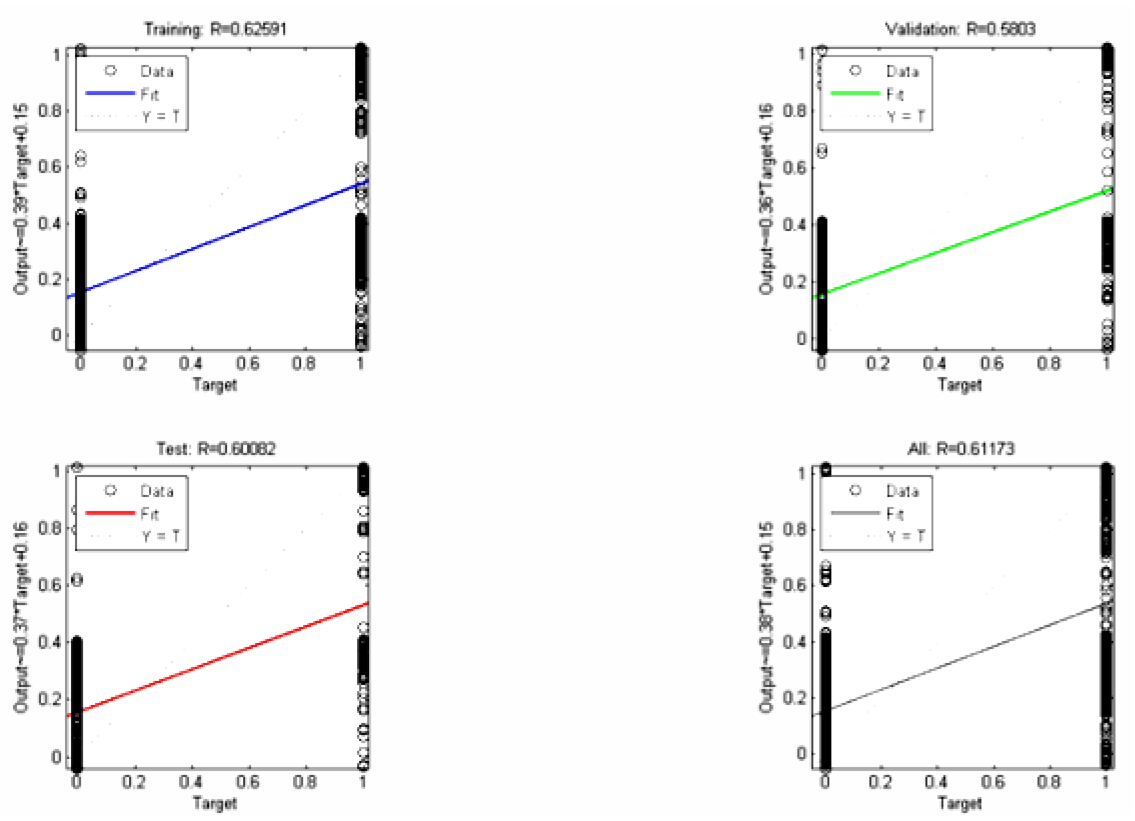
c) Resilient Backpropagation



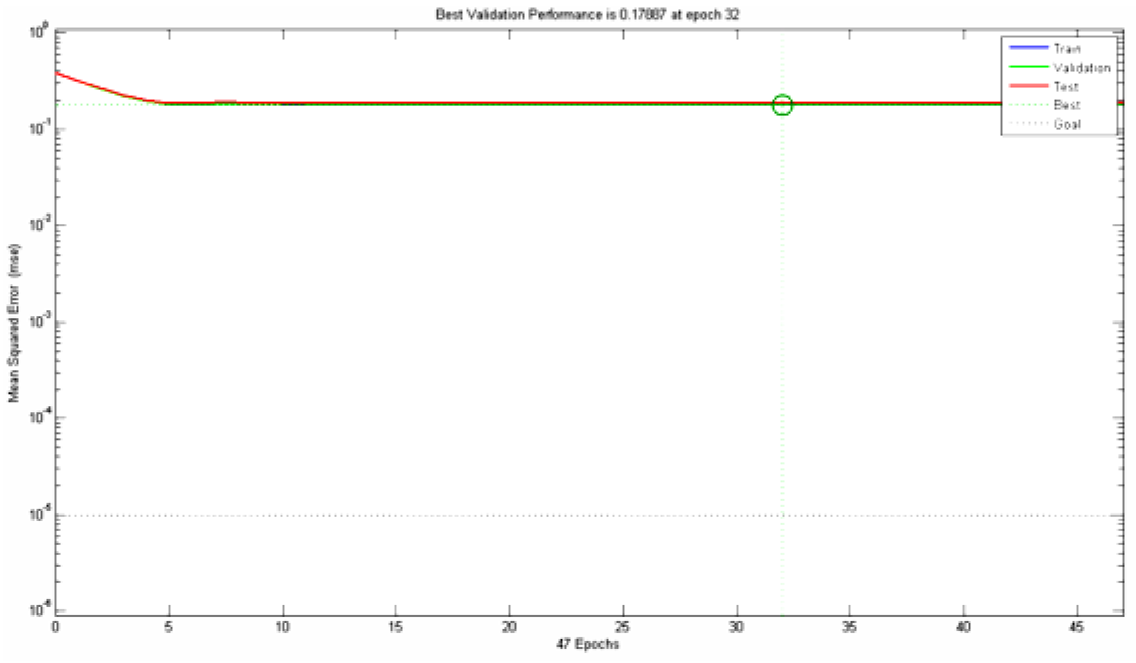


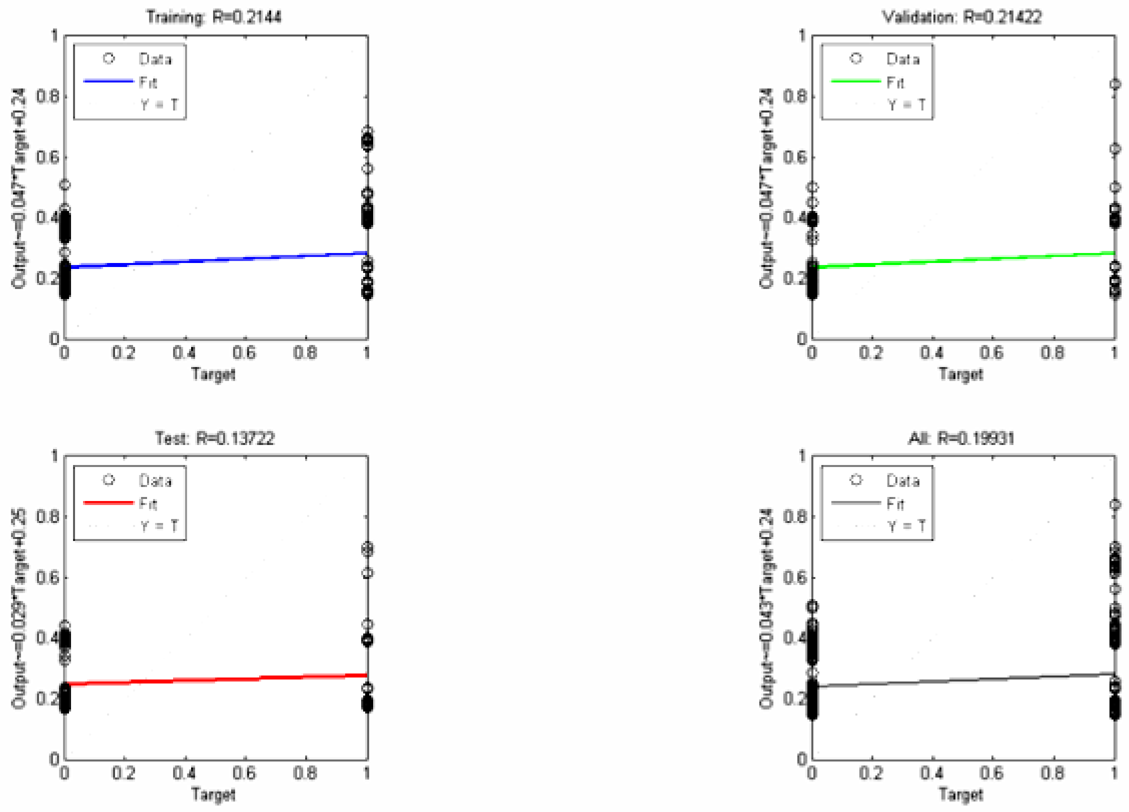
d) Scaled Conjugate Gradient



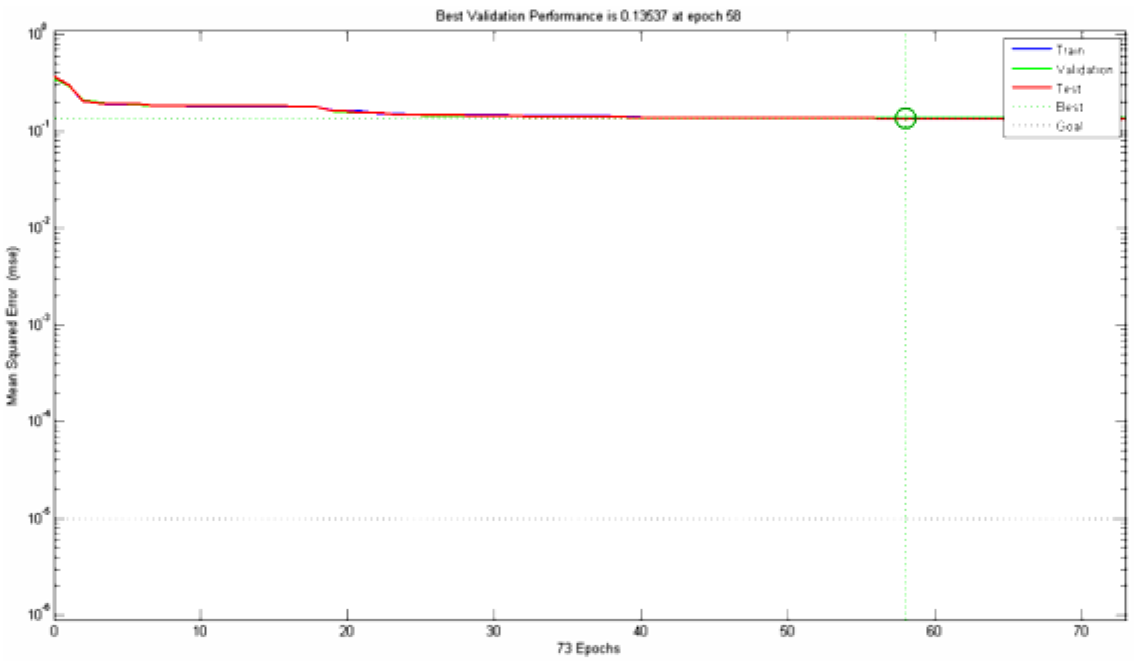


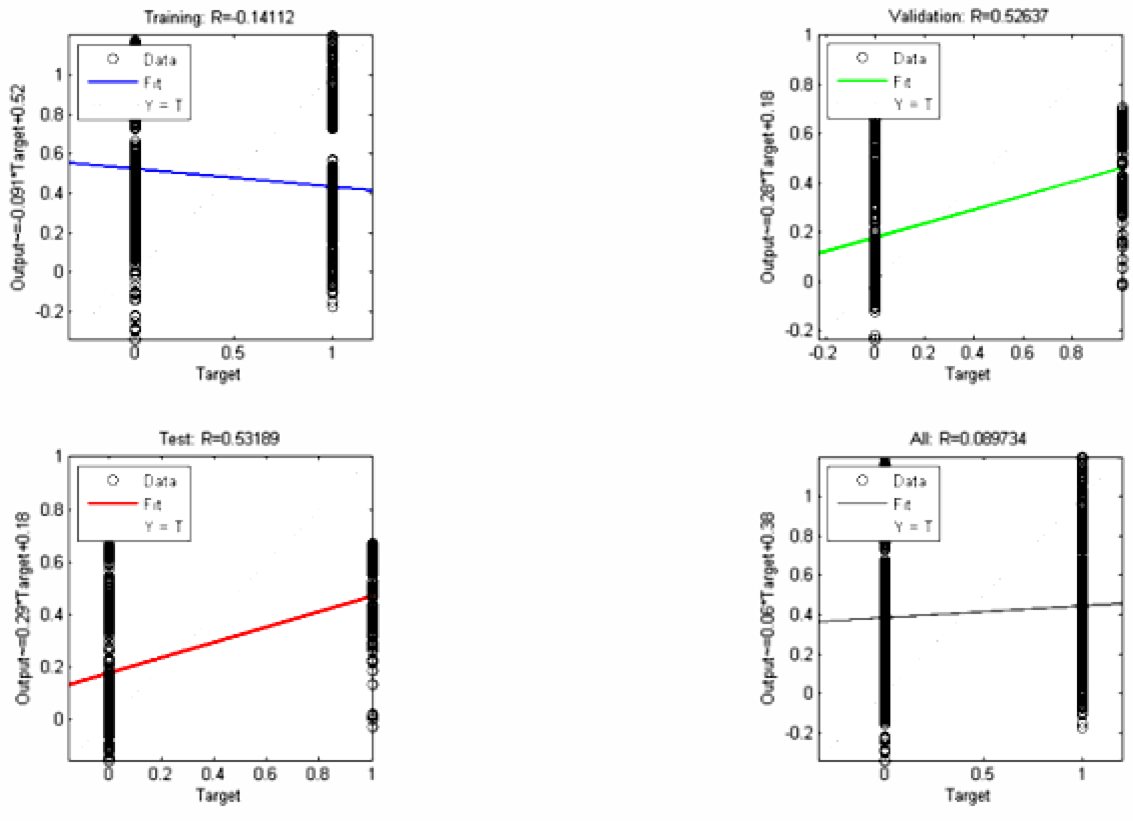
e) Variable Learning Rate Backpropagation





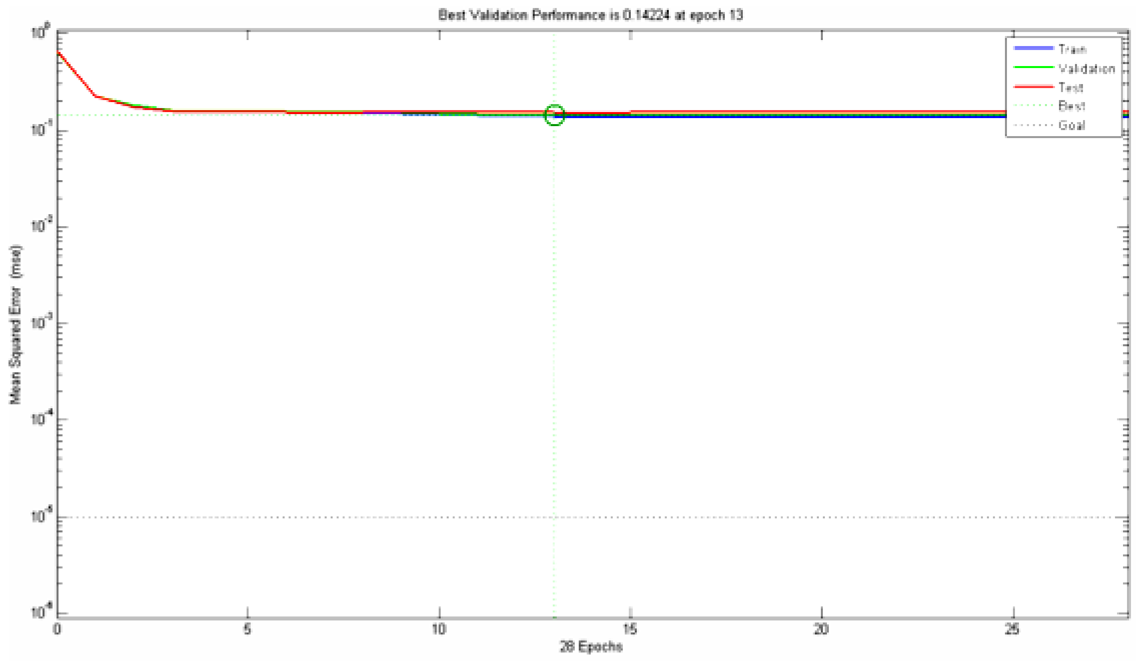
f) BFGS quasi–Newton Backpropagation

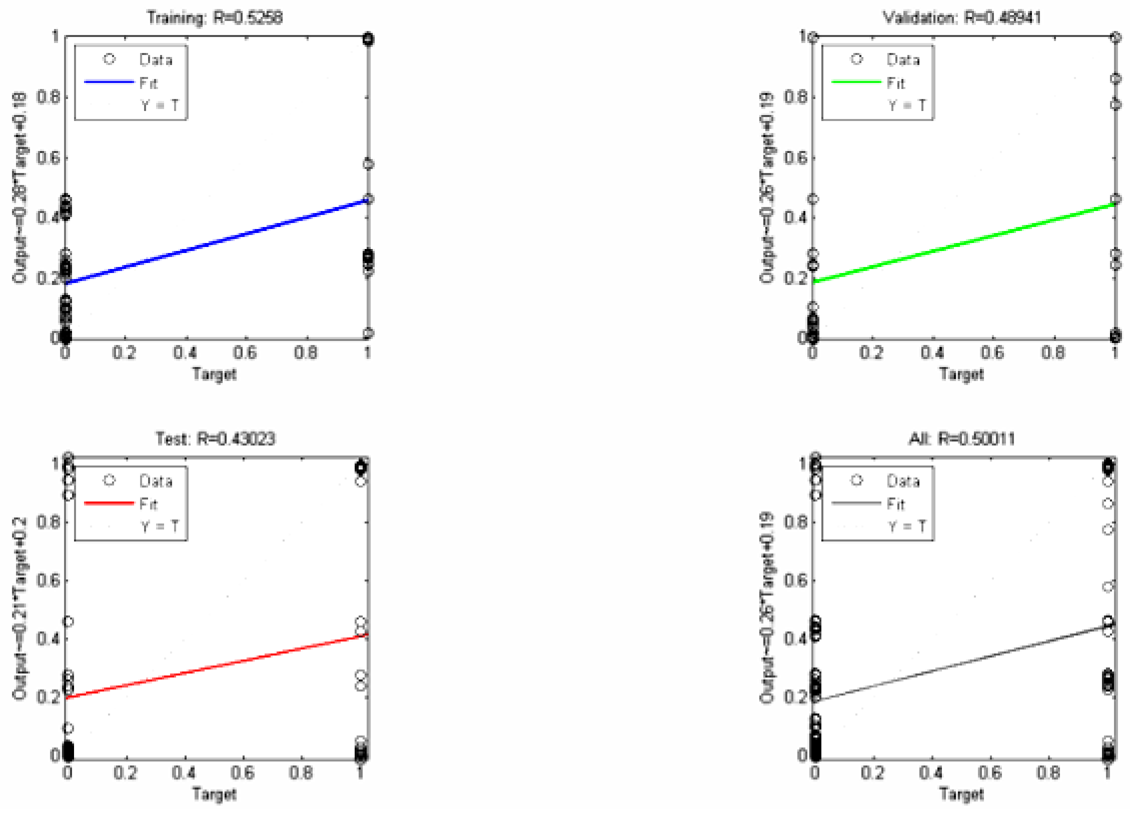




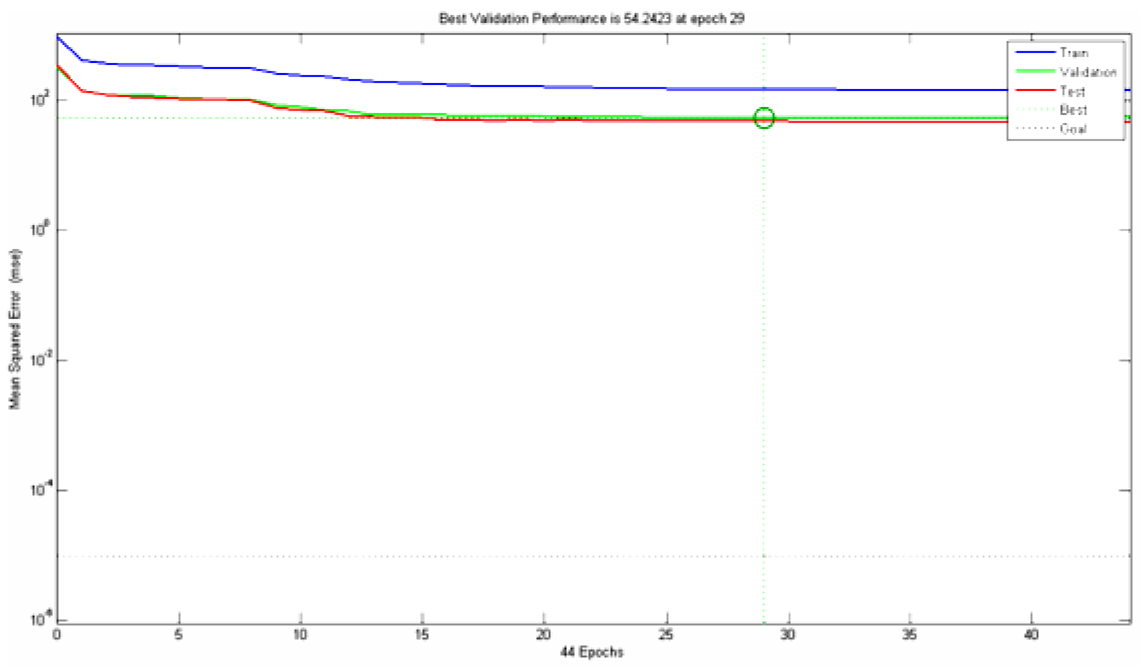
4) [5 3 2]

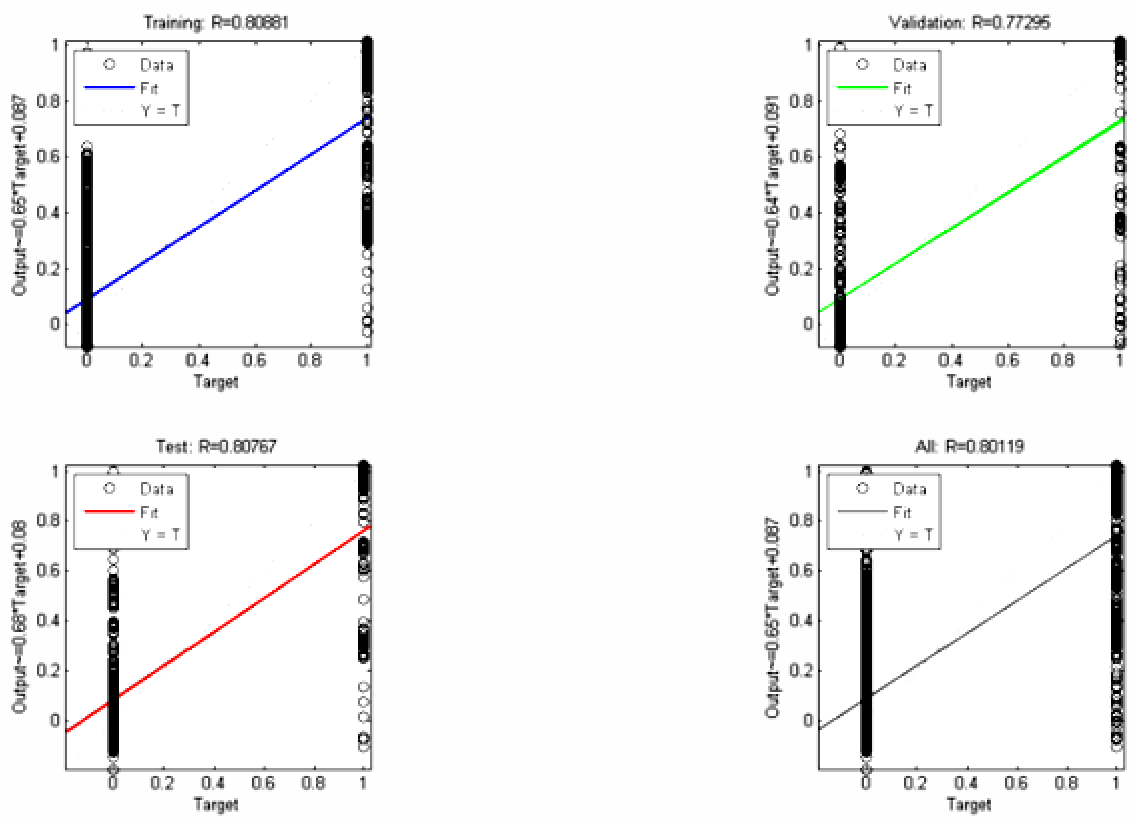
a) Levenberg–Marquardt



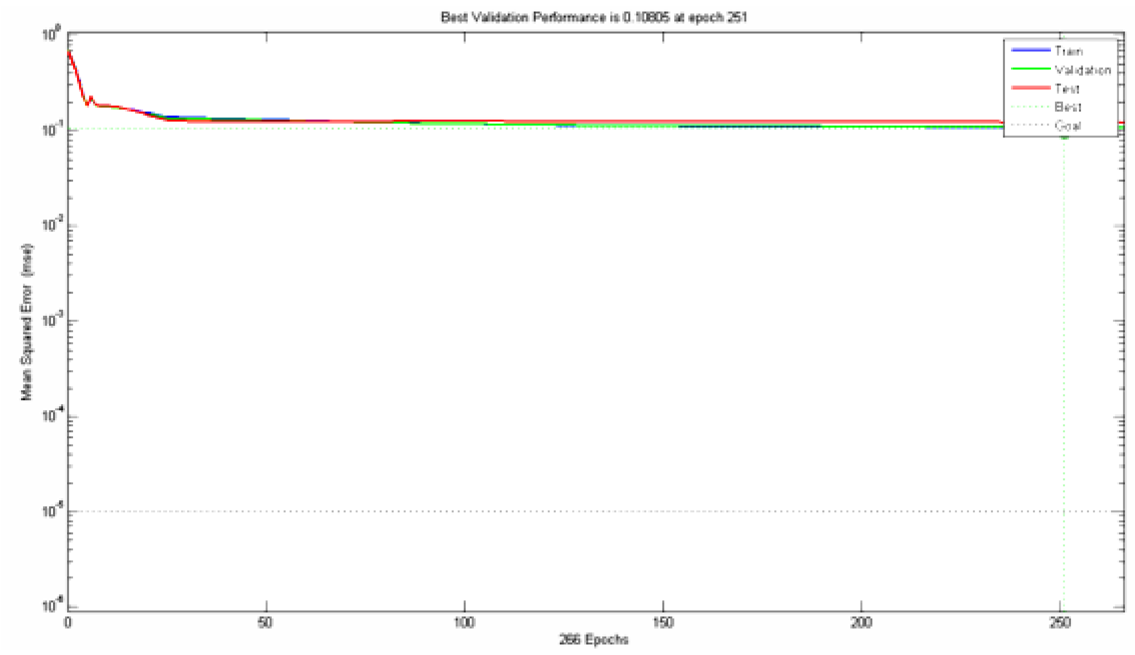


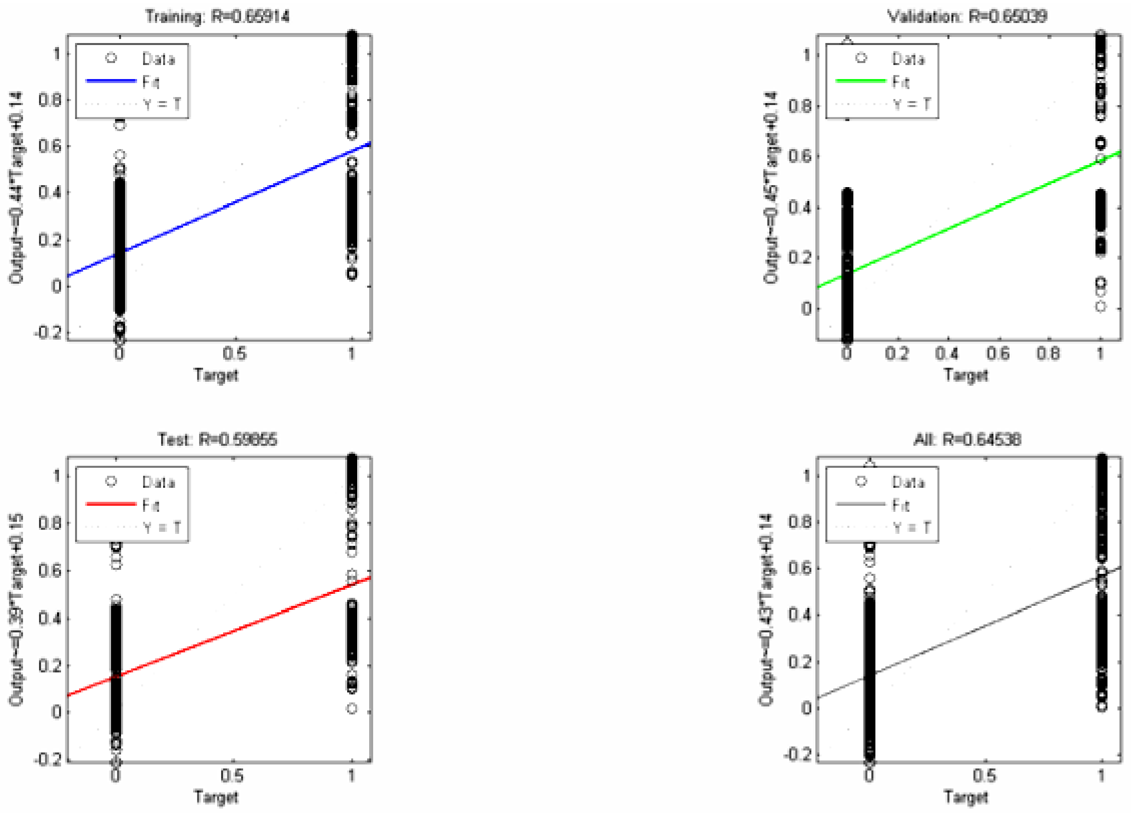
b) Bayesian Regularisation



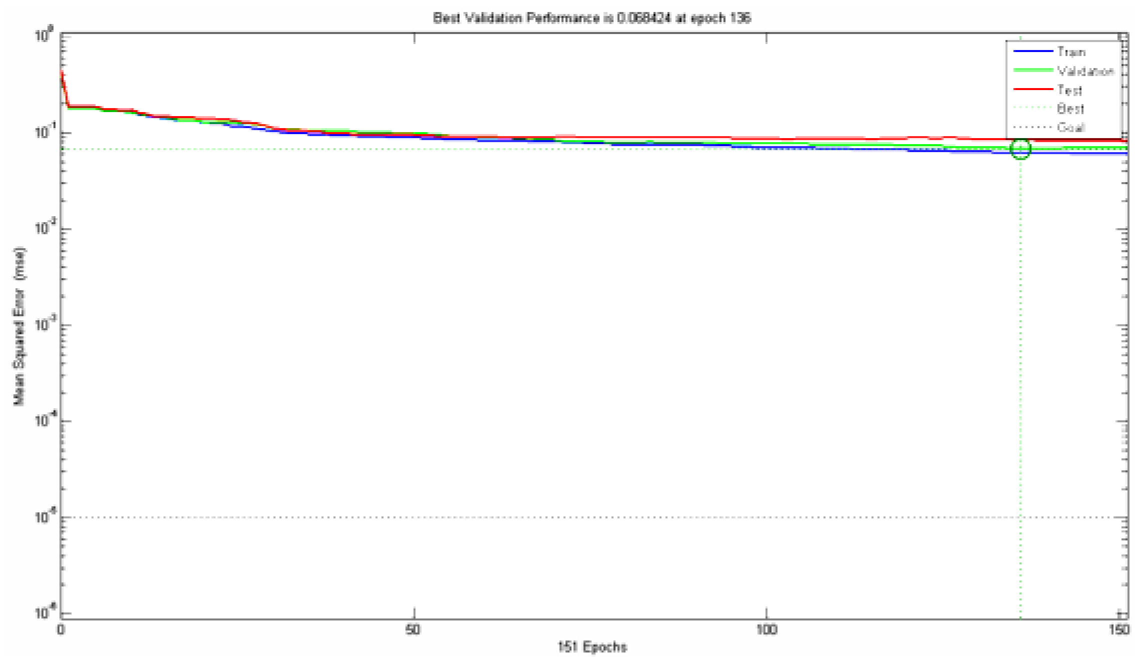


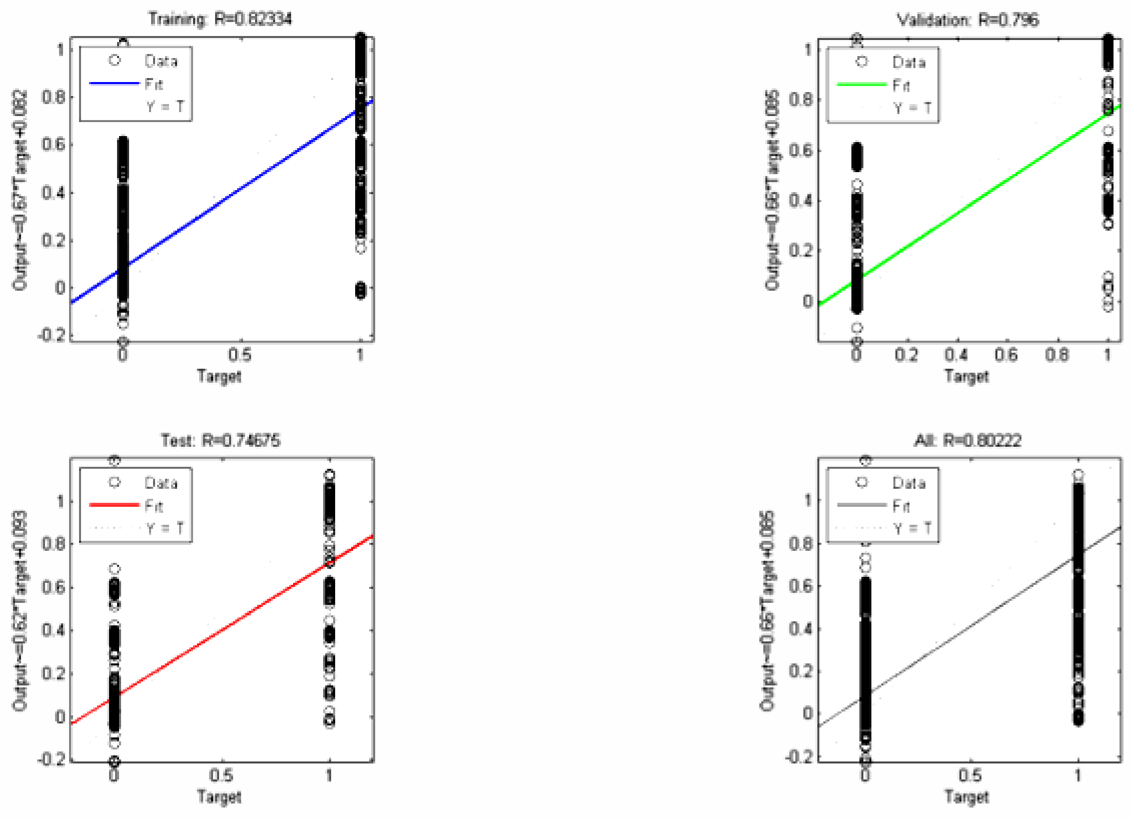
c) Resilient Backpropagation



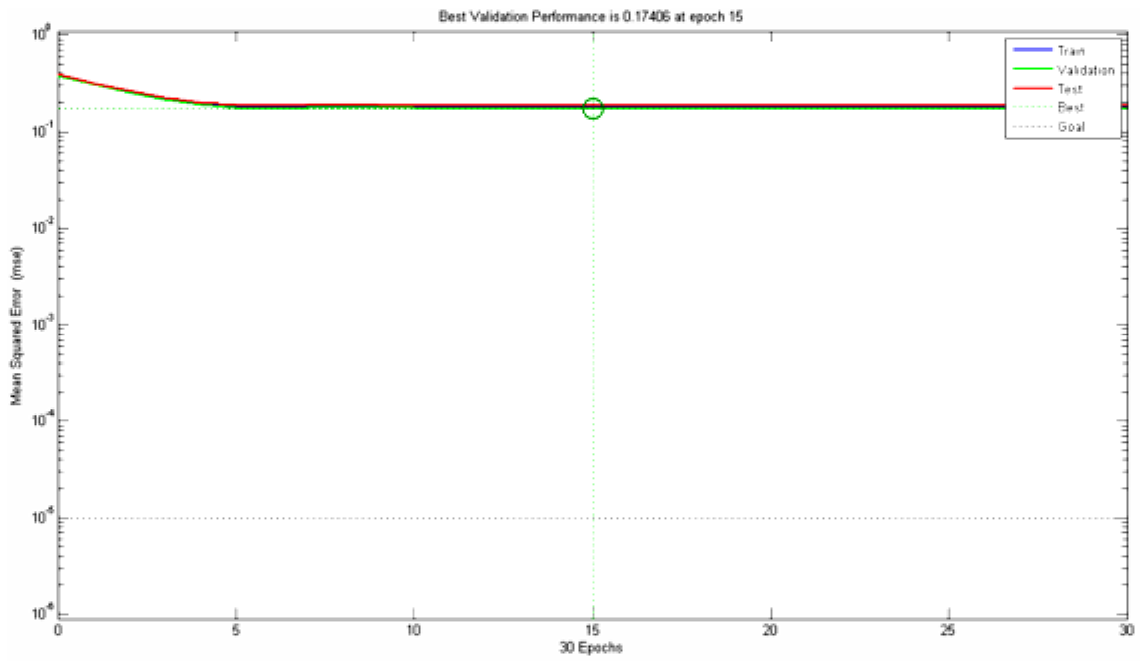


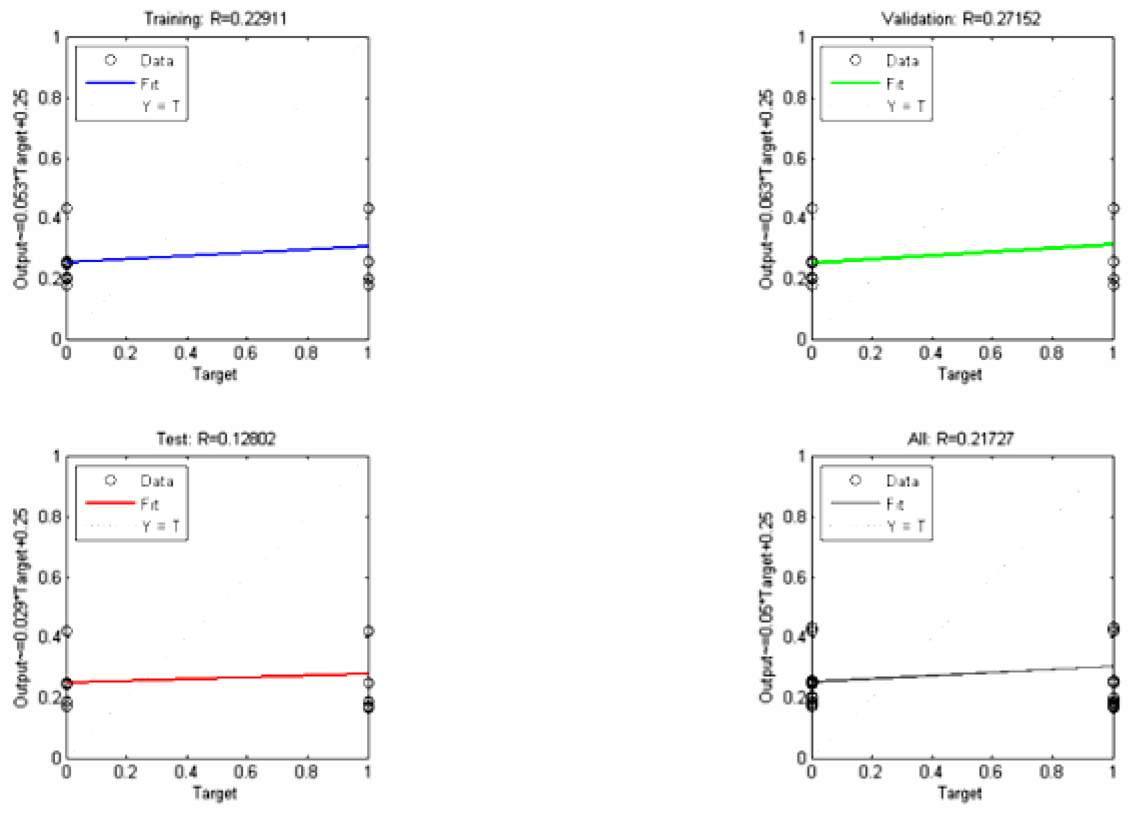
d) Scaled Conjugate Gradient



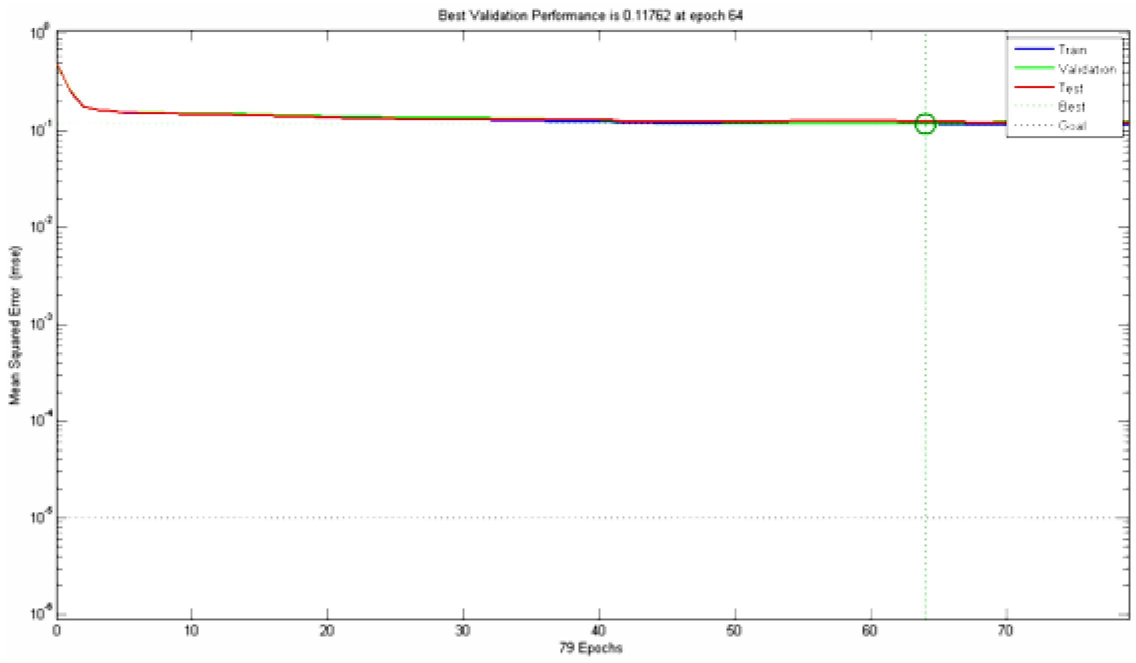


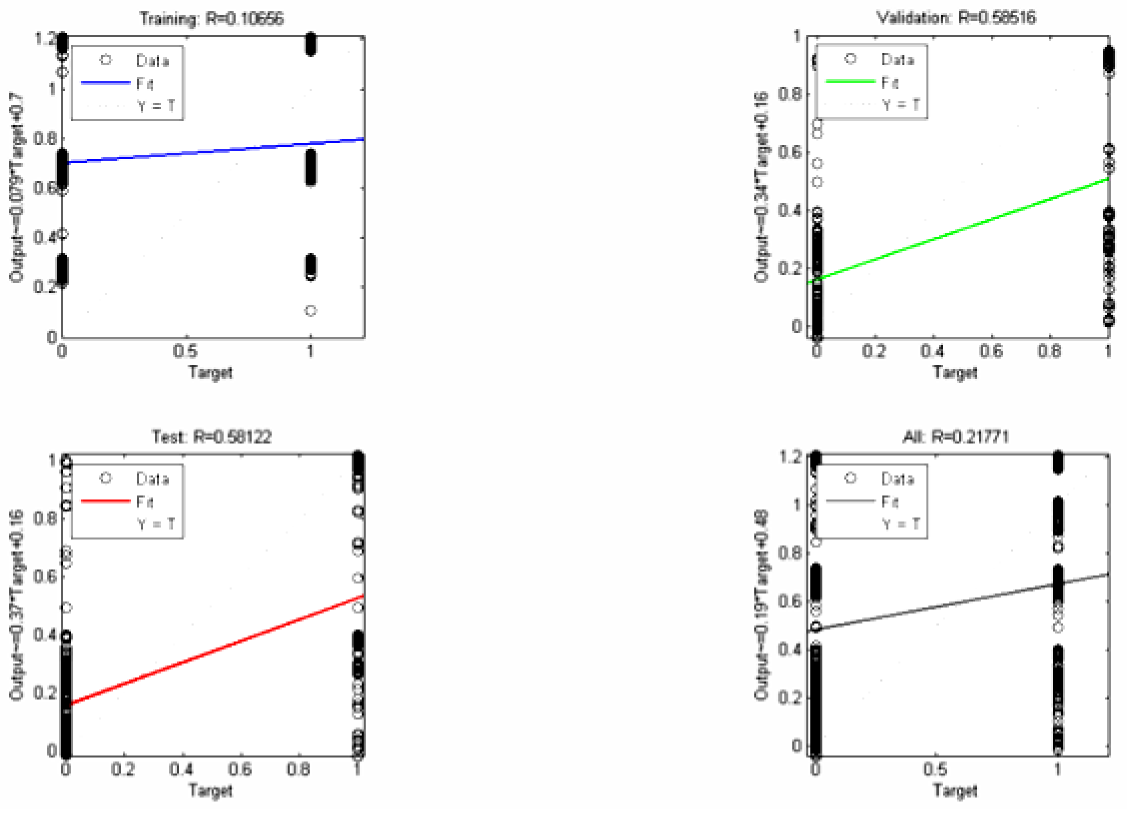
e) Variable Learning Rate Backpropagation





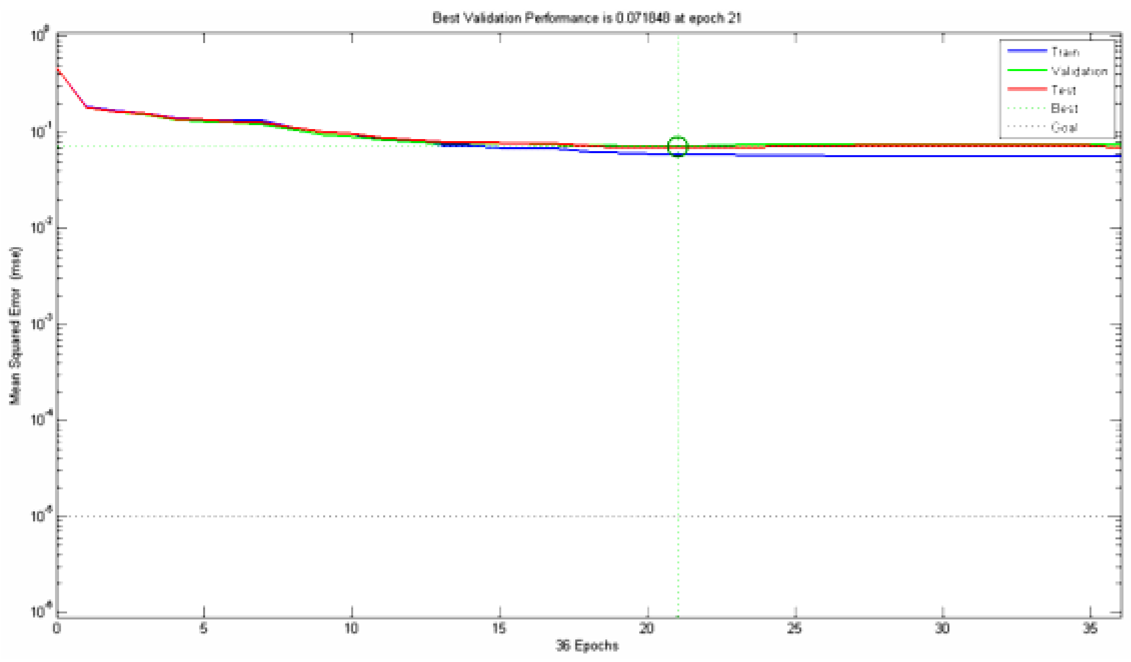
f) BFGS quasi–Newton Backpropagation

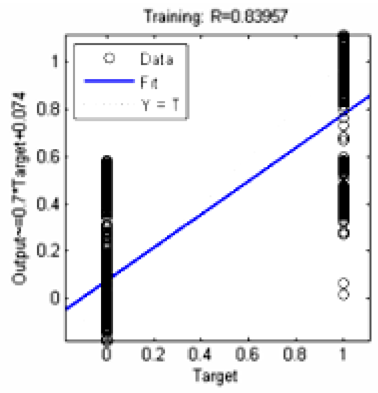


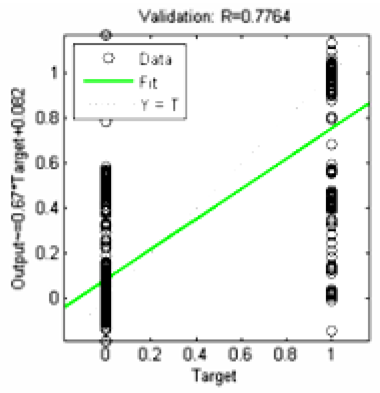


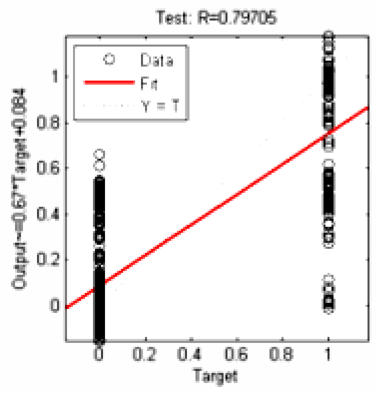
5) [8 3 2]

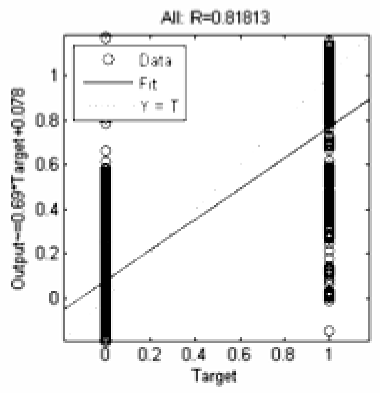
a) Levenberg–Marquardt



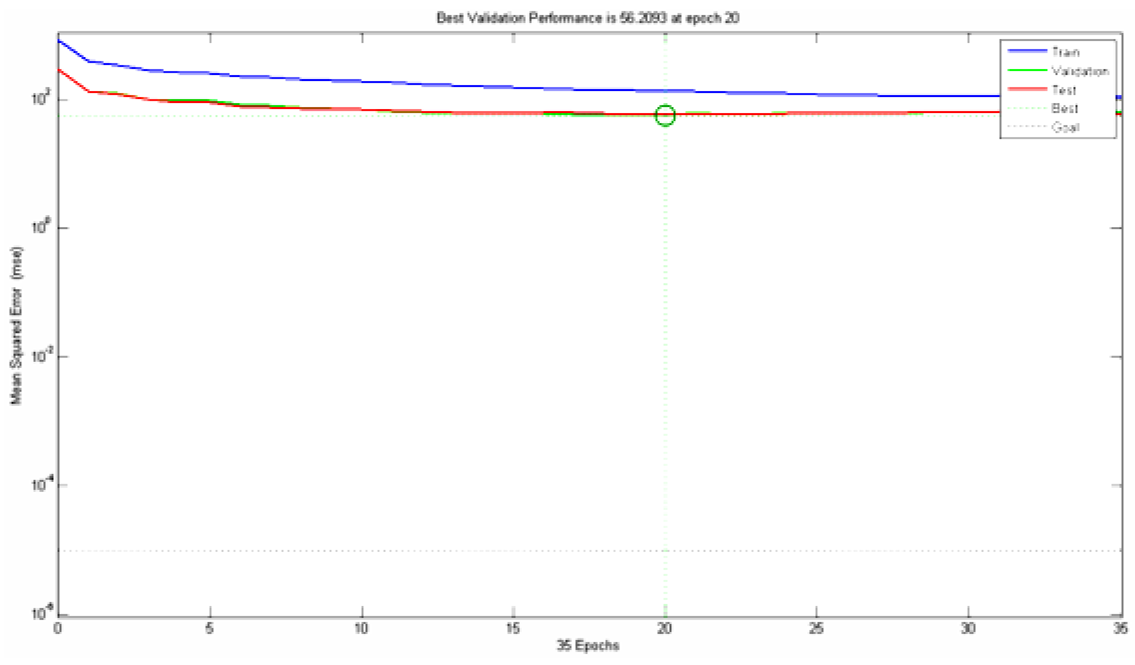


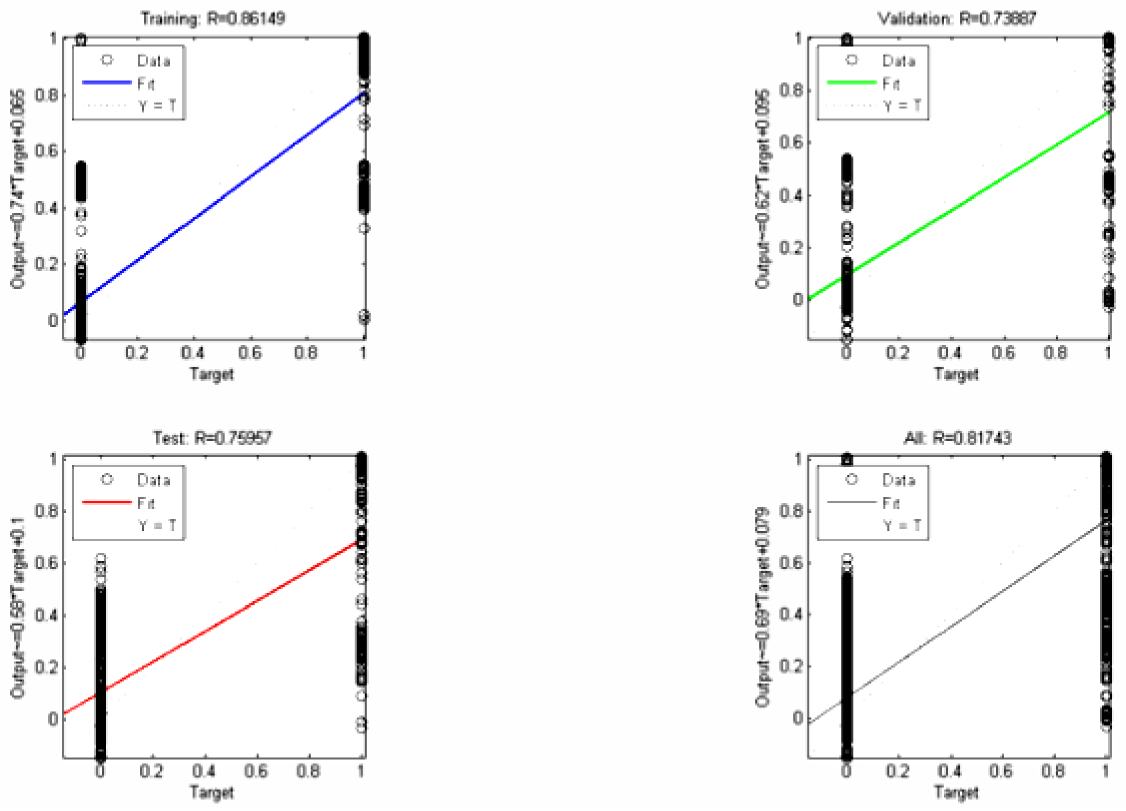




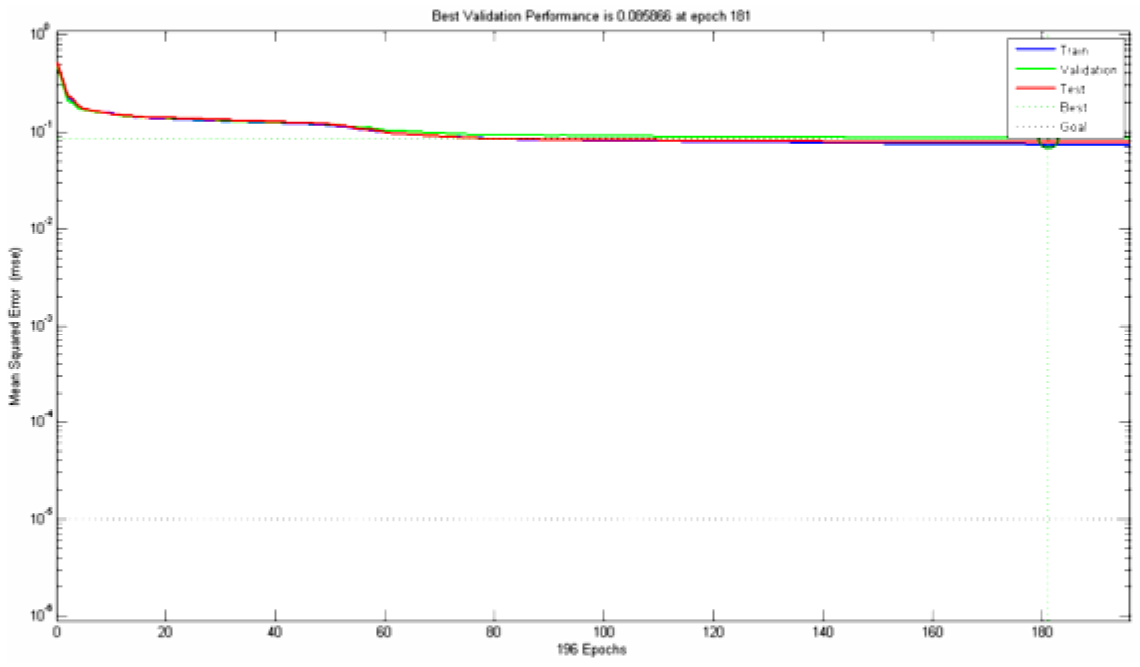


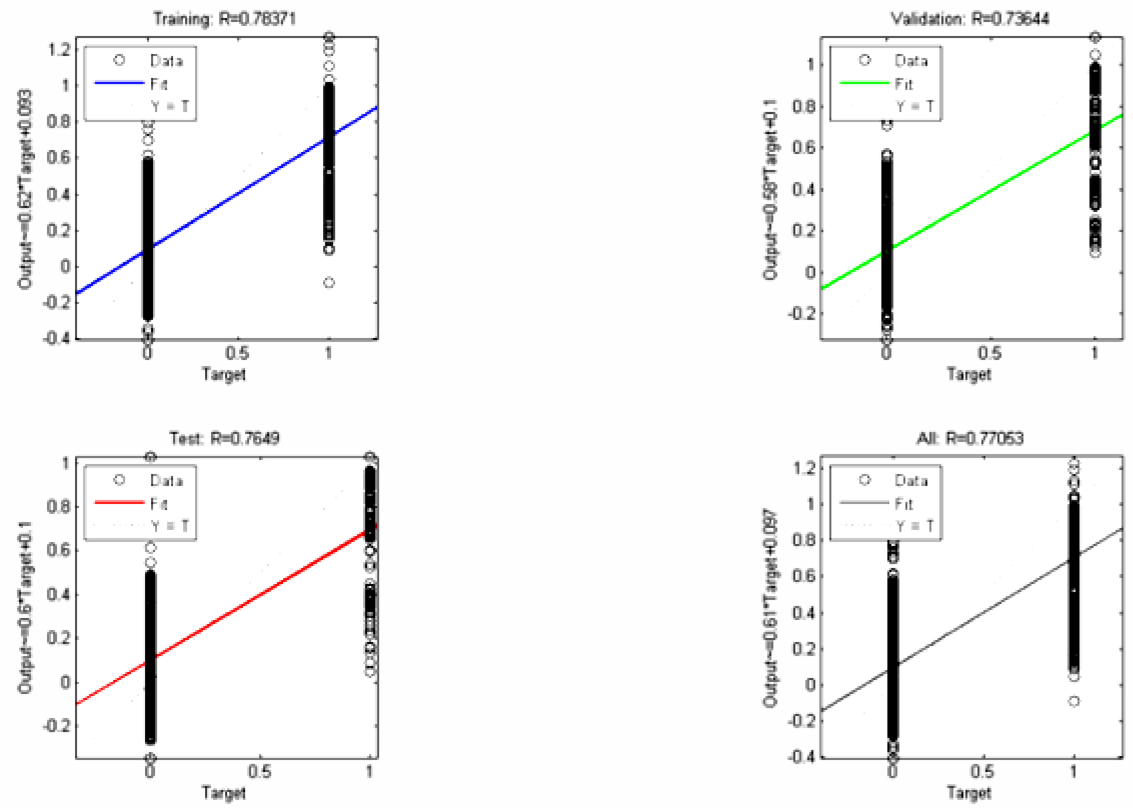
b) Bayesian Regularisation



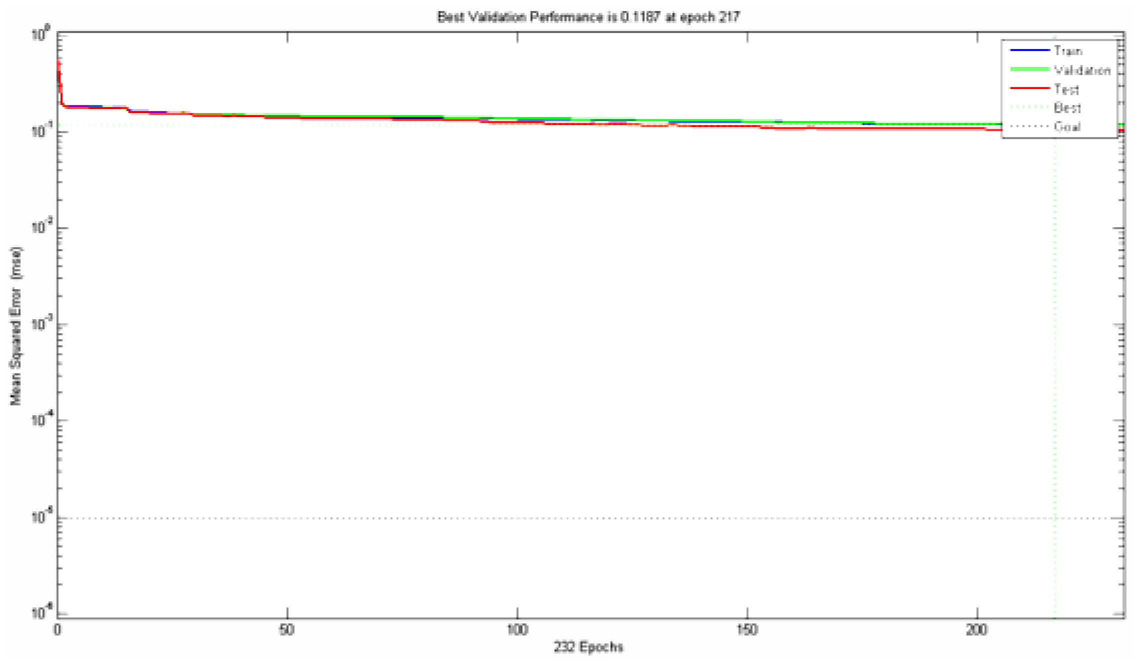


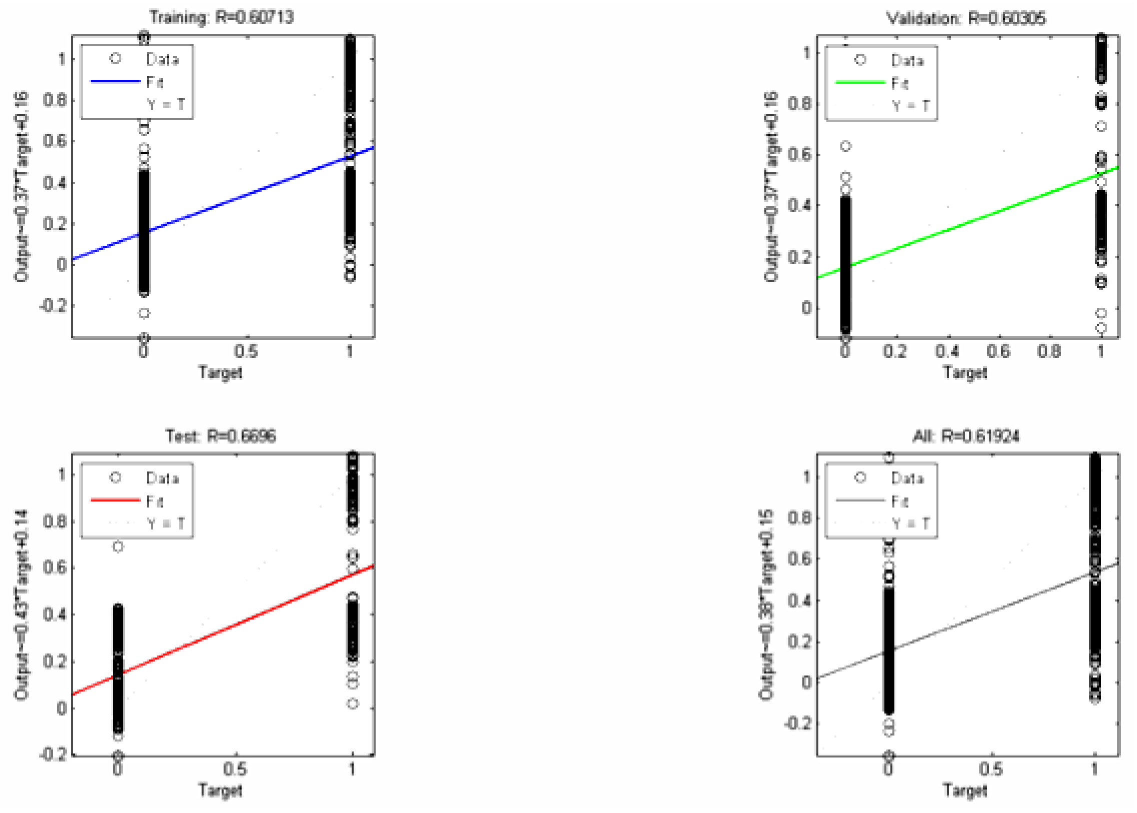
c) Resilient Backpropagation



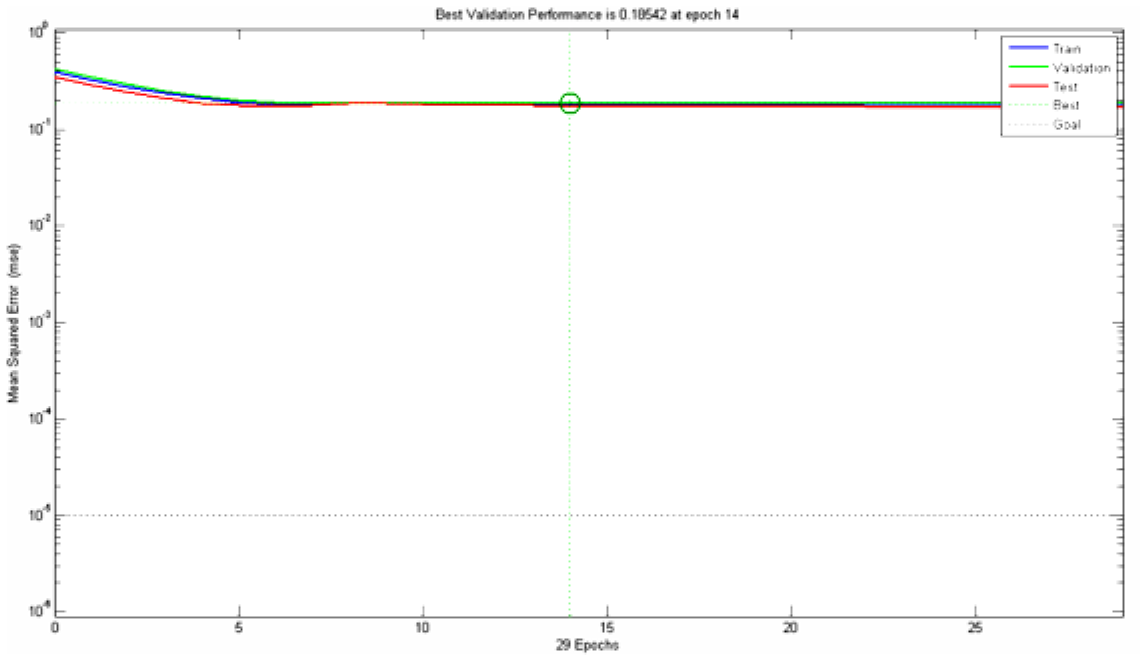


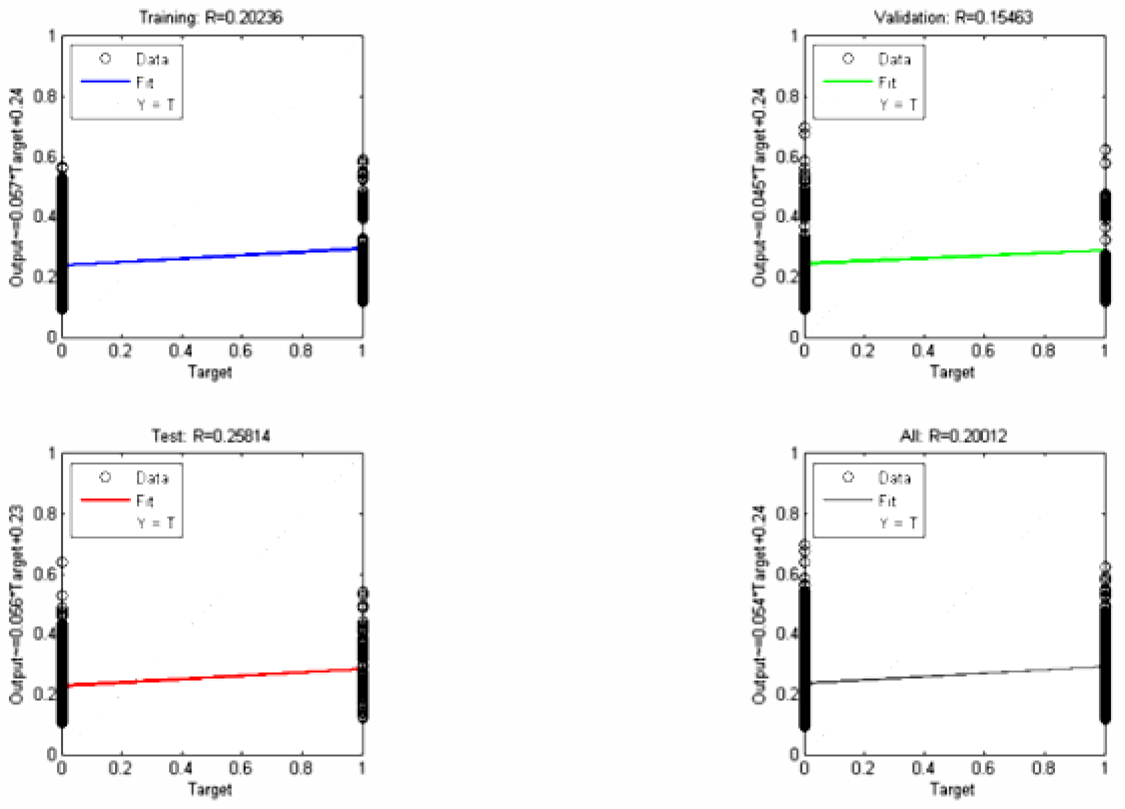
d) Scaled Conjugate Gradient



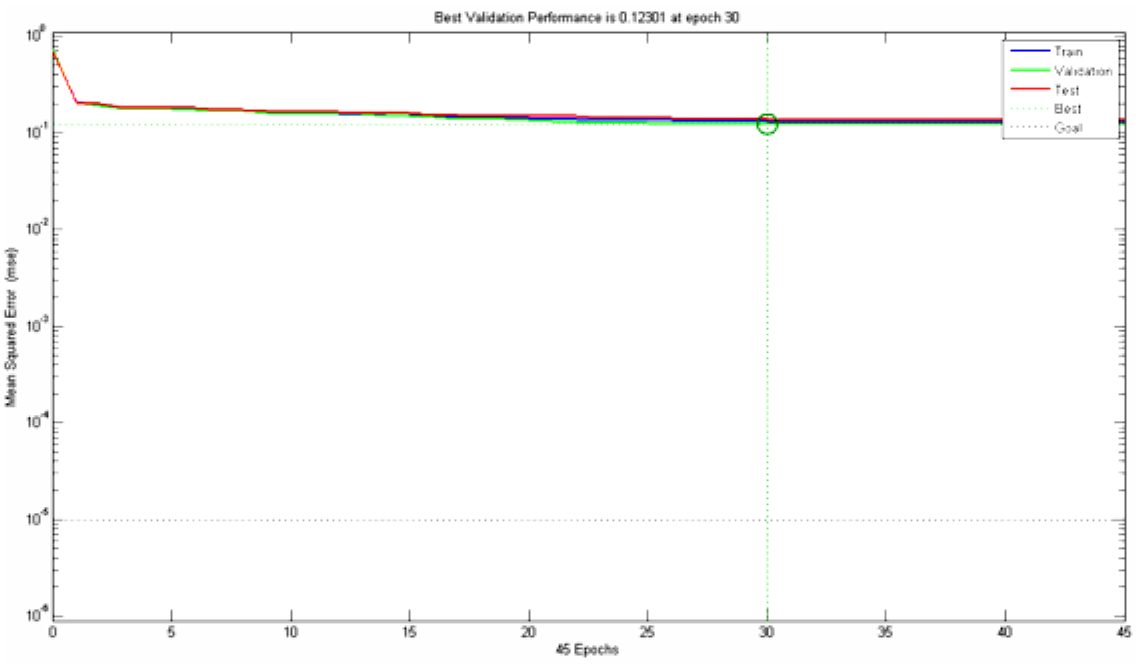


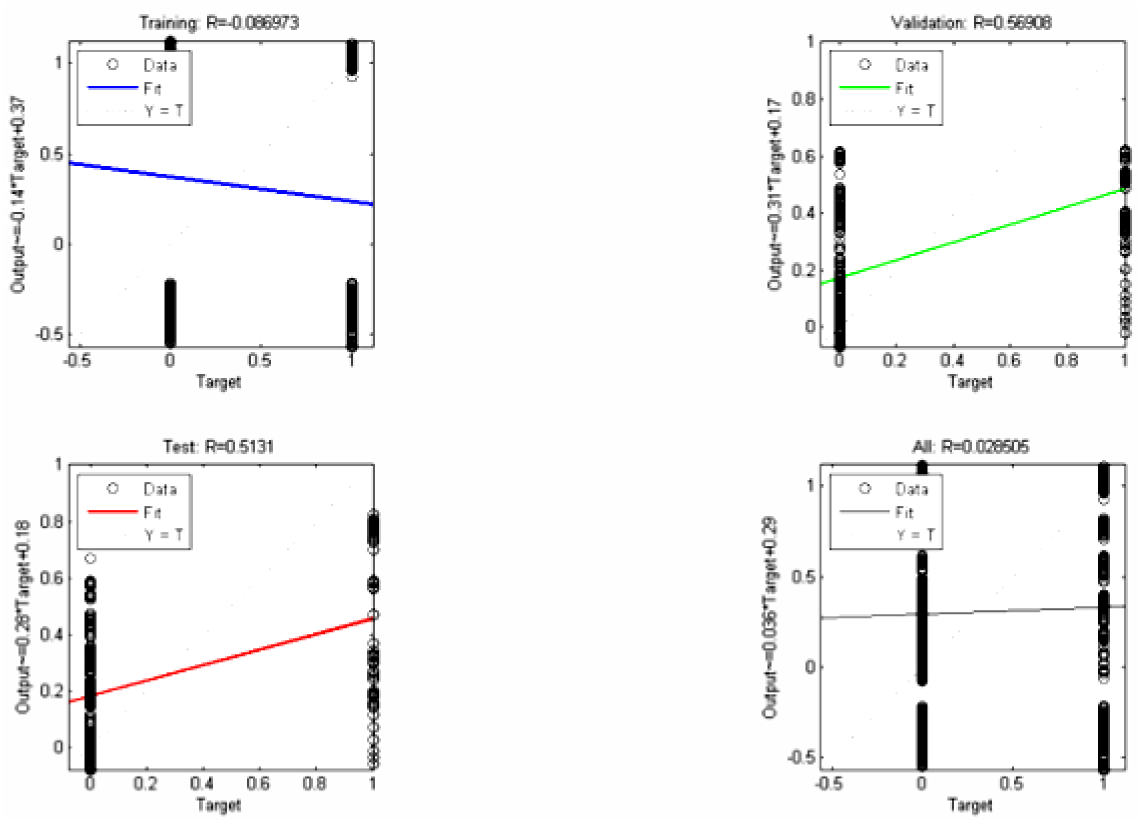
e) Variable Learning Rate Backpropagation





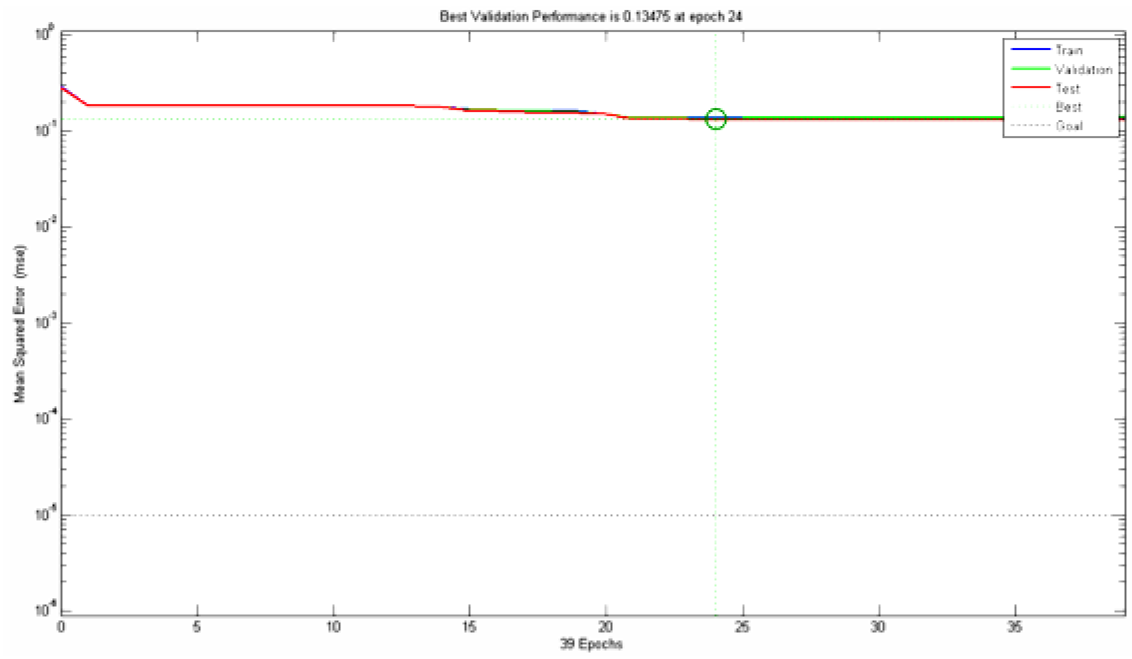
f) BFGS quasi–Newton Backpropagation

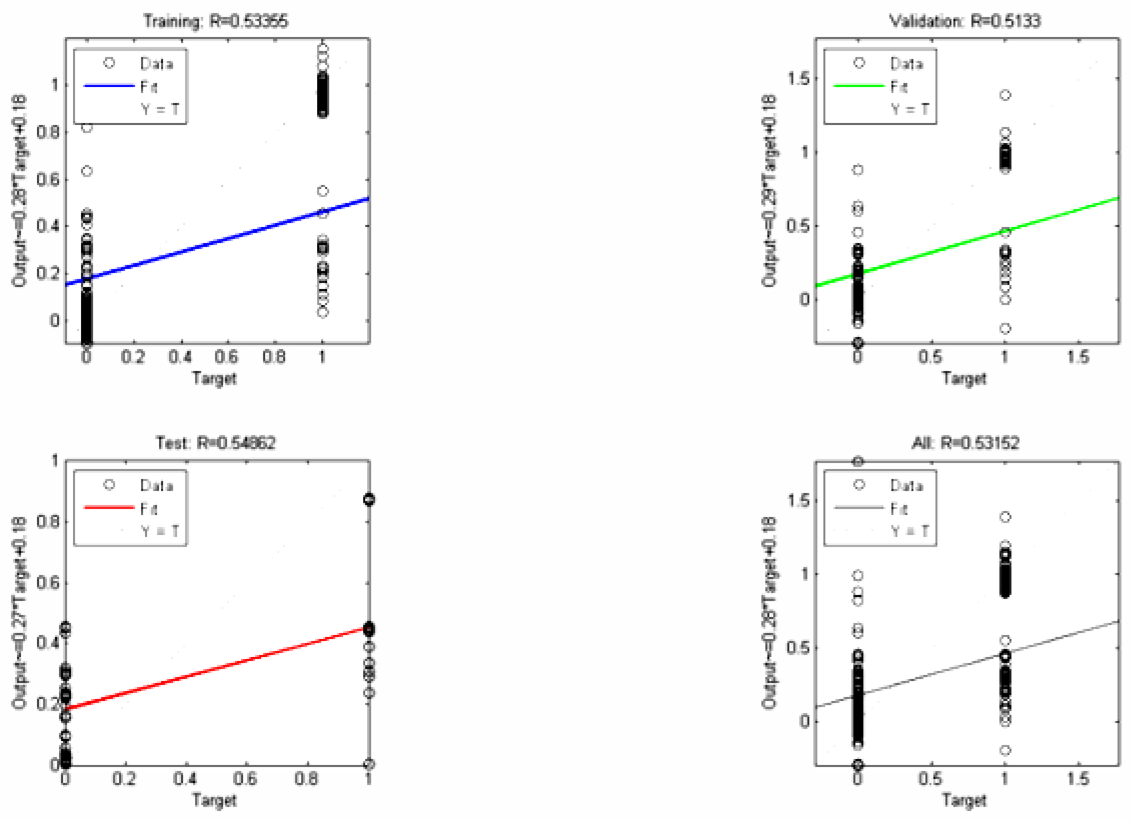




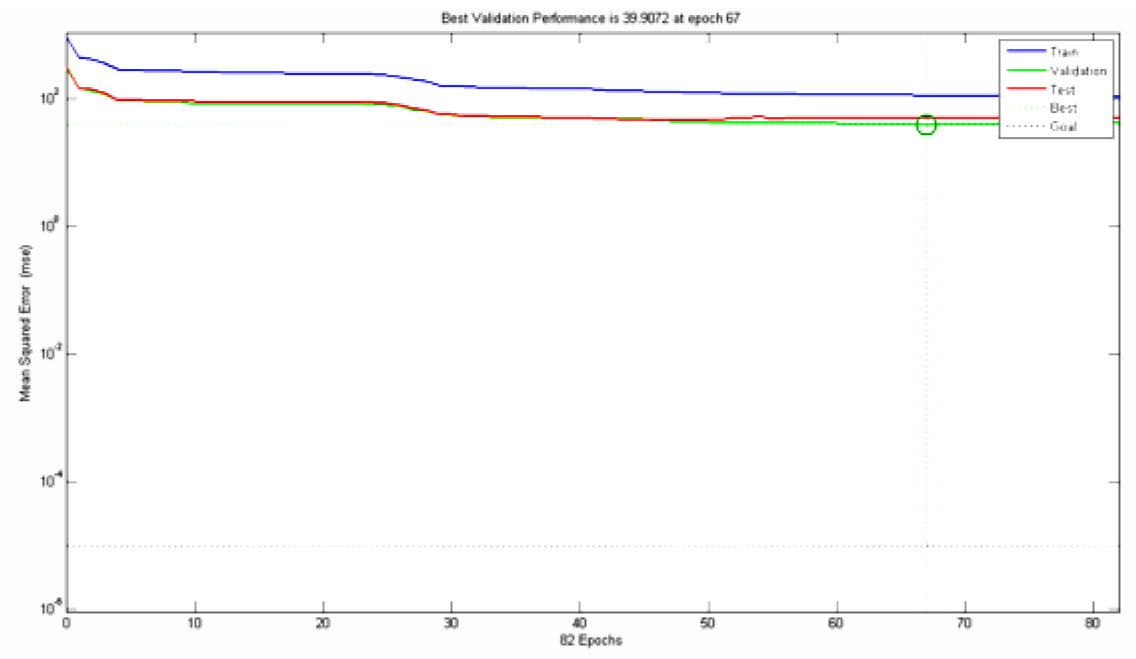
6) [8 4 2]

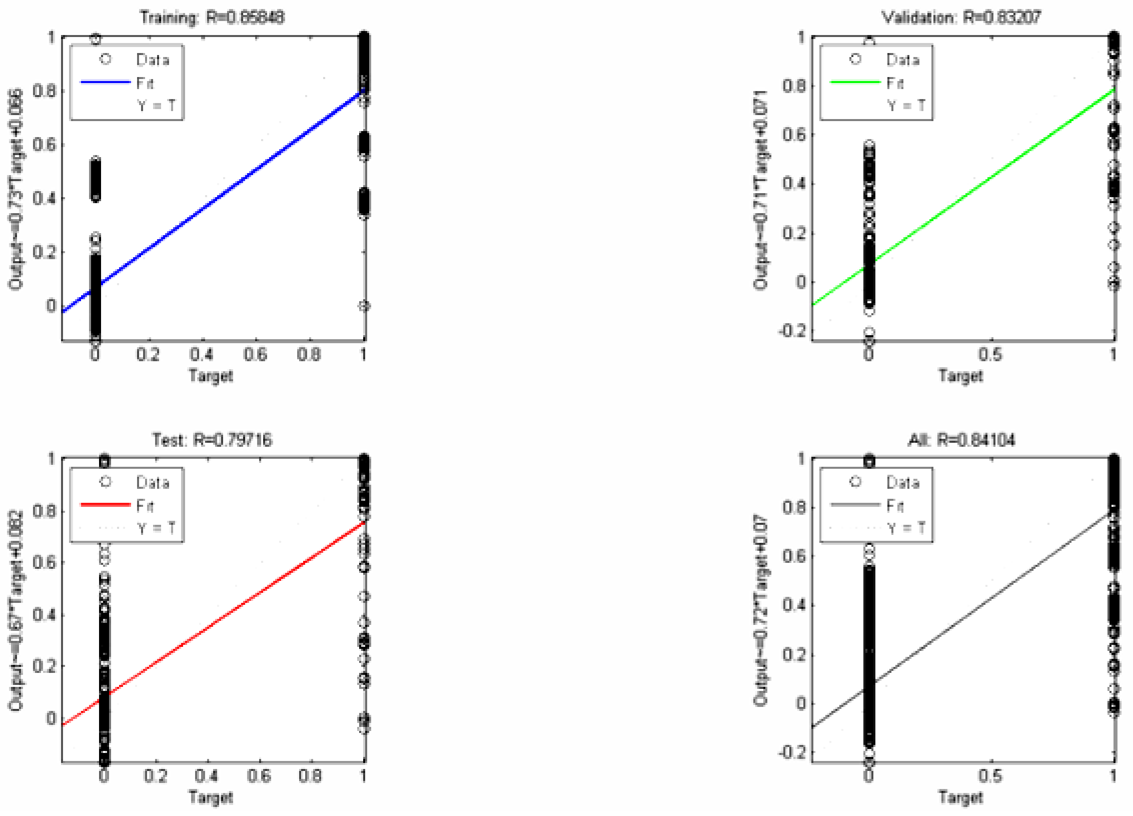
a) Levenberg–Marquardt



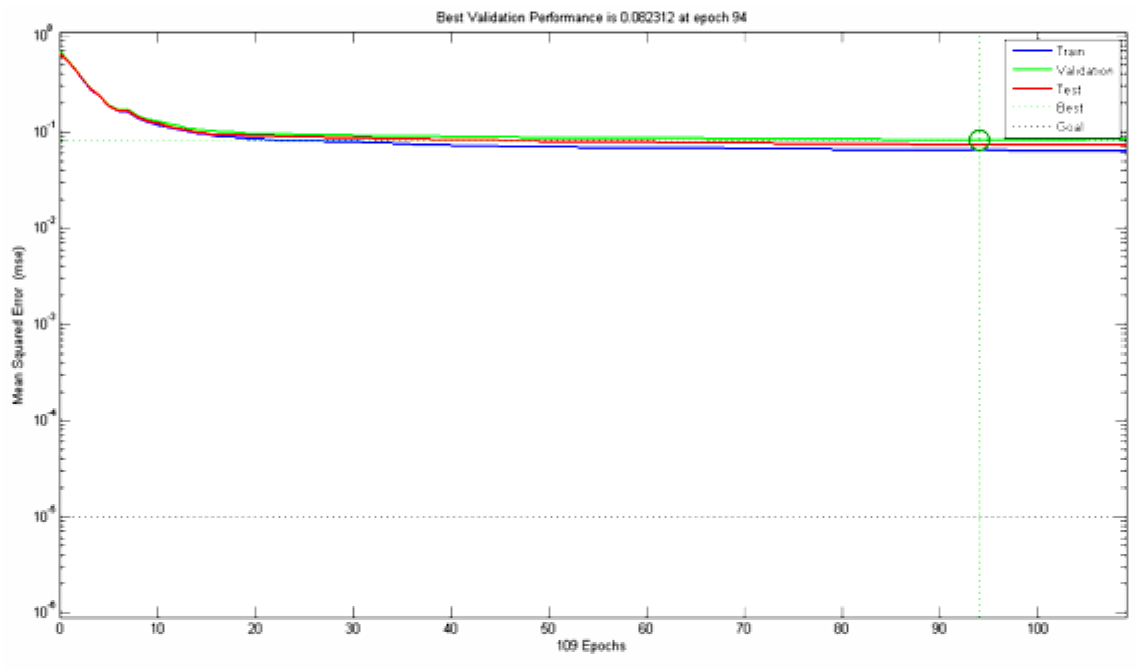


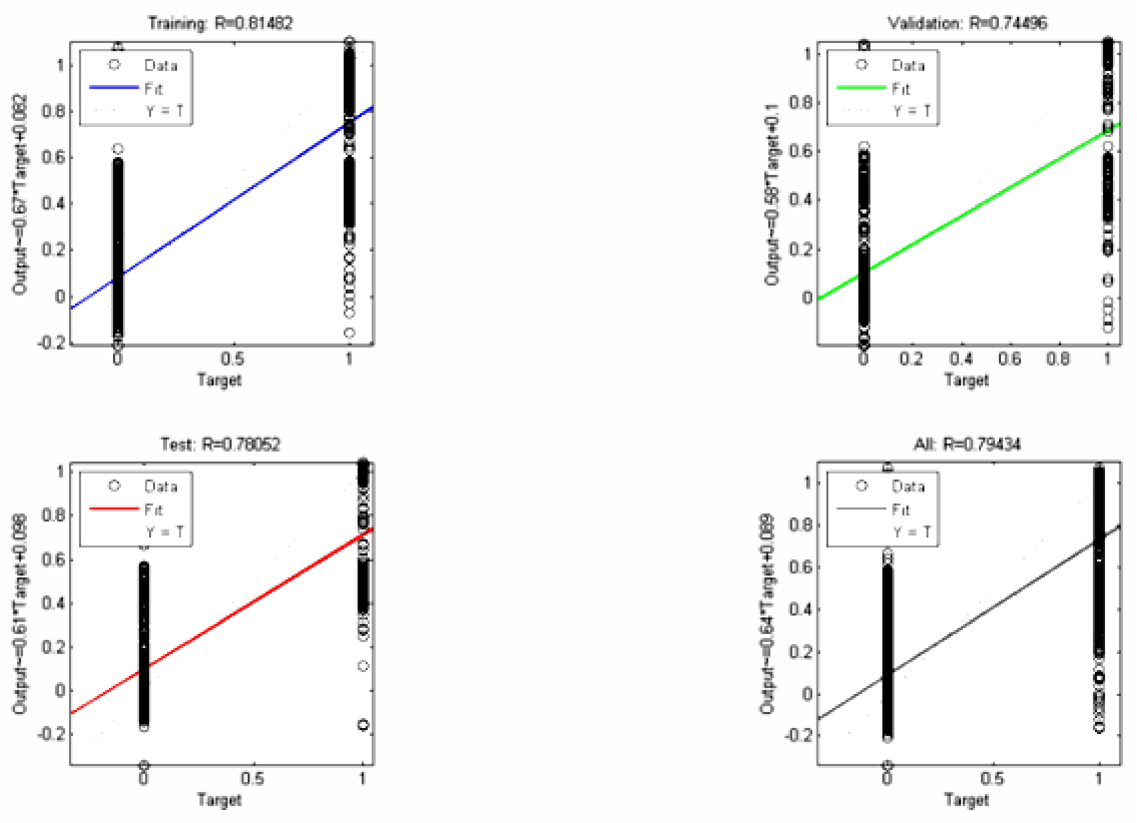
b) Bayesian Regularisation



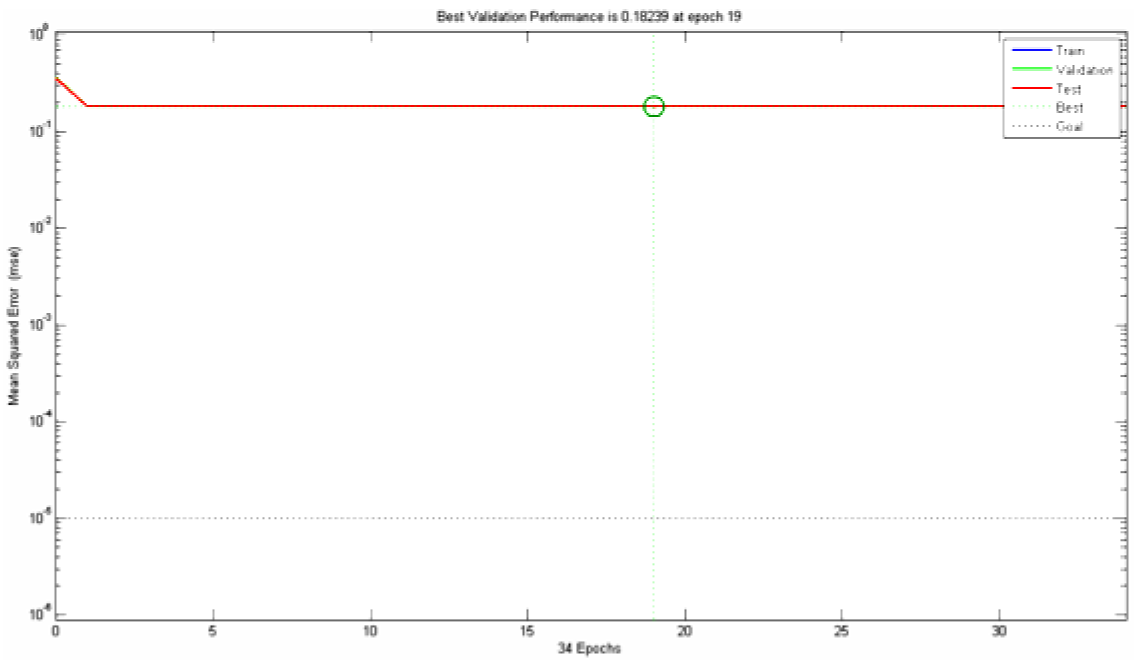


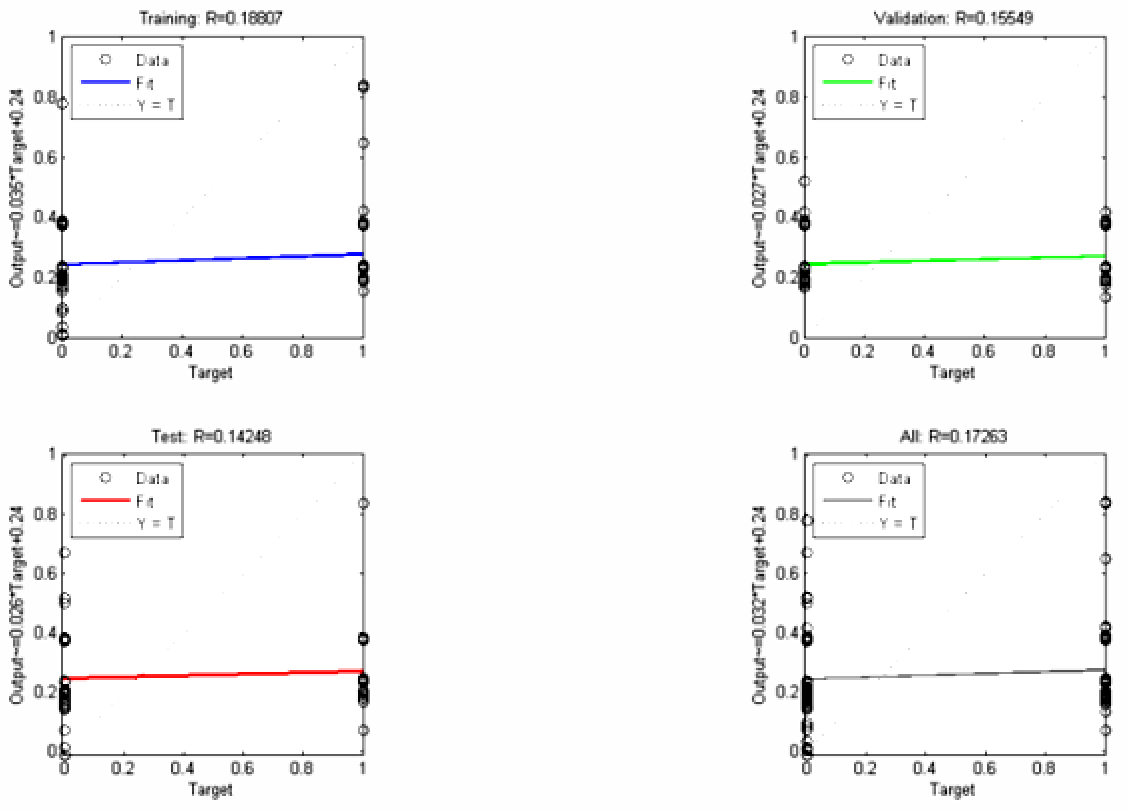
c) Resilient Backpropagation



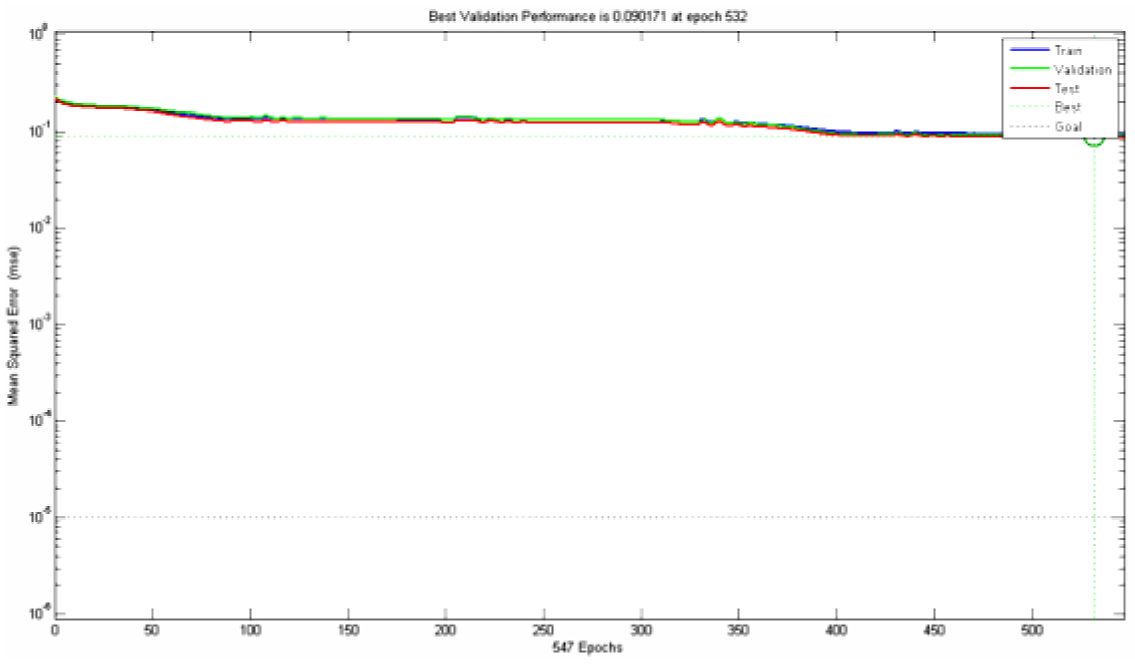


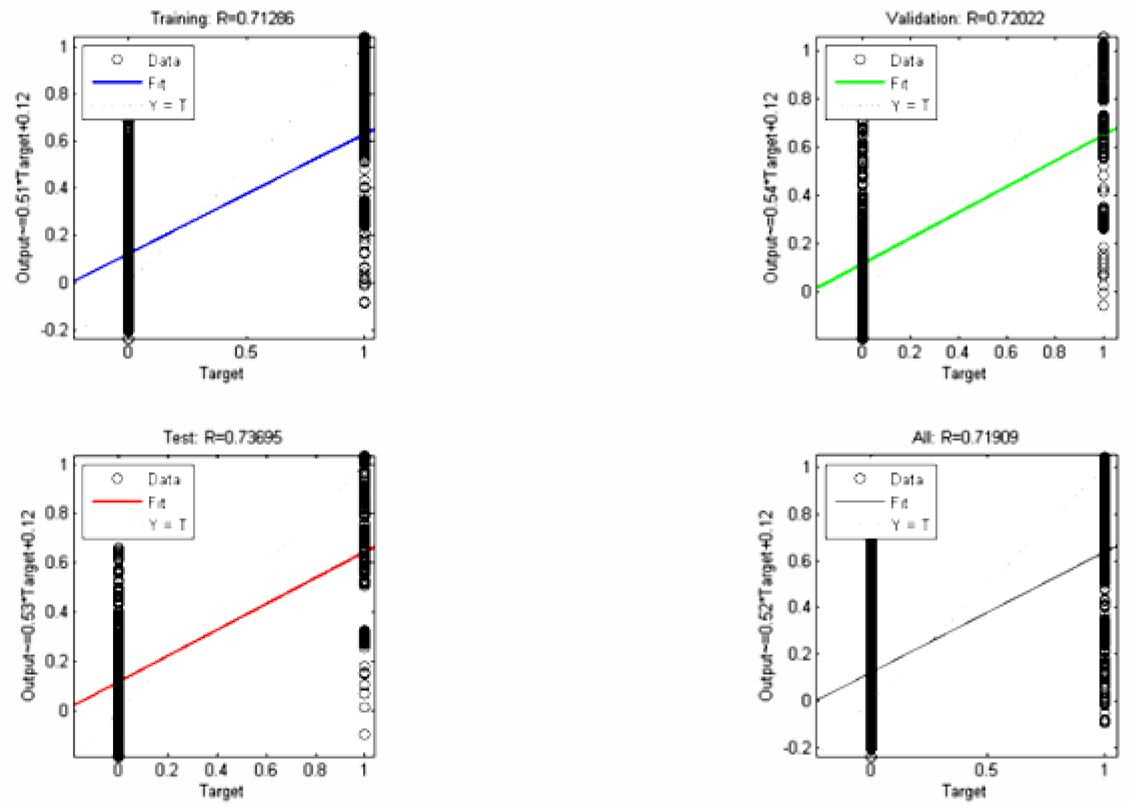
d) Scaled Conjugate Gradient



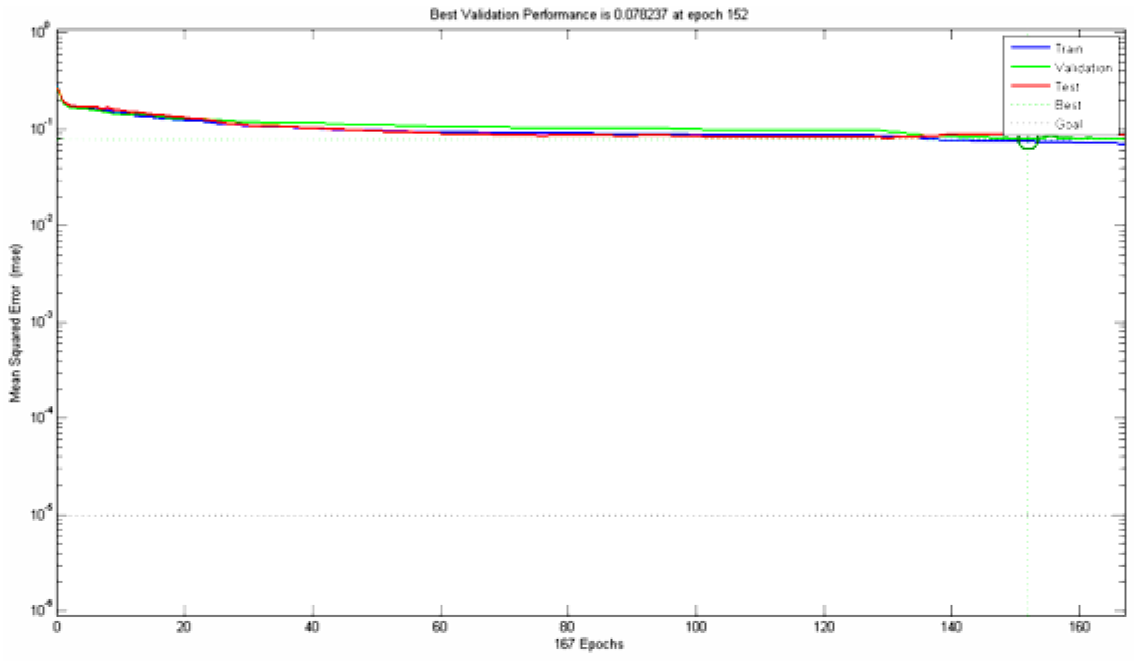


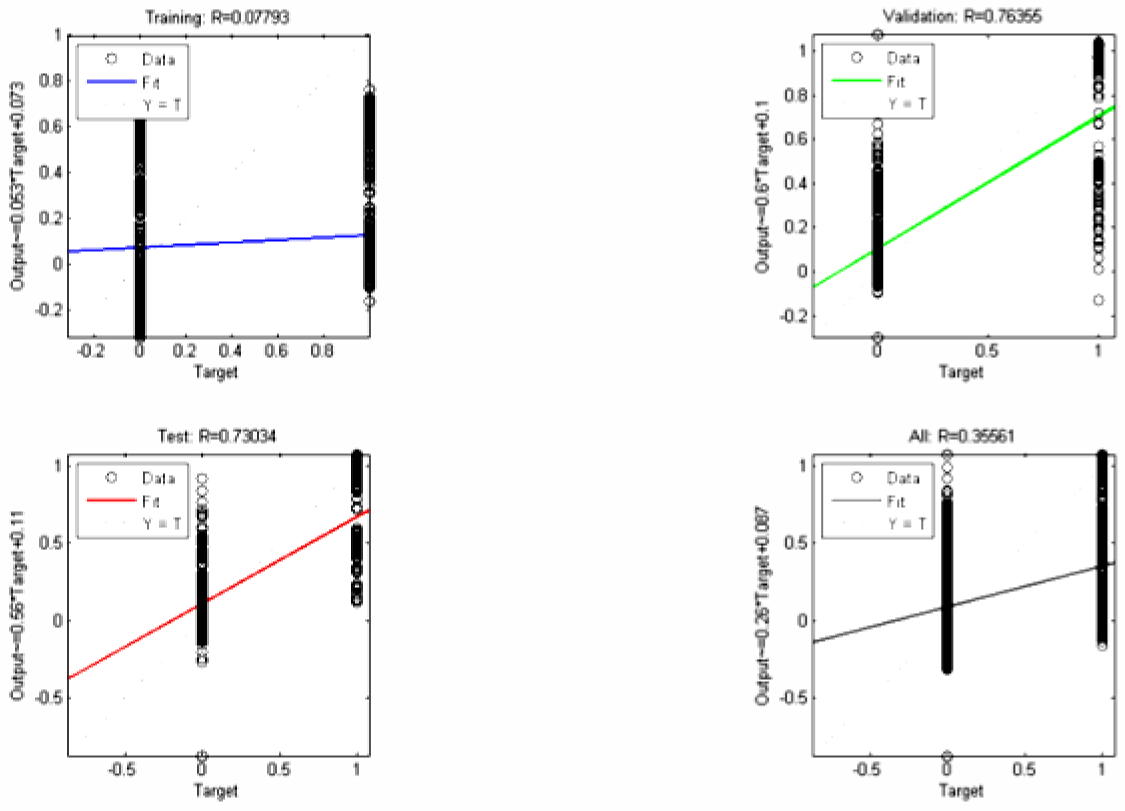
e) Variable Learning Rate Backpropagation





f) BFGS quasi–Newton Backpropagation

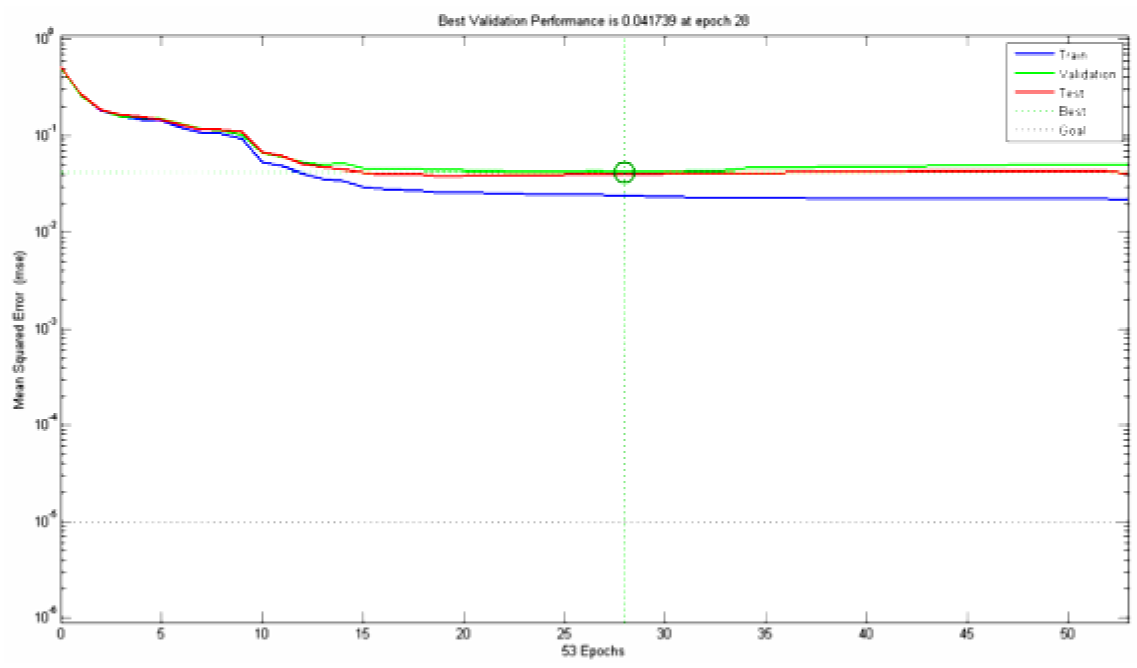


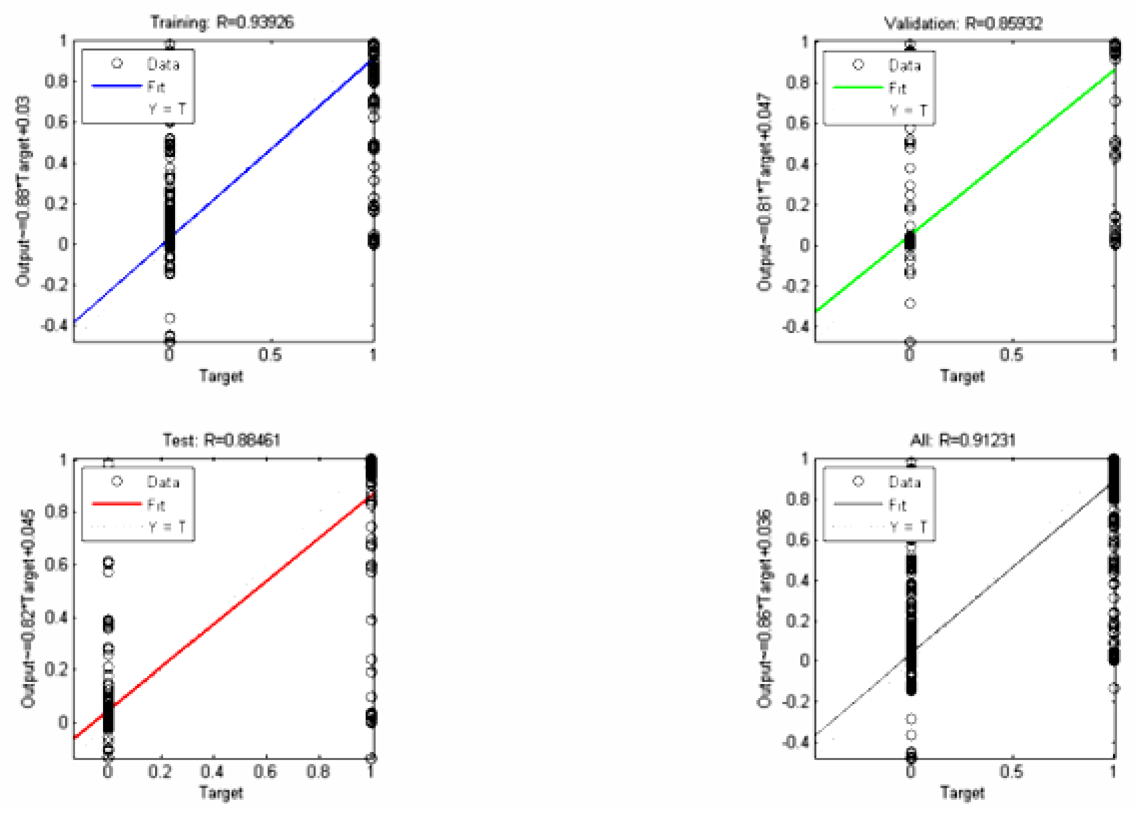


4) Four hidden layers

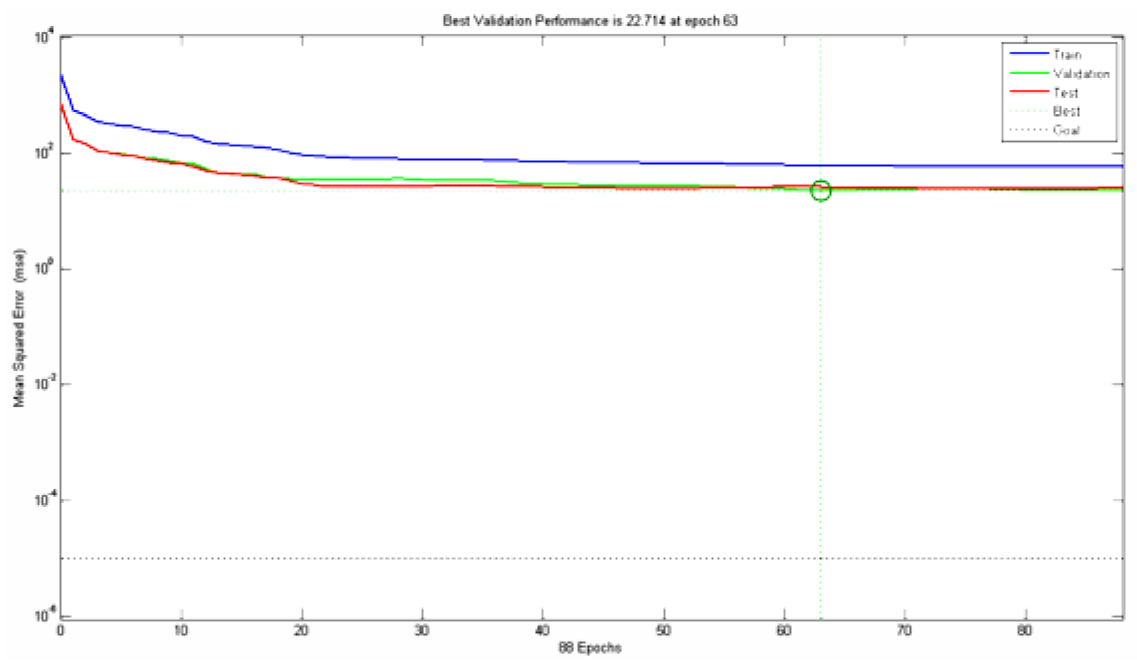
1) [3 3 3 3]

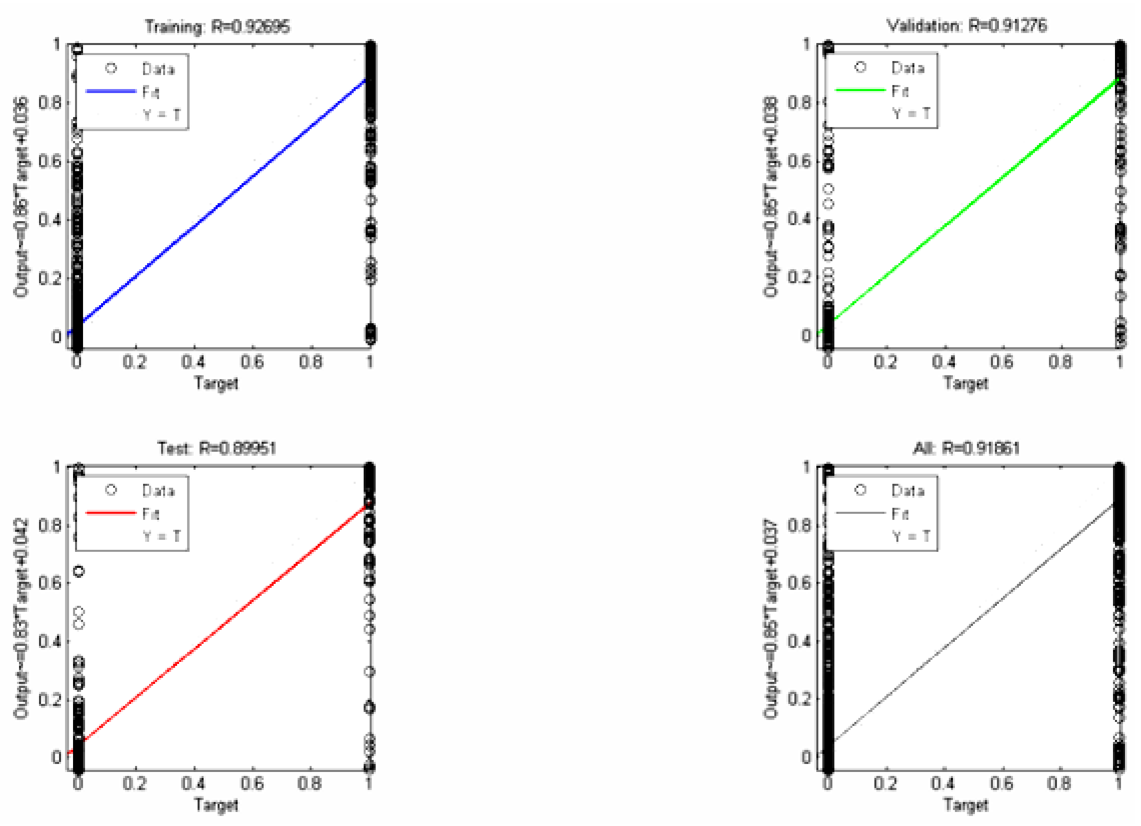
a) Levenberg–Marquardt



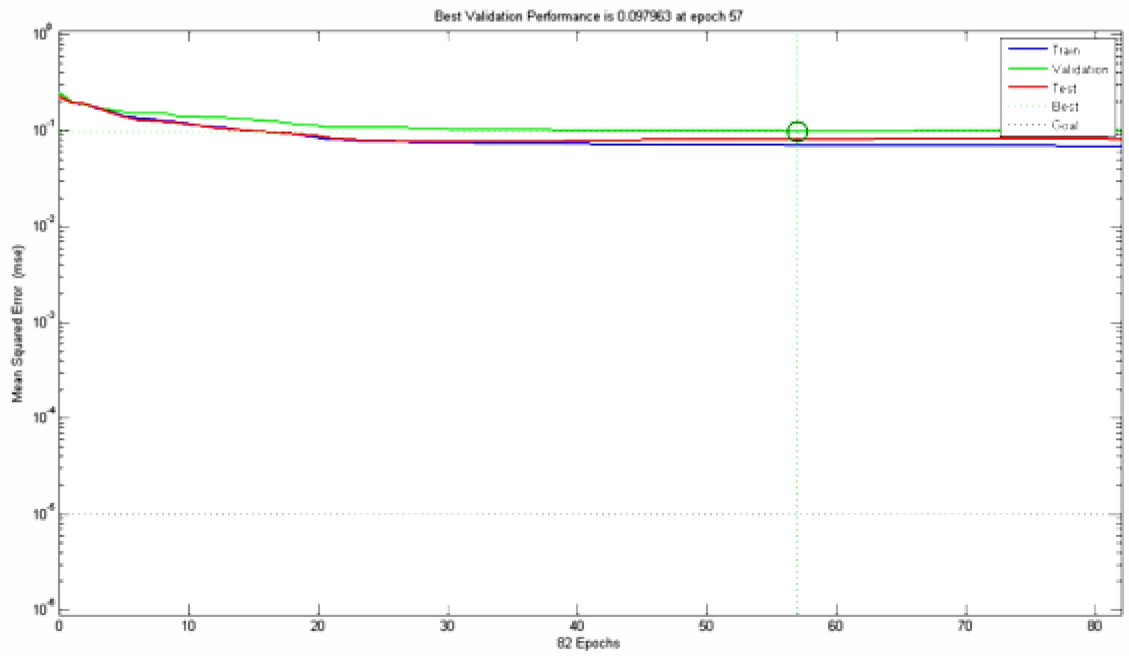


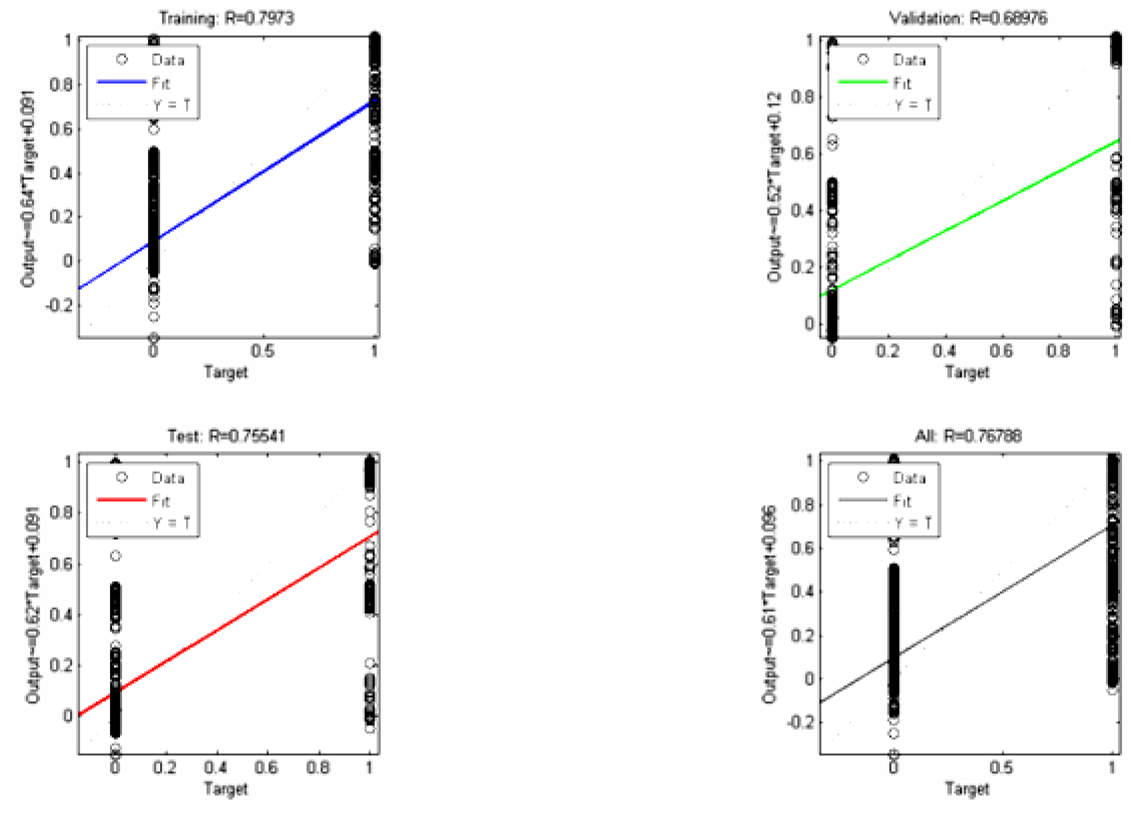
b) Bayesian Regularisation



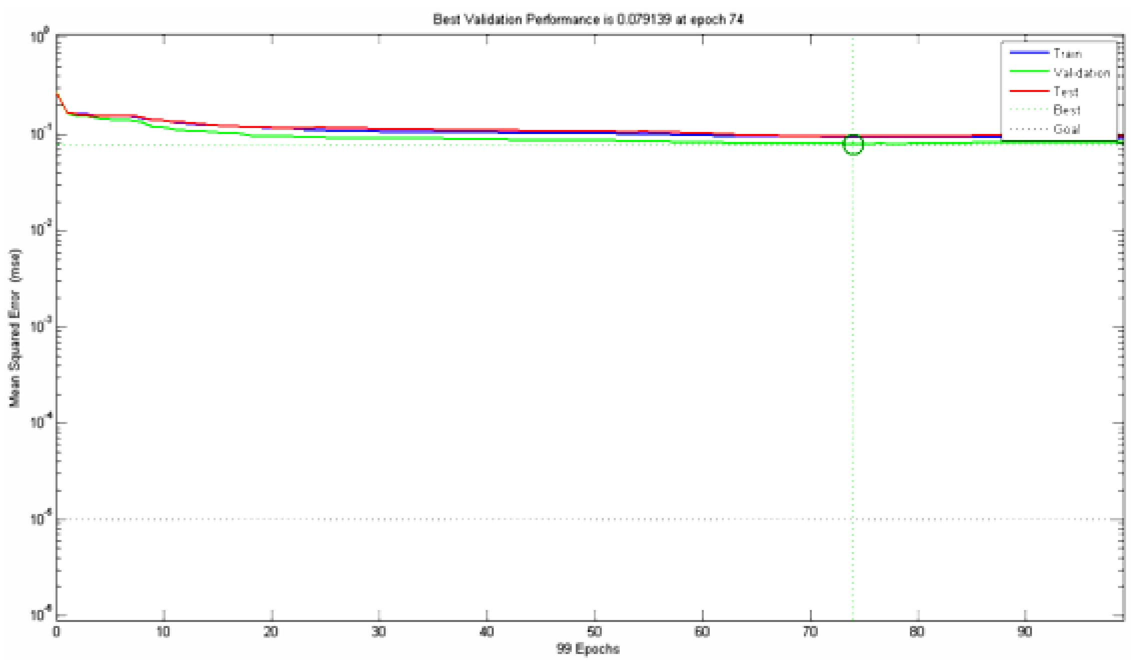


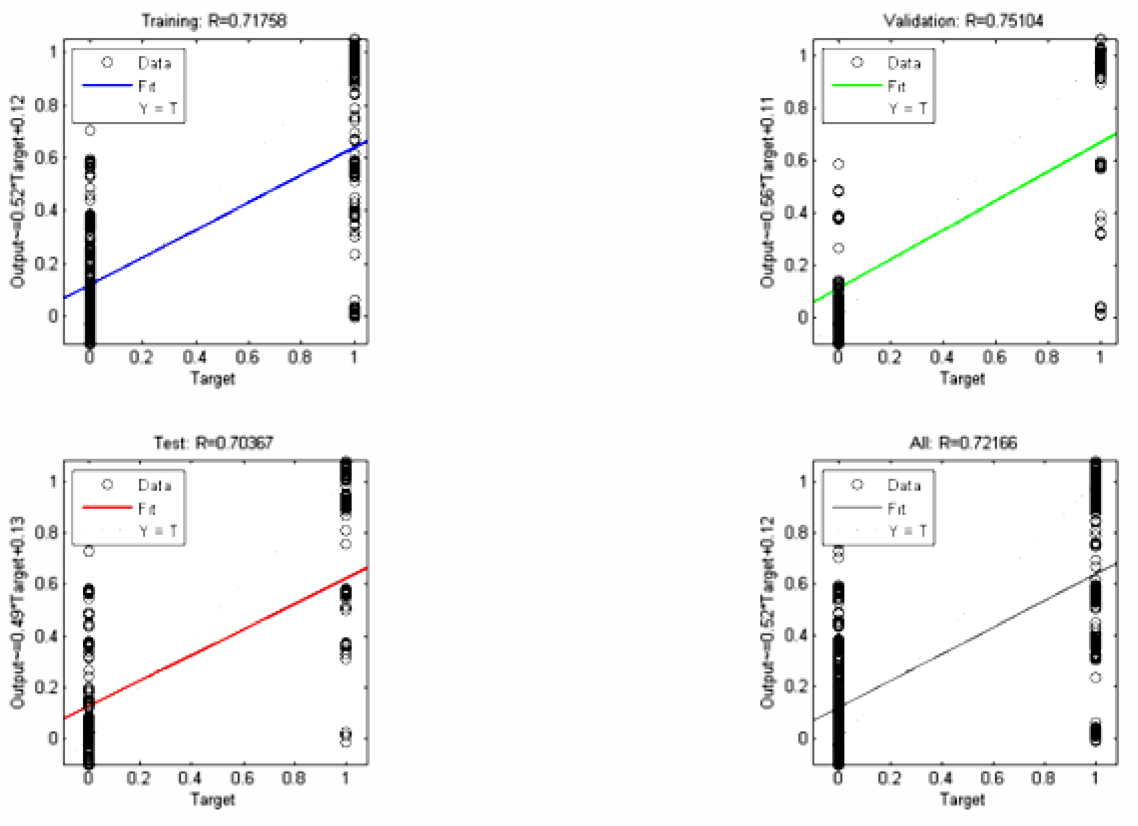
c) Resilient Backpropagation



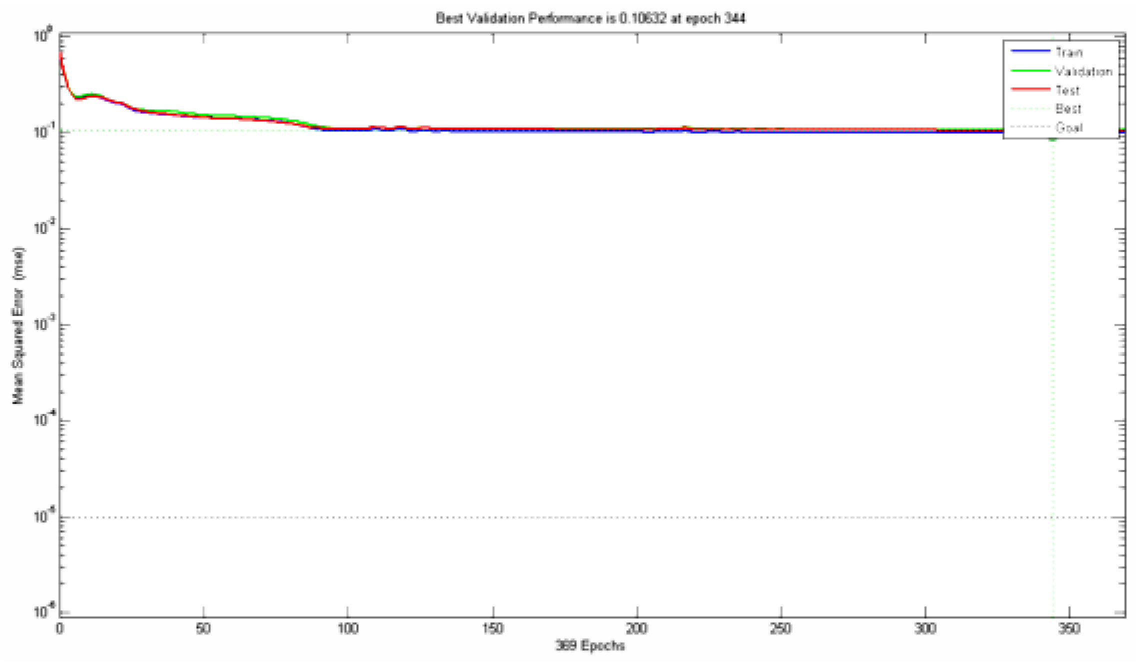


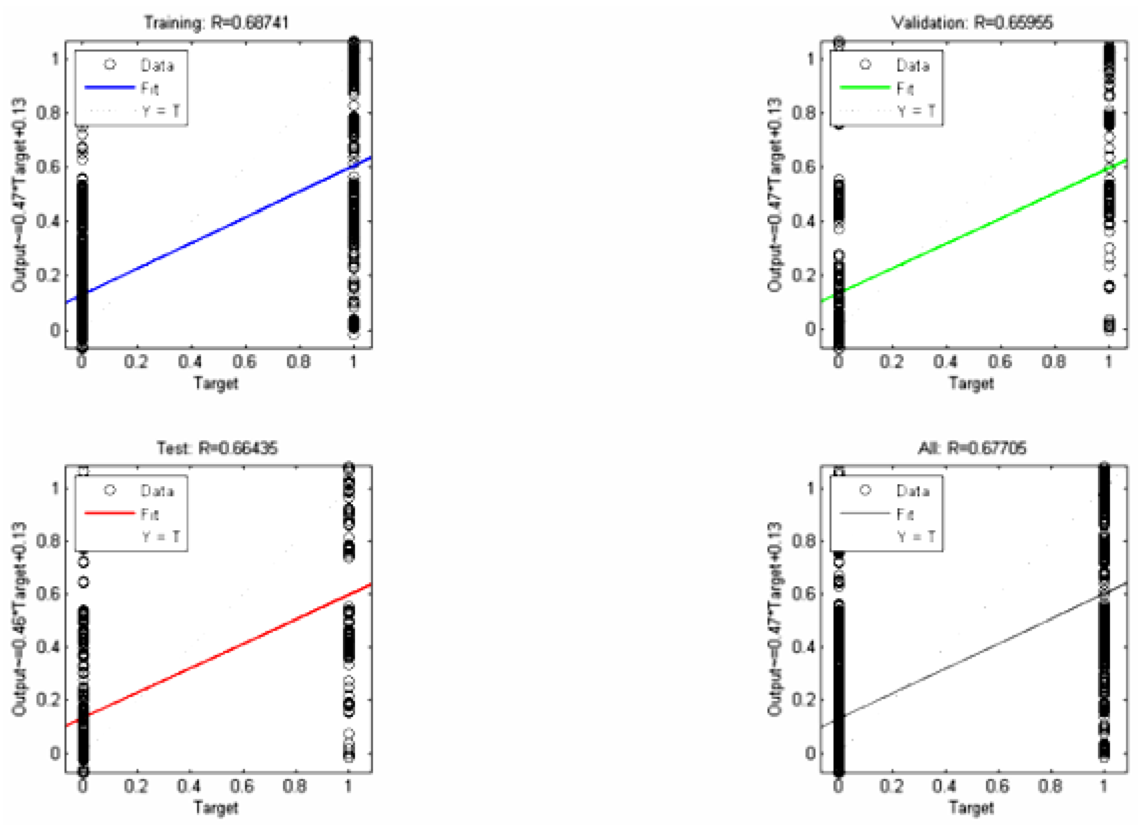
d) Scaled Conjugate Gradient



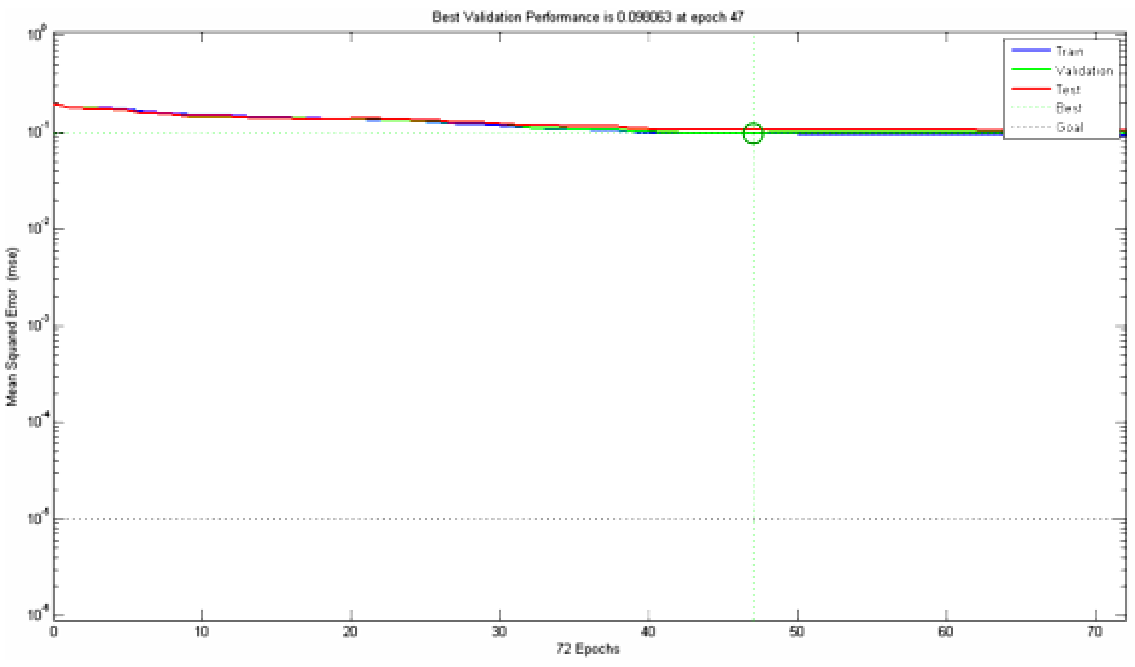


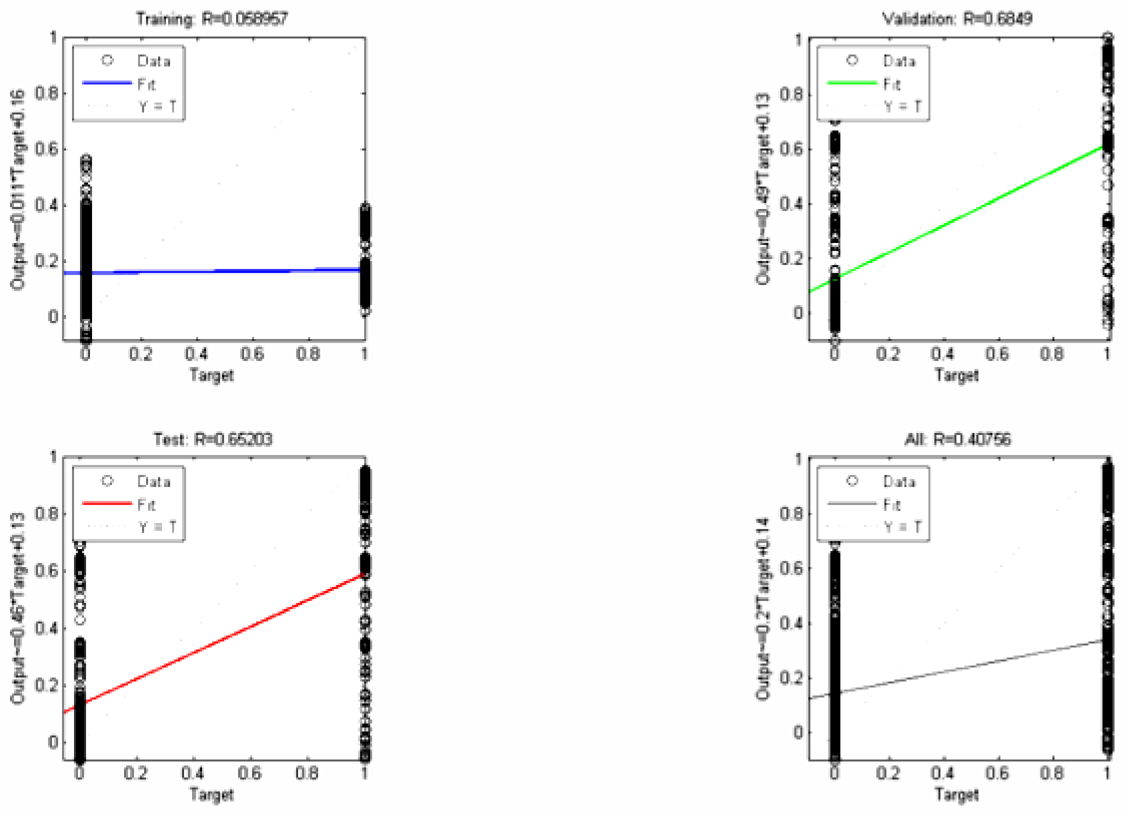
e) Variable Learning Rate Backpropagation





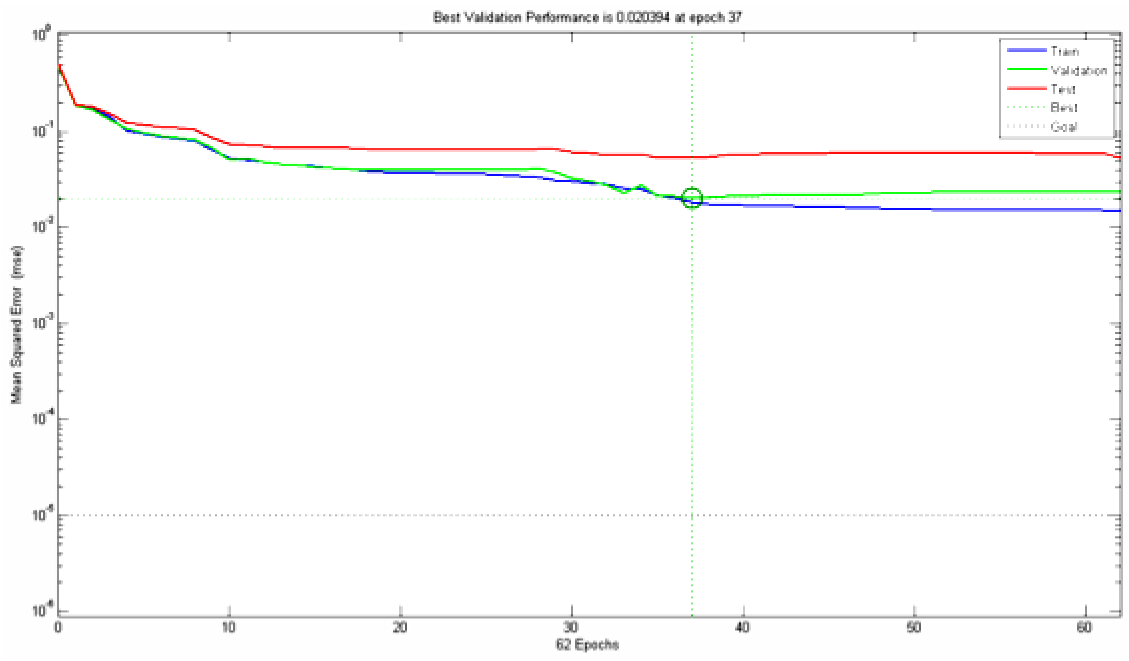
f) BFGS quasi–Newton Backpropagation

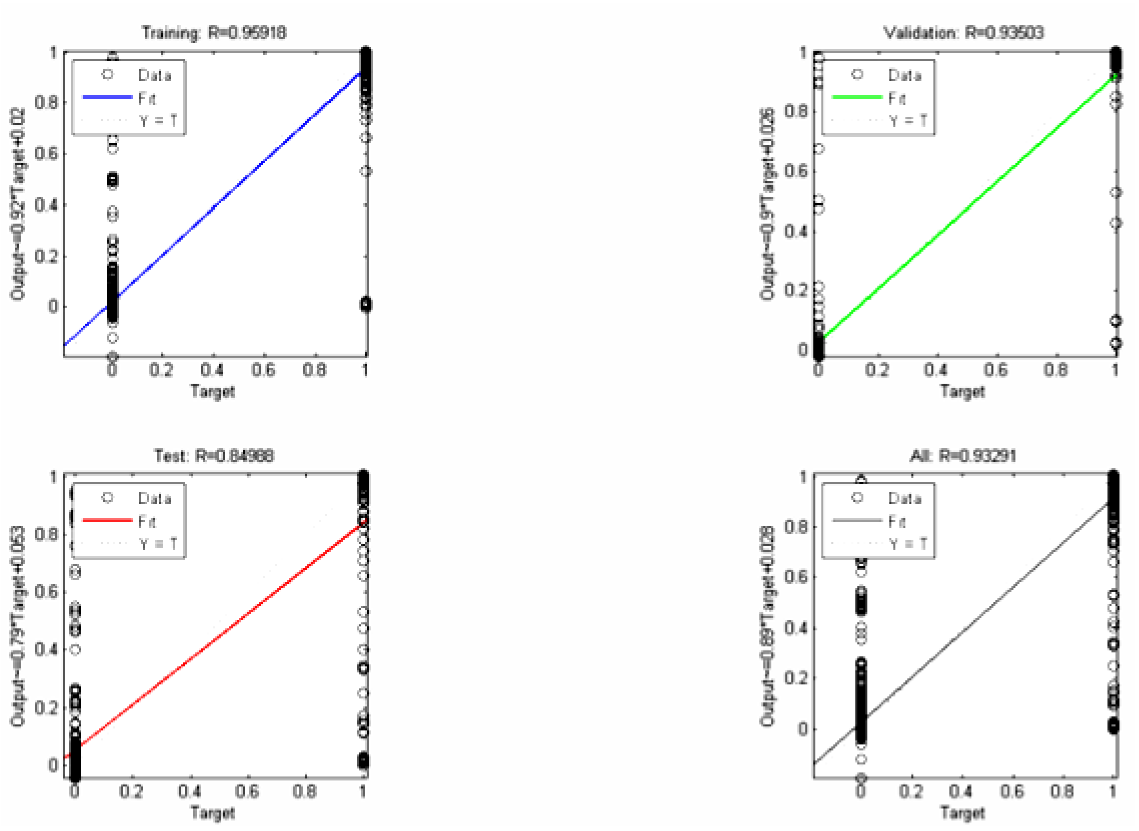




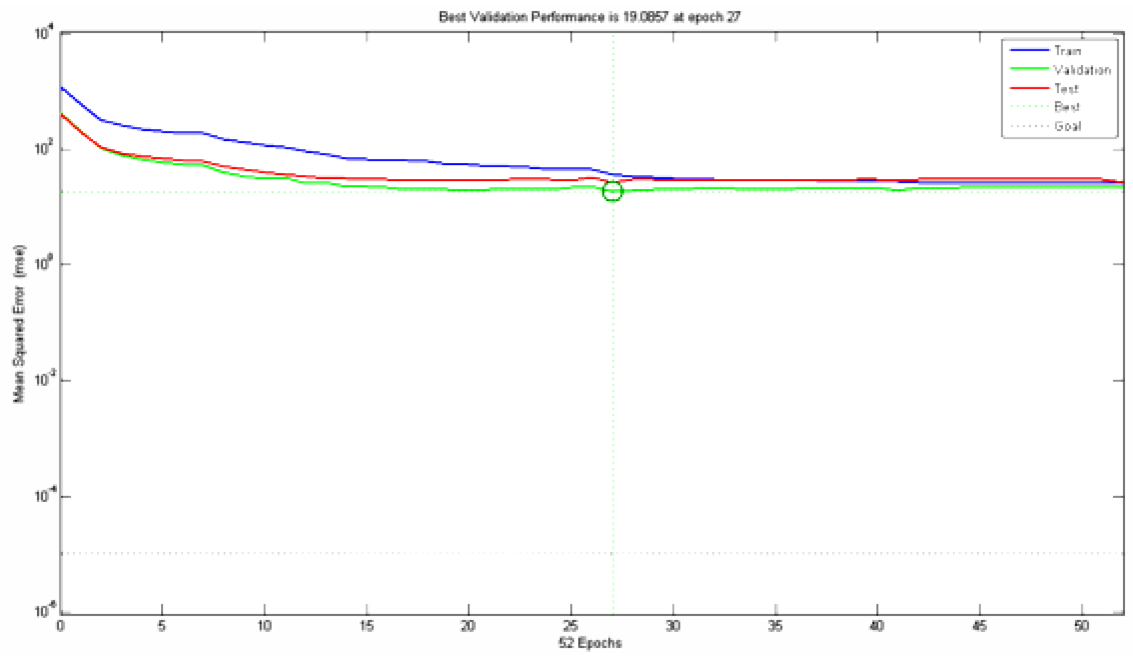
2) [4 3 3 3]

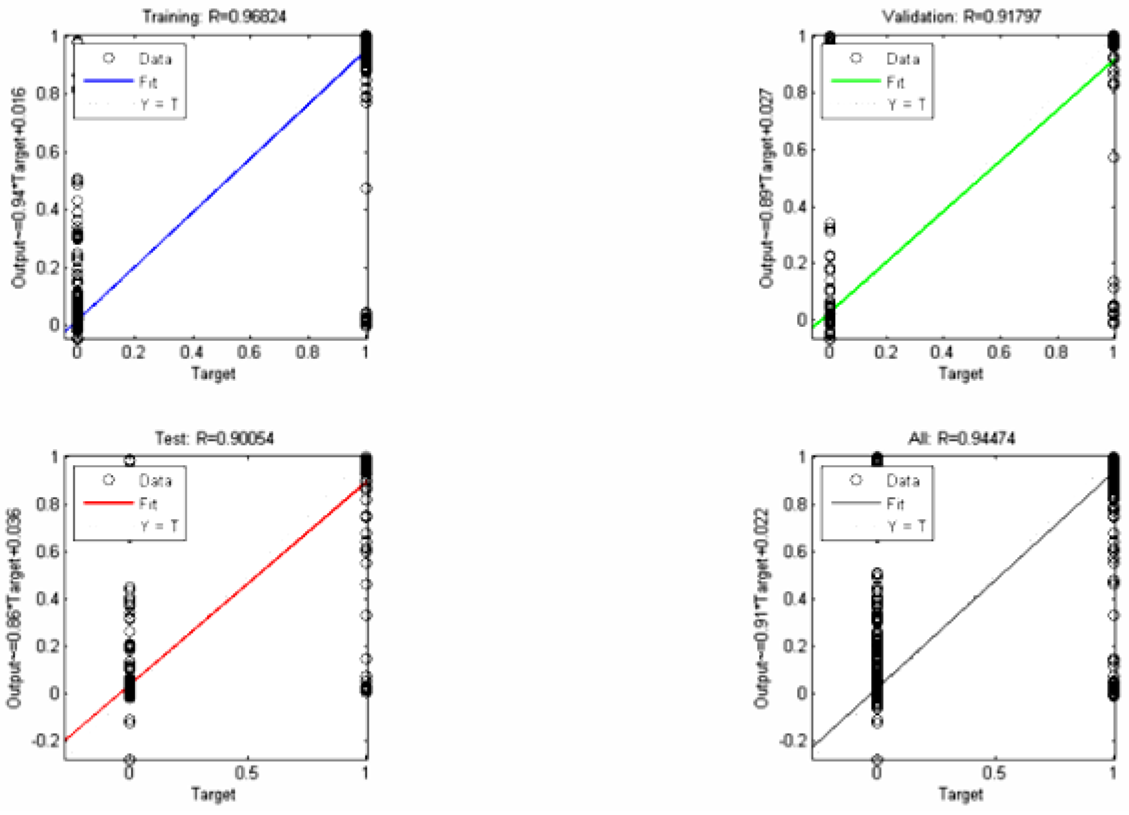
a) Levenberg–Marquardt



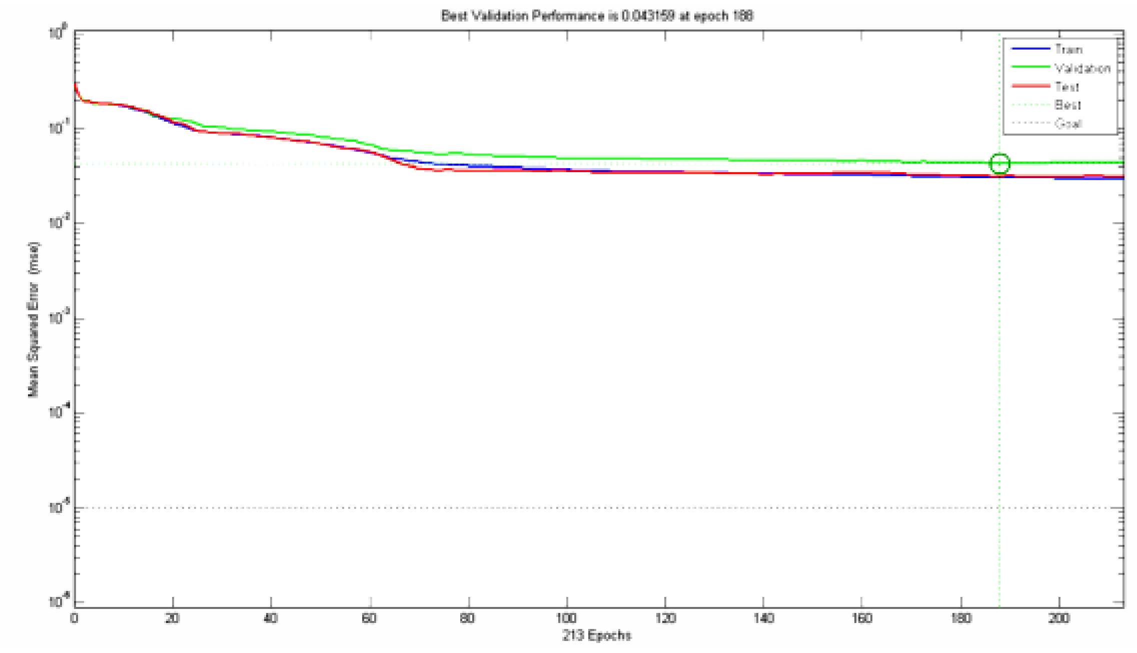


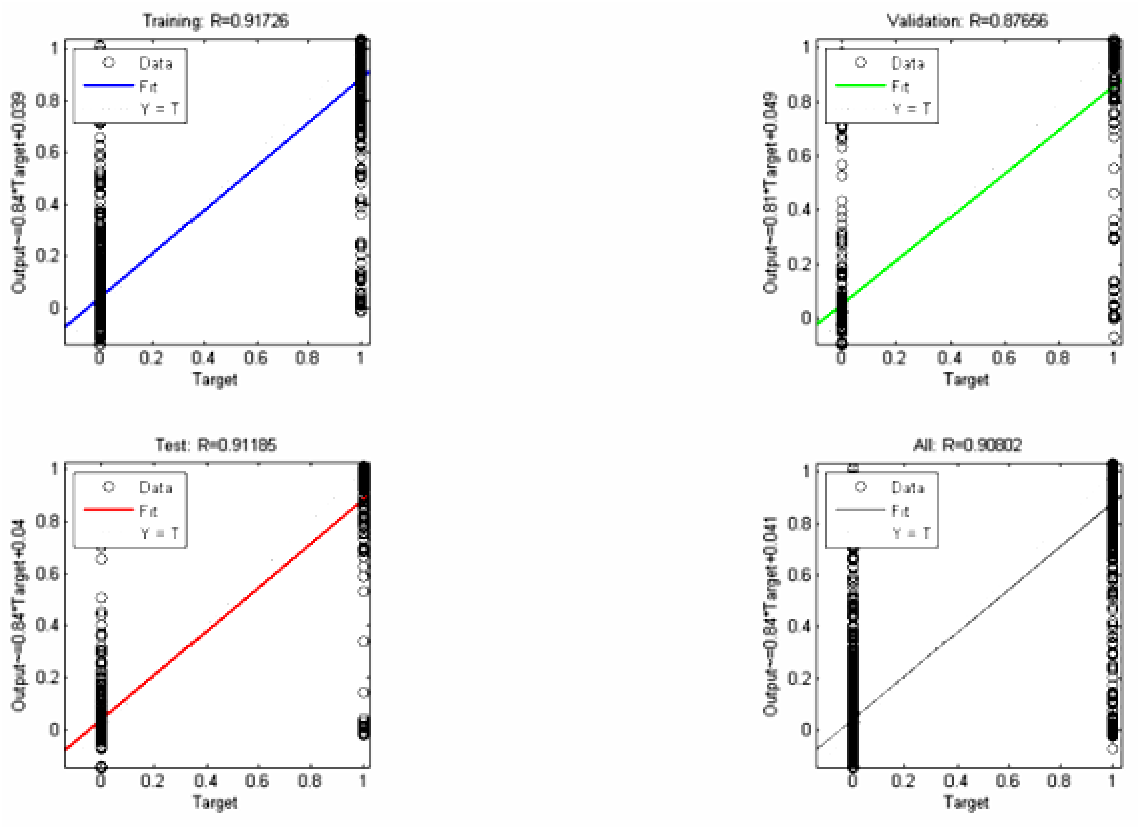
b) Bayesian Regularisation



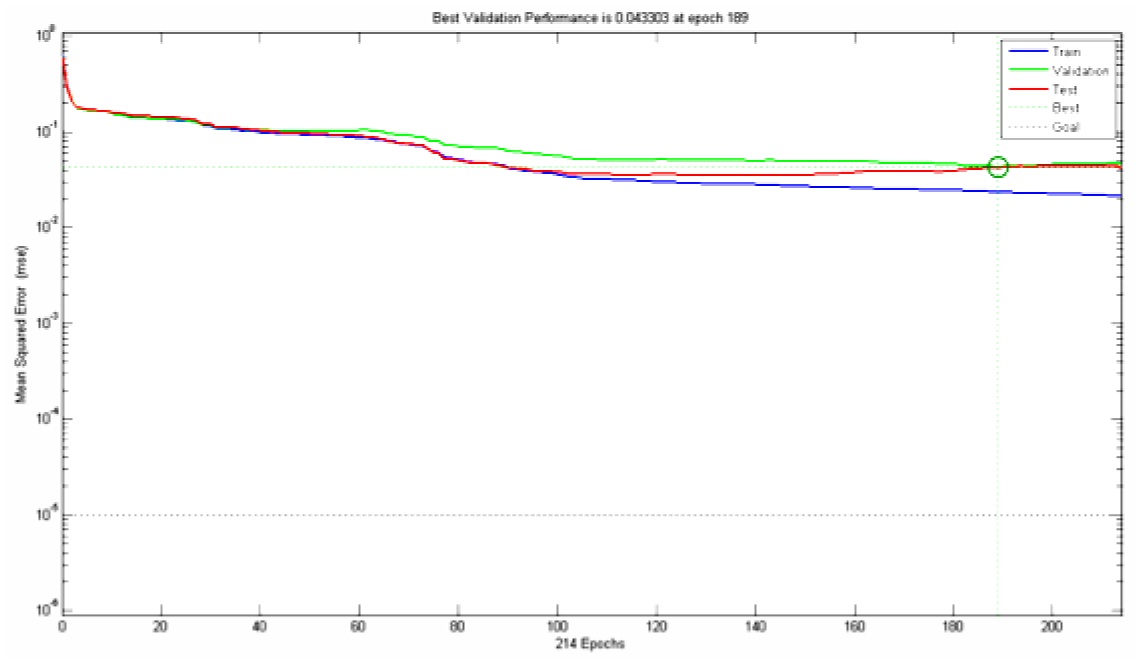


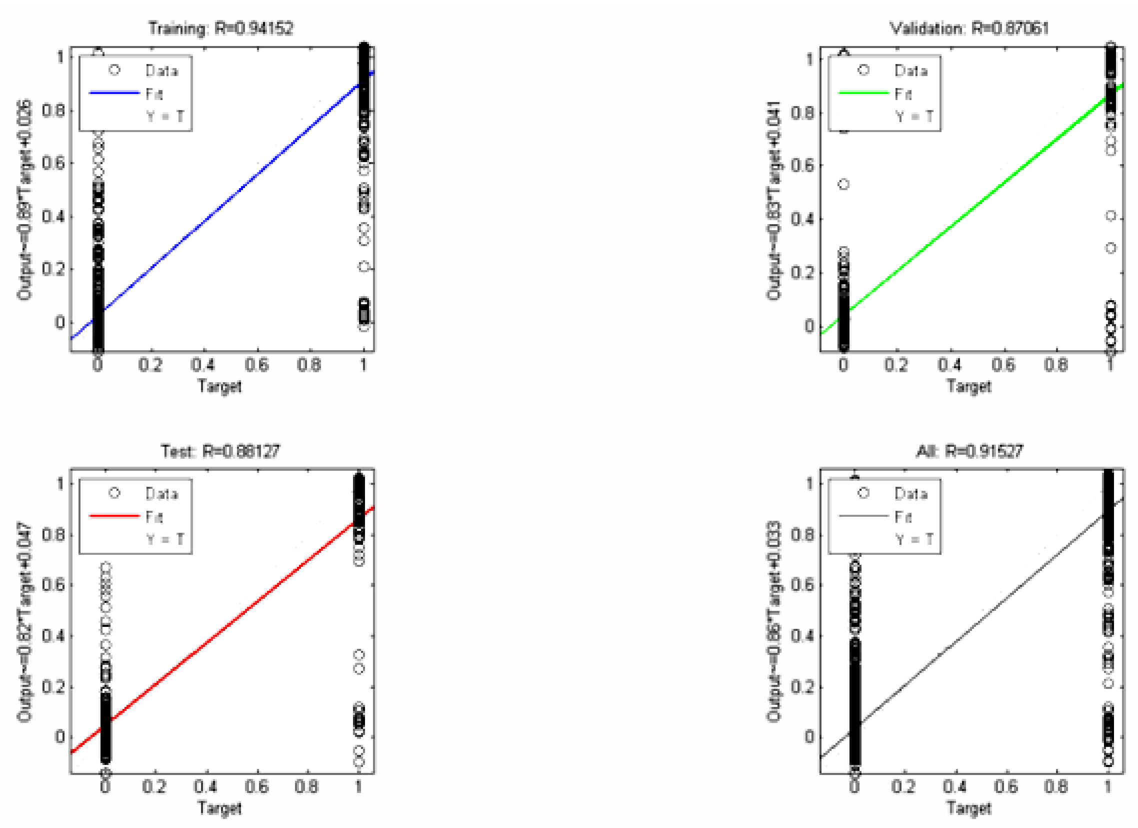
c) Resilient Backpropagation



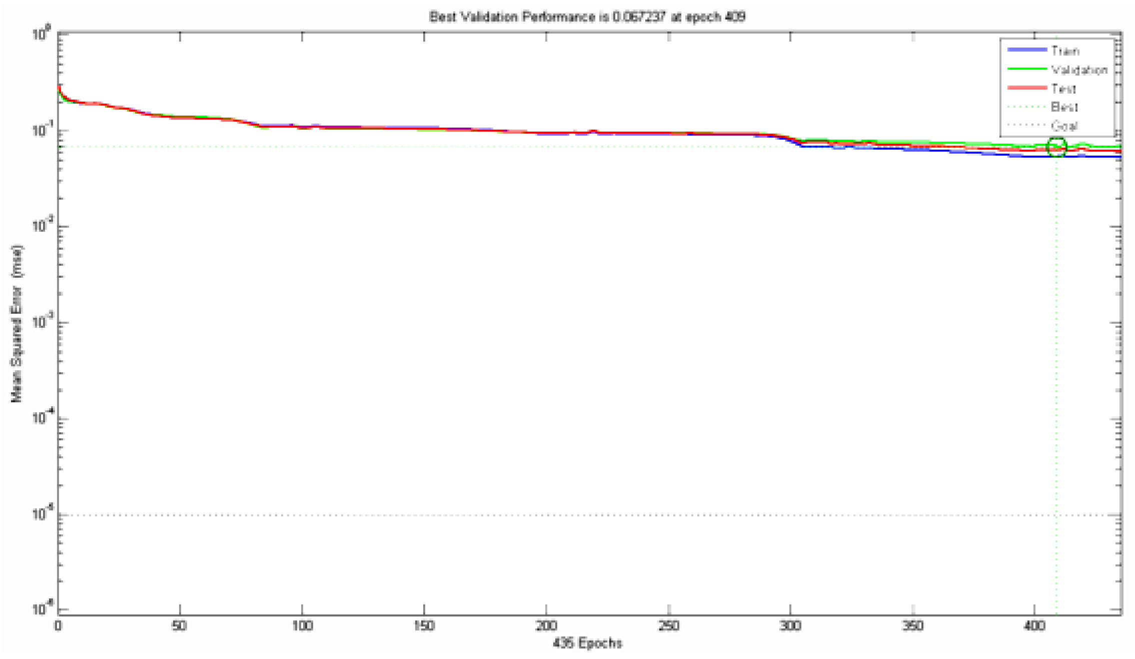


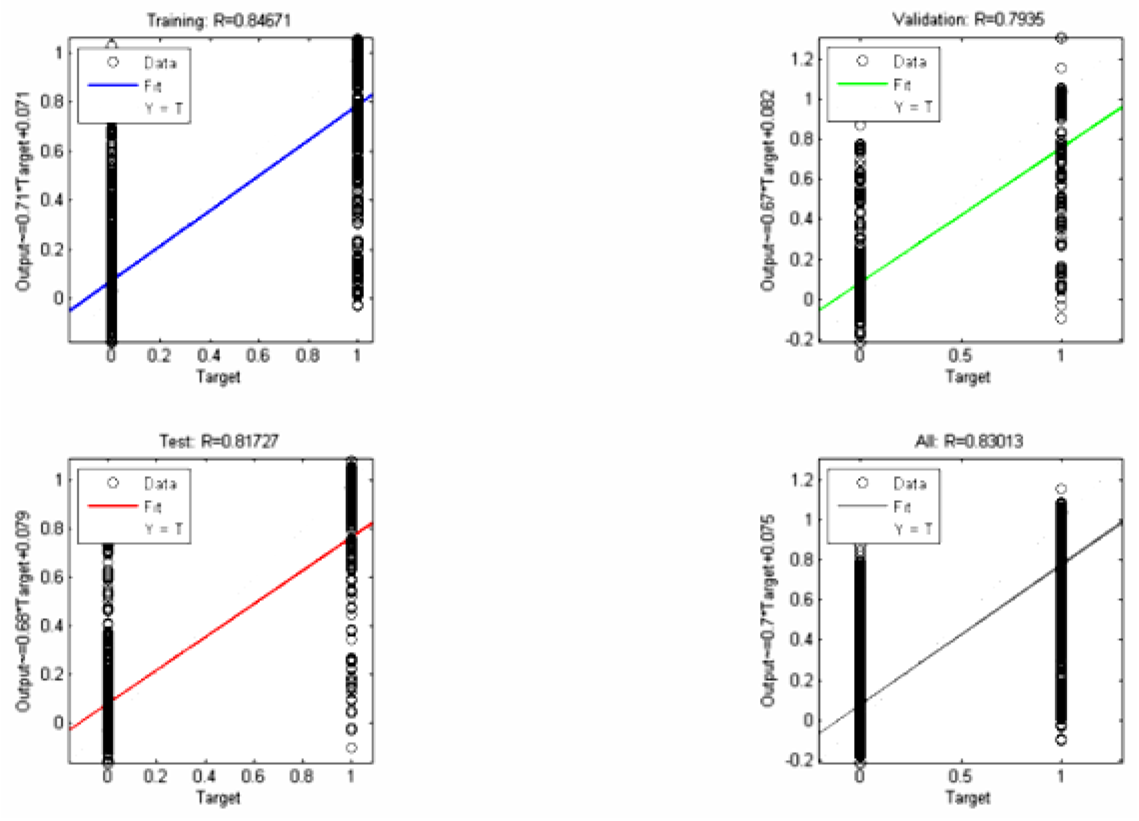
d) Scaled Conjugate Gradient



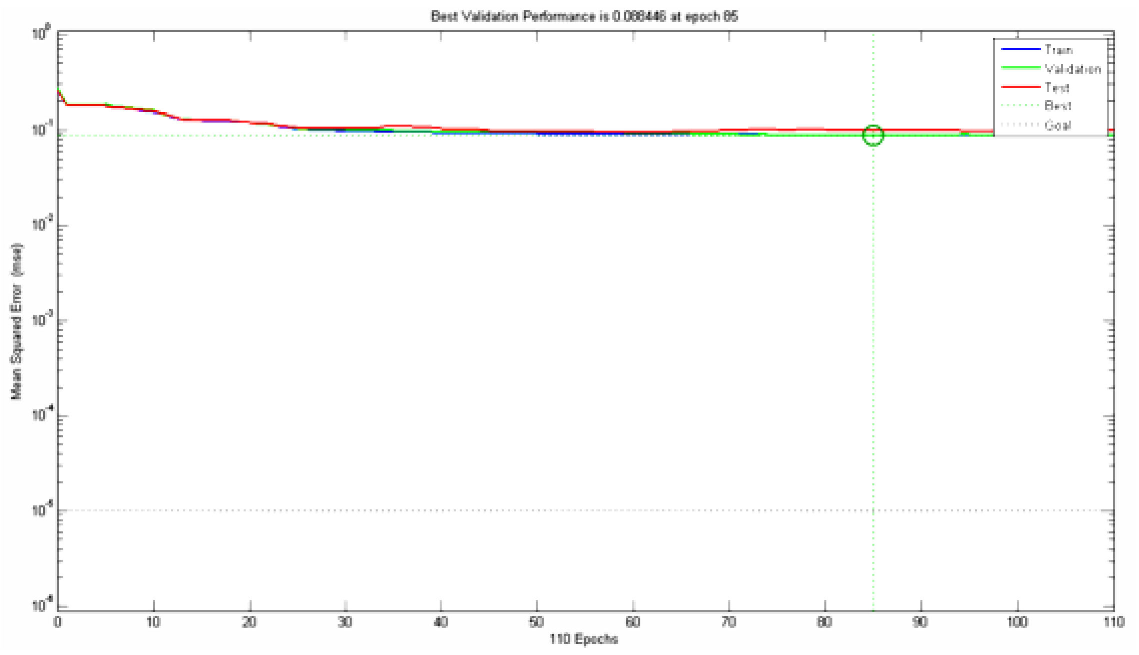


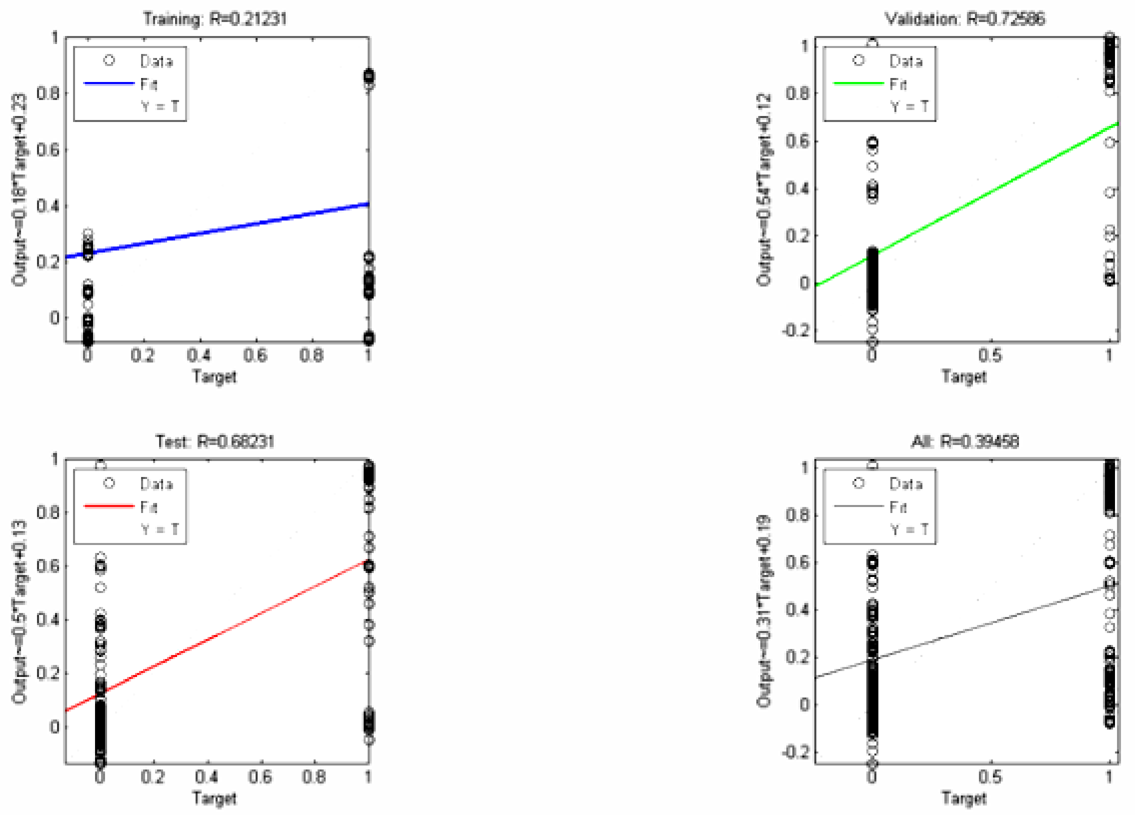
e) Variable Learning Rate Backpropagation





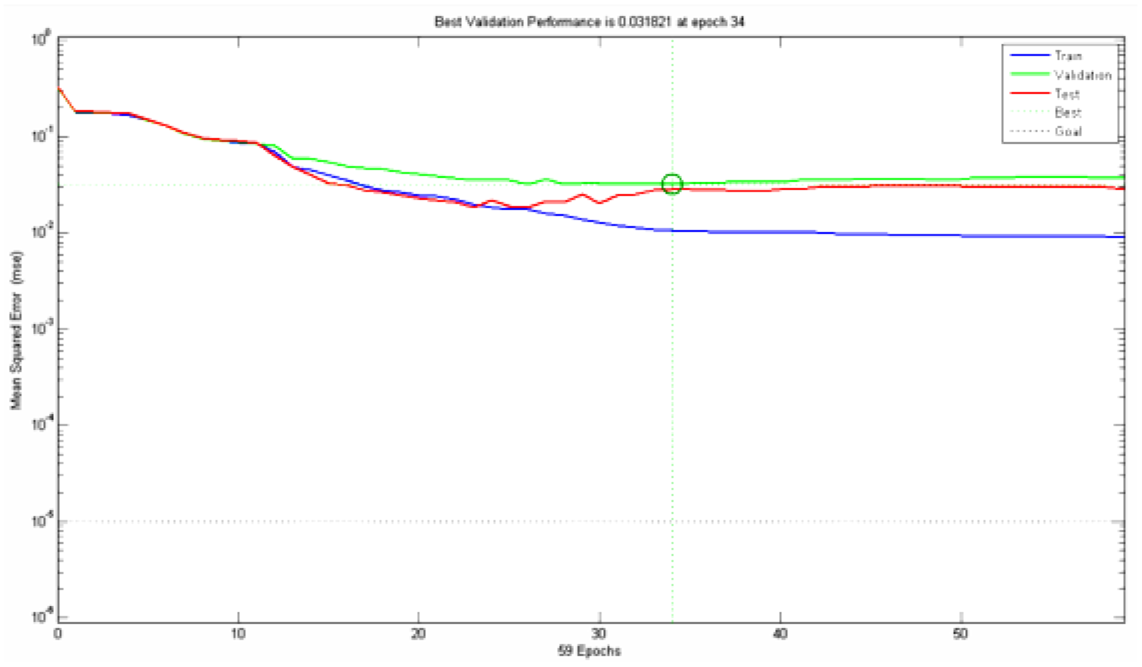
f) BFGS quasi–Newton Backpropagation

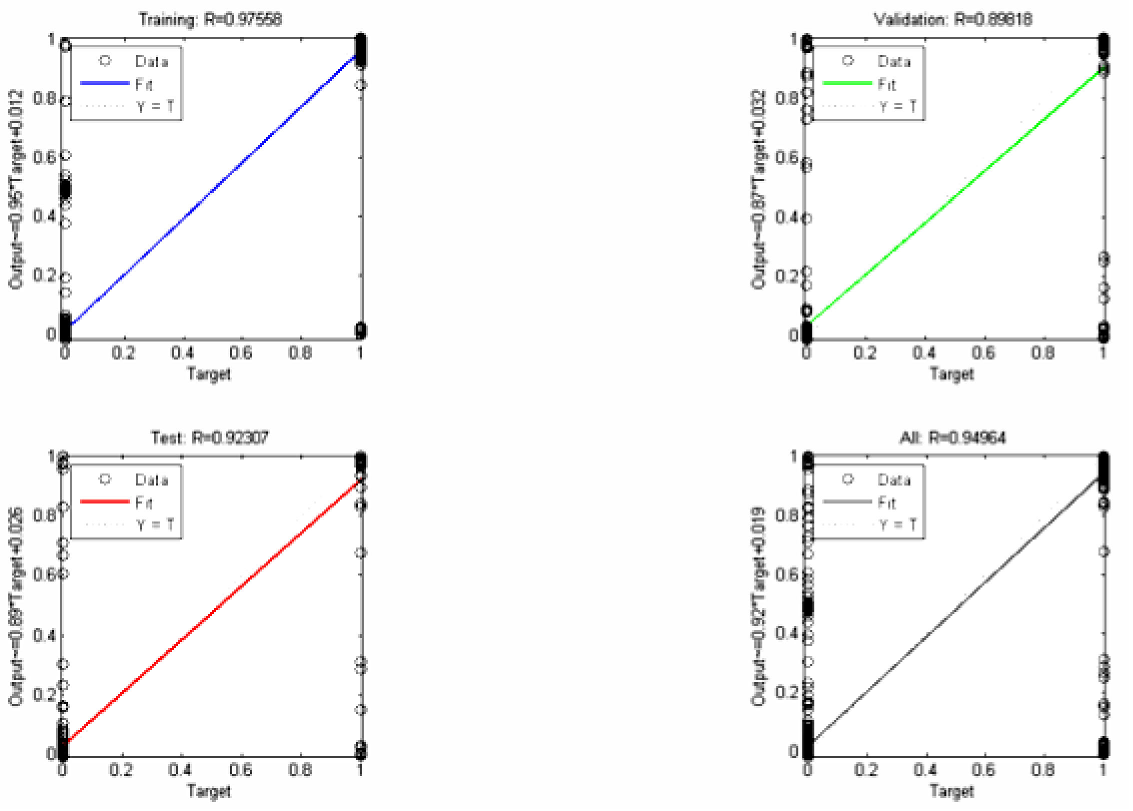




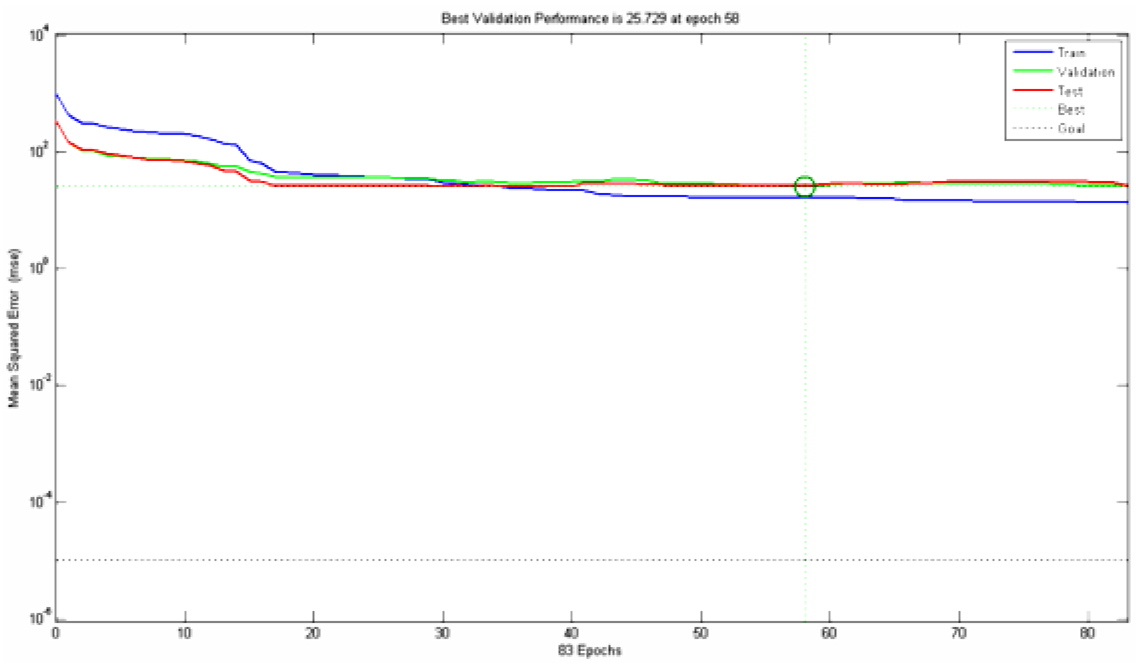
3) [5 4 3 3]

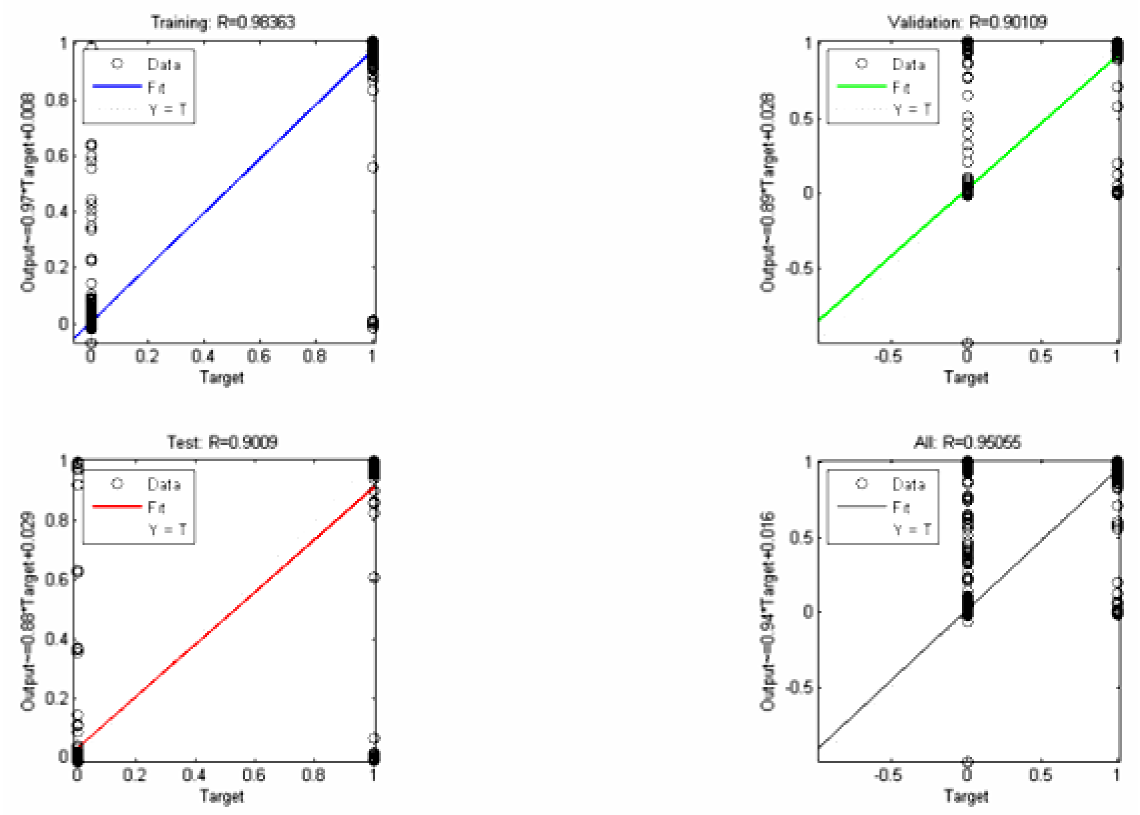
a) Levenberg–Marquardt



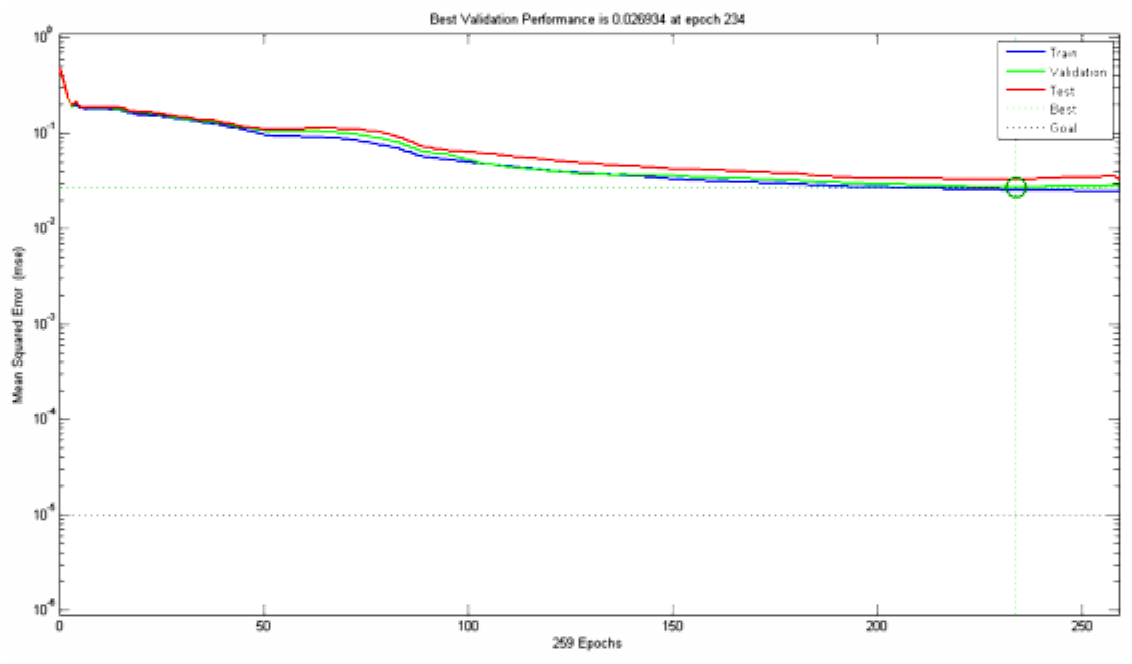


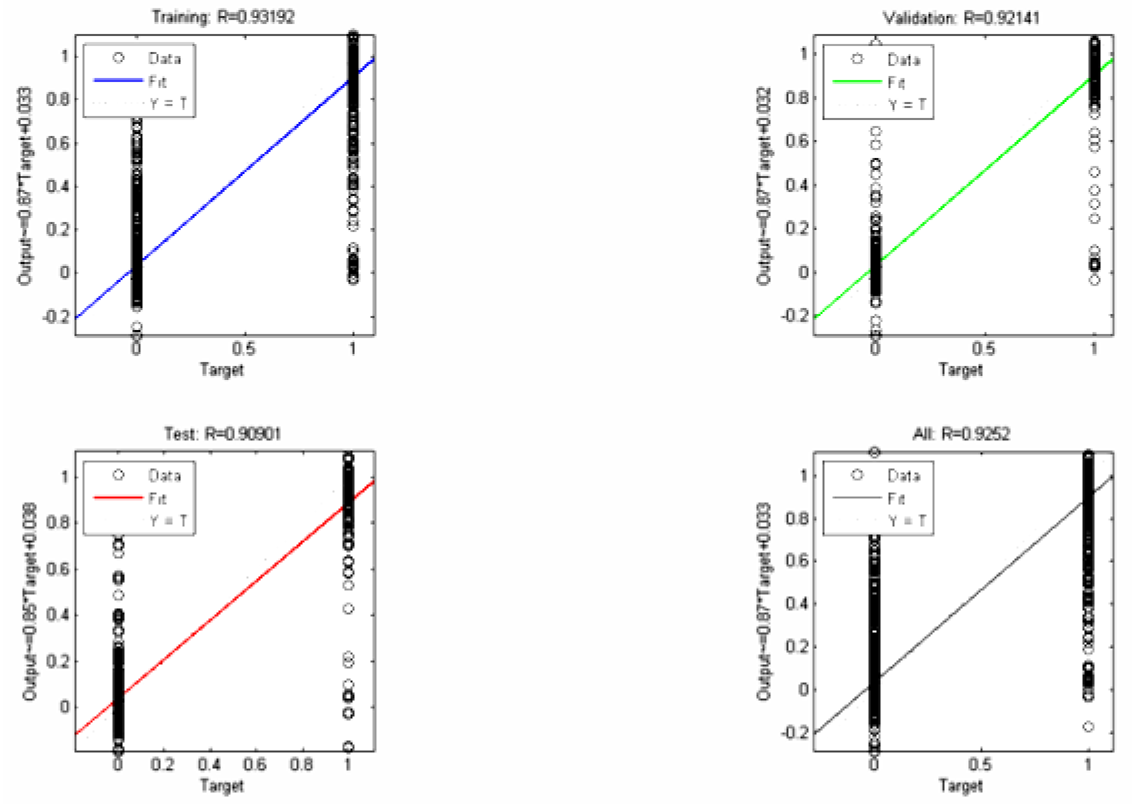
b) Bayesian Regularisation



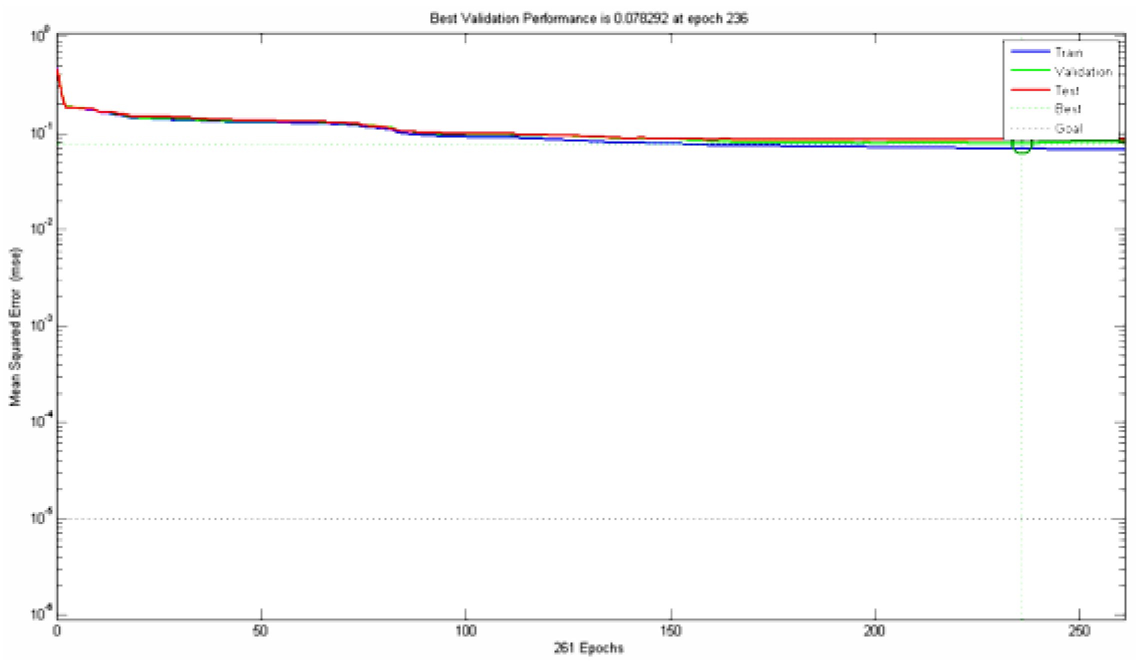


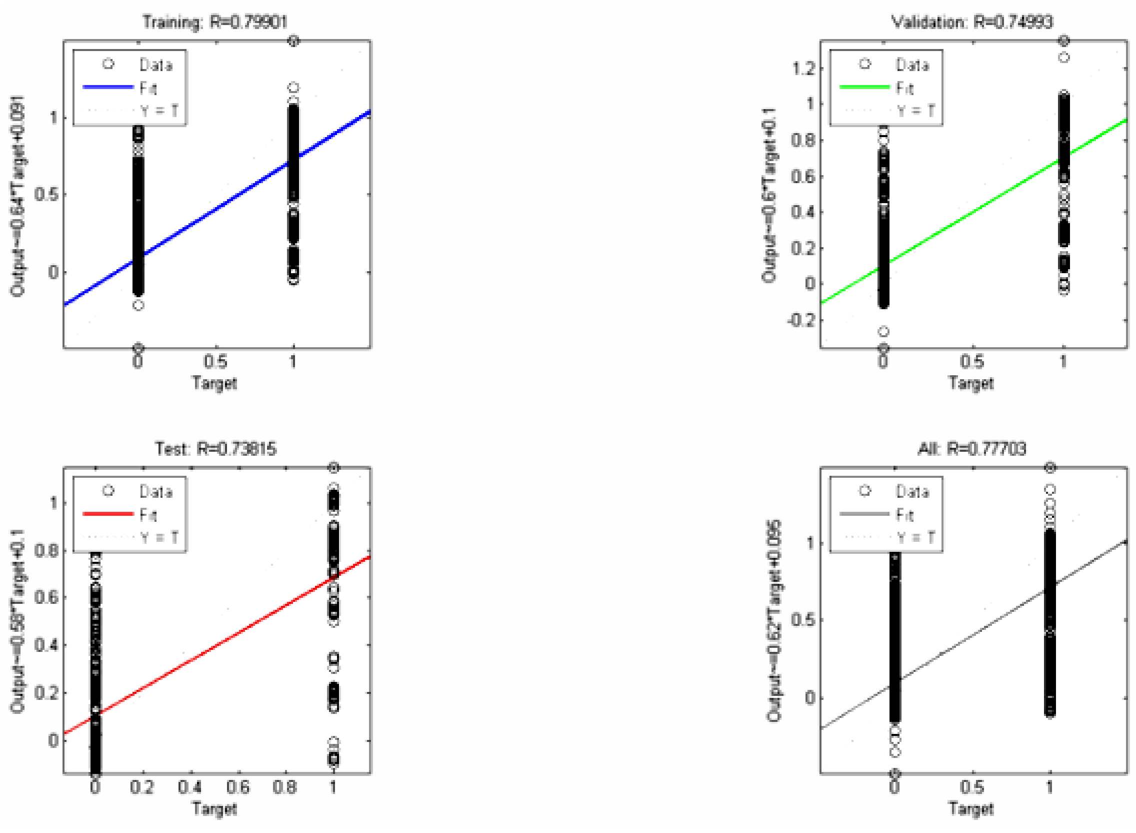
c) Resilient Backpropagation



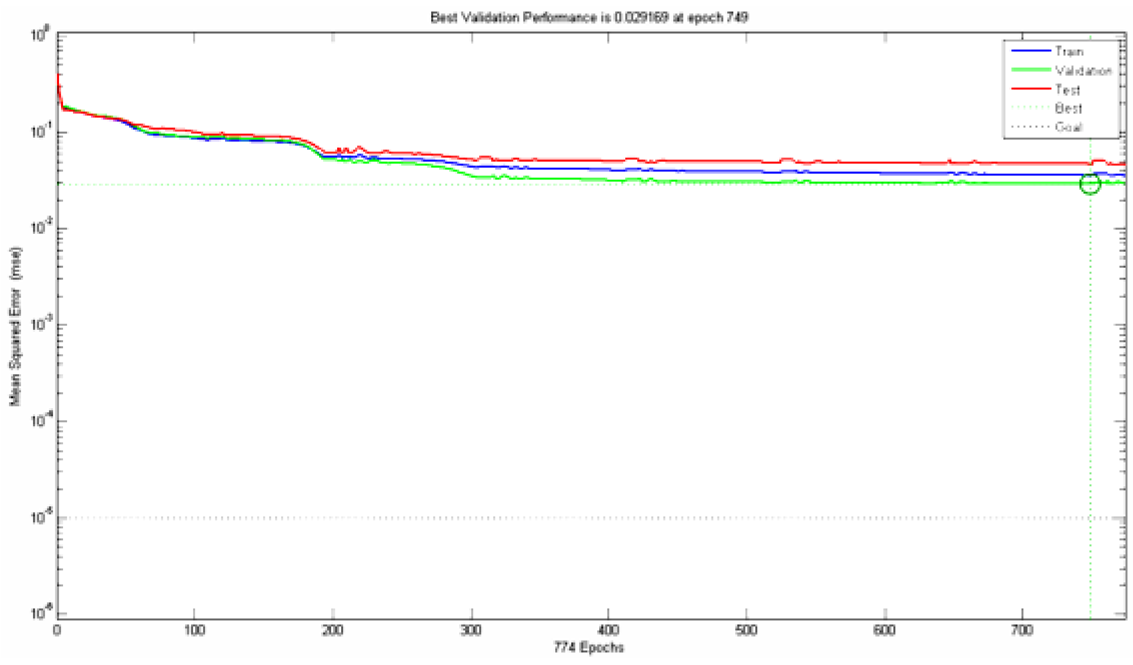


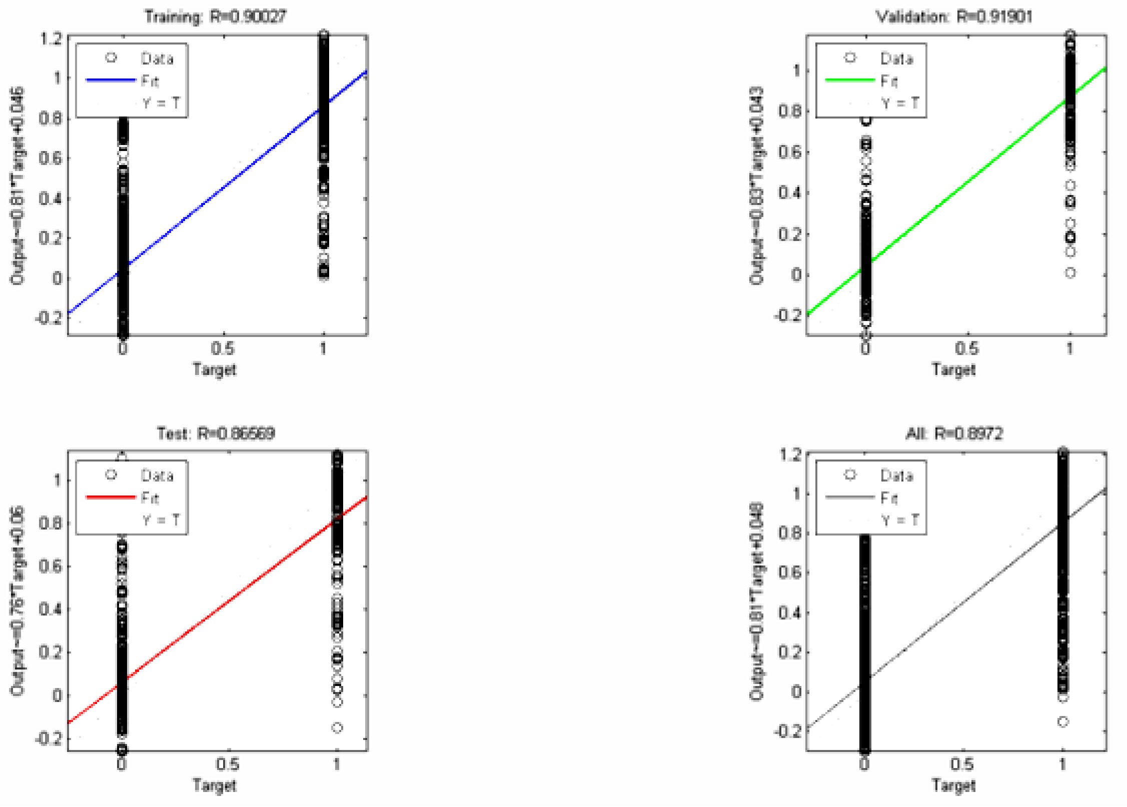
d) Scaled Conjugate Gradient



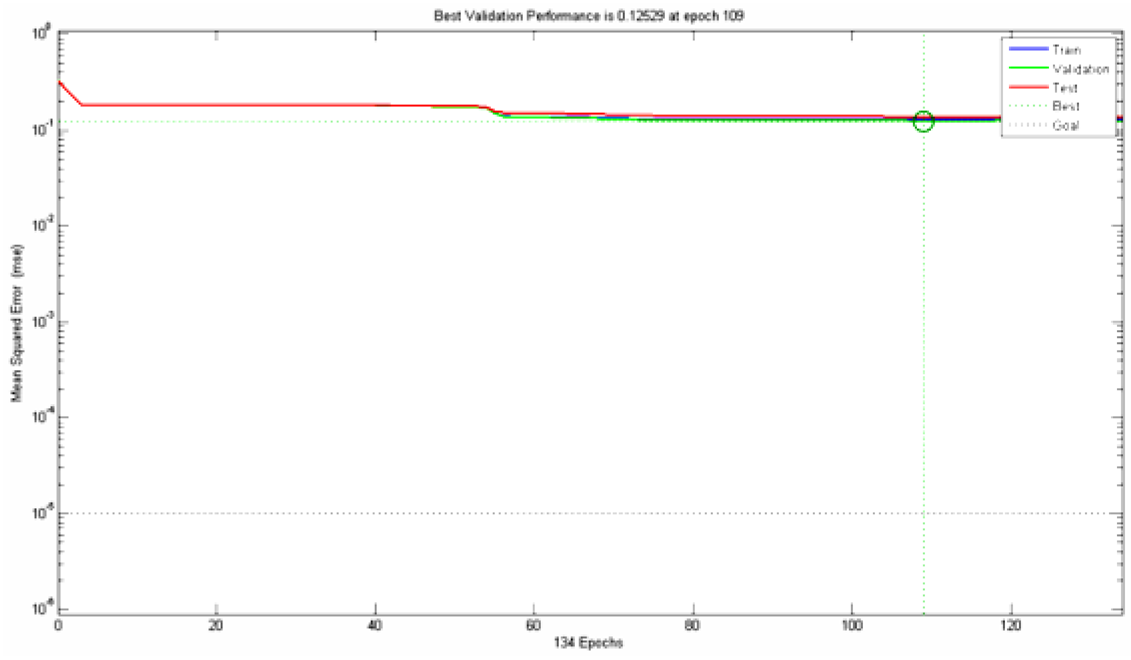


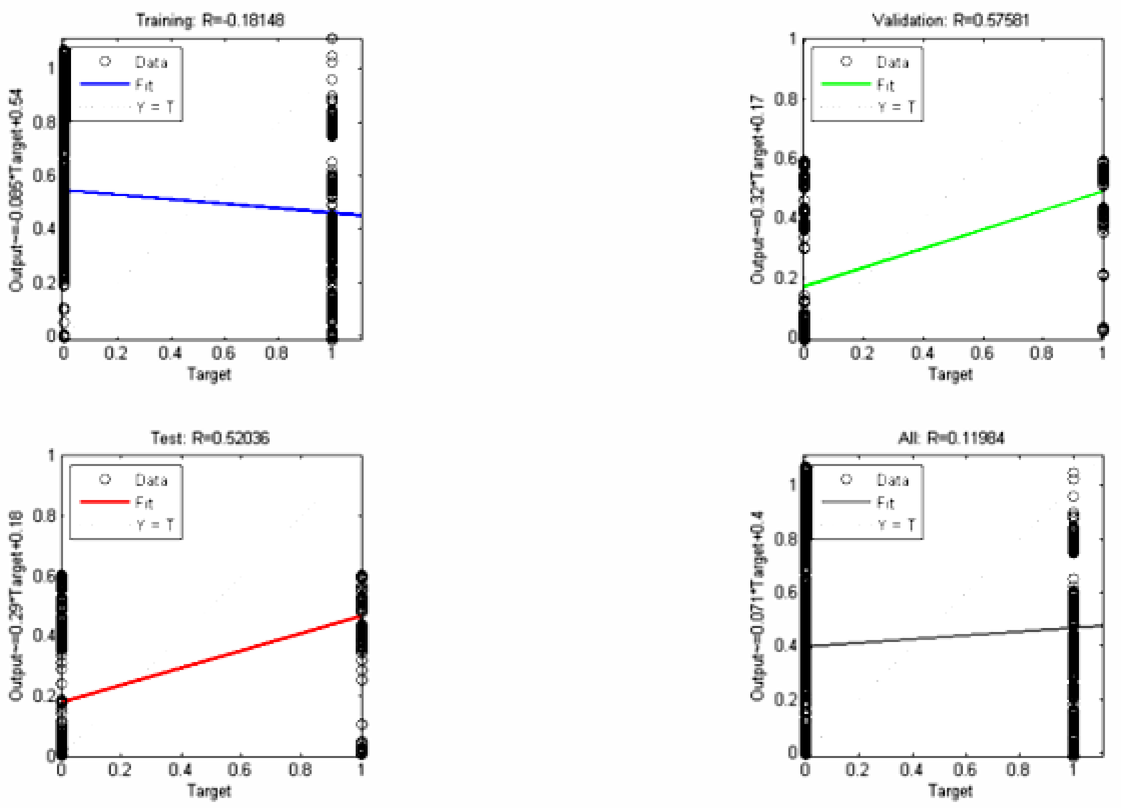
e) Variable Learning Rate Backpropagation





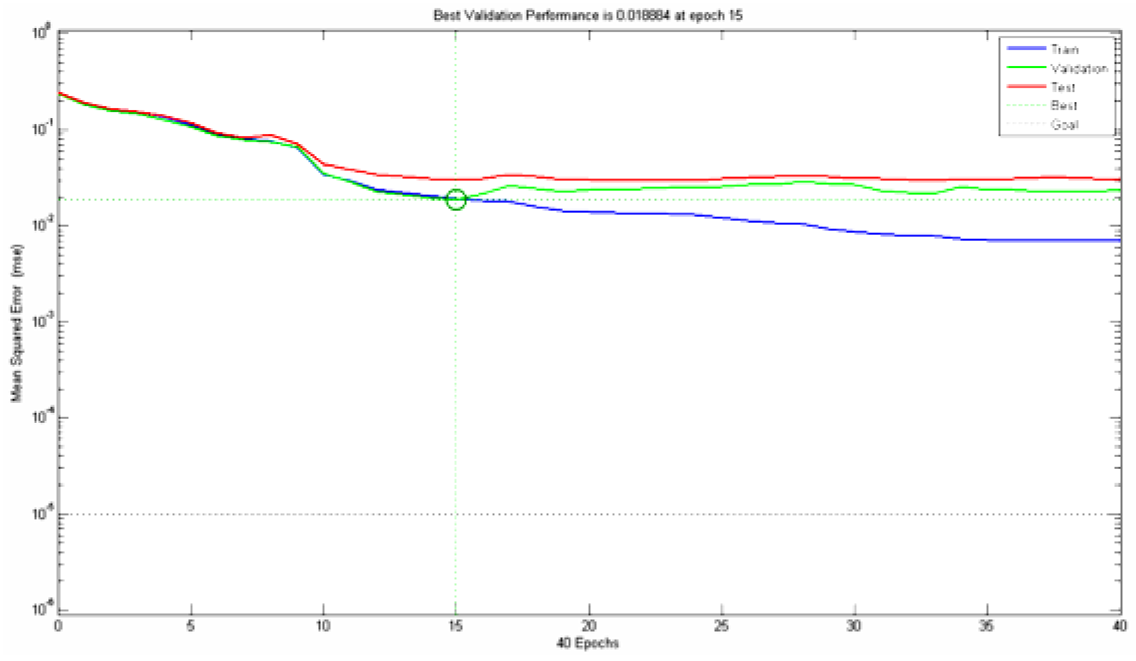
f) BFGS quasi–Newton Backpropagation

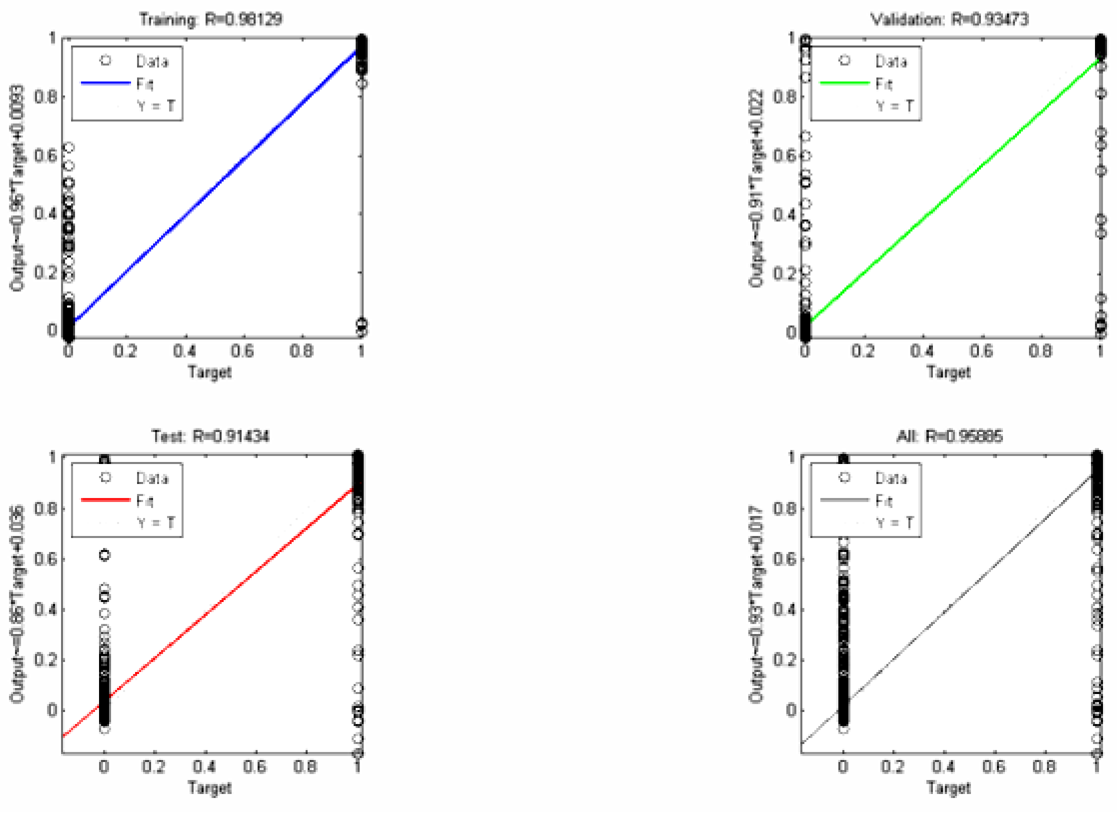




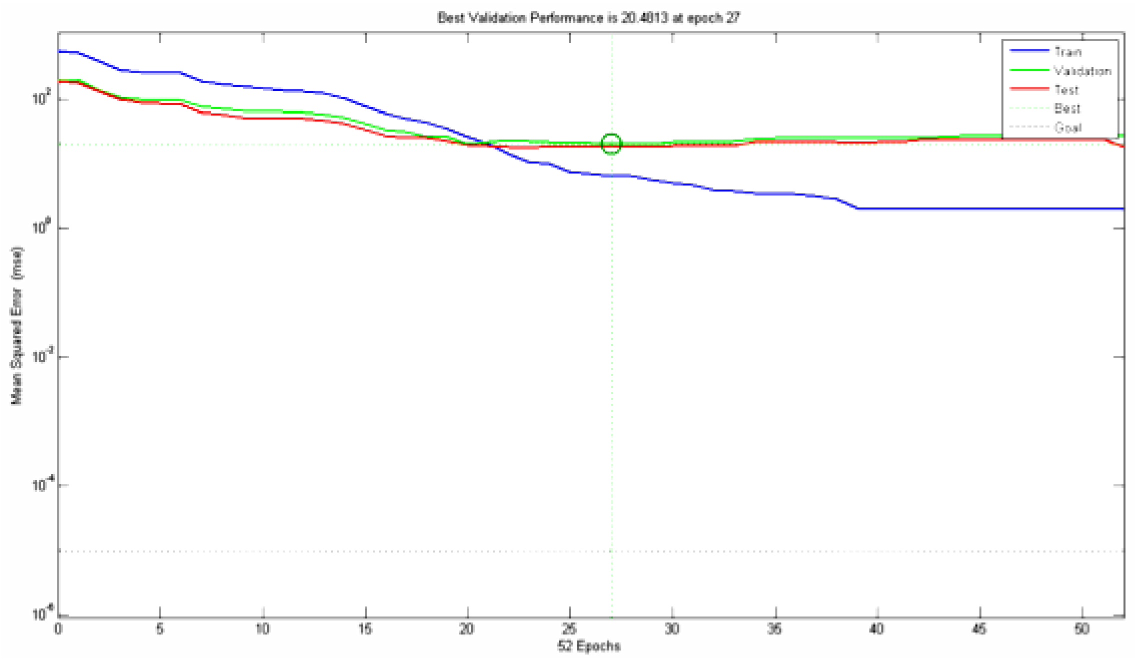
4) [8 5 4 3]

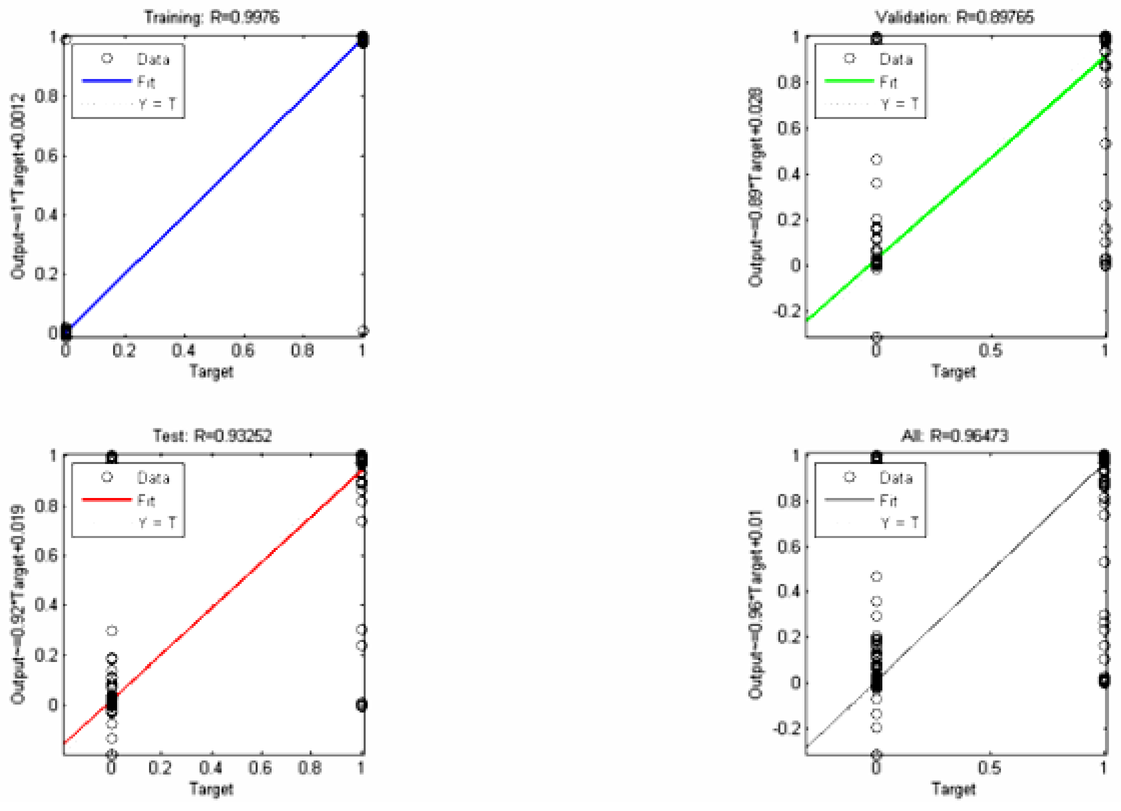
a) Levenberg–Marquardt



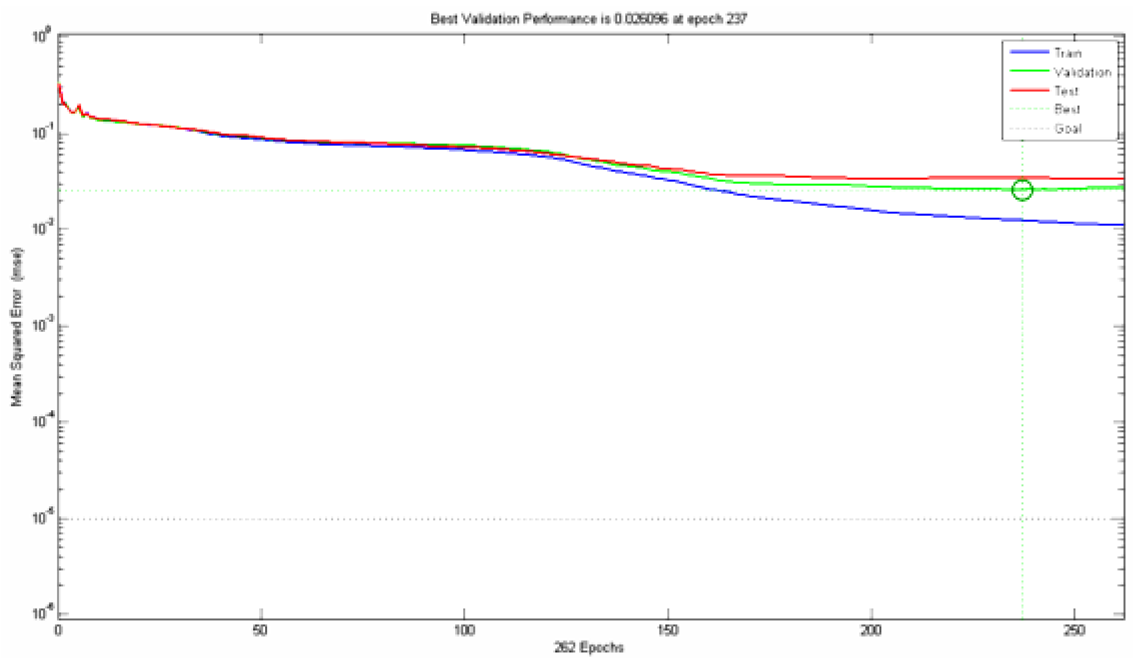


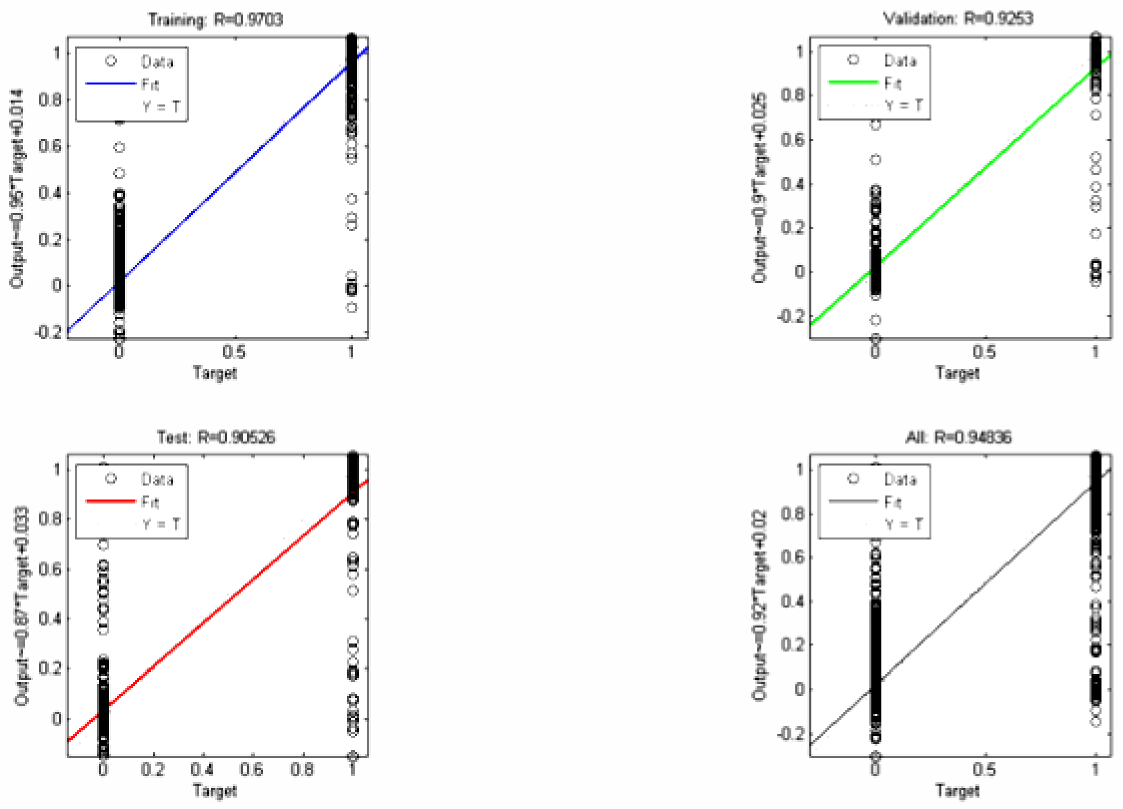
b) Bayesian Regularisation



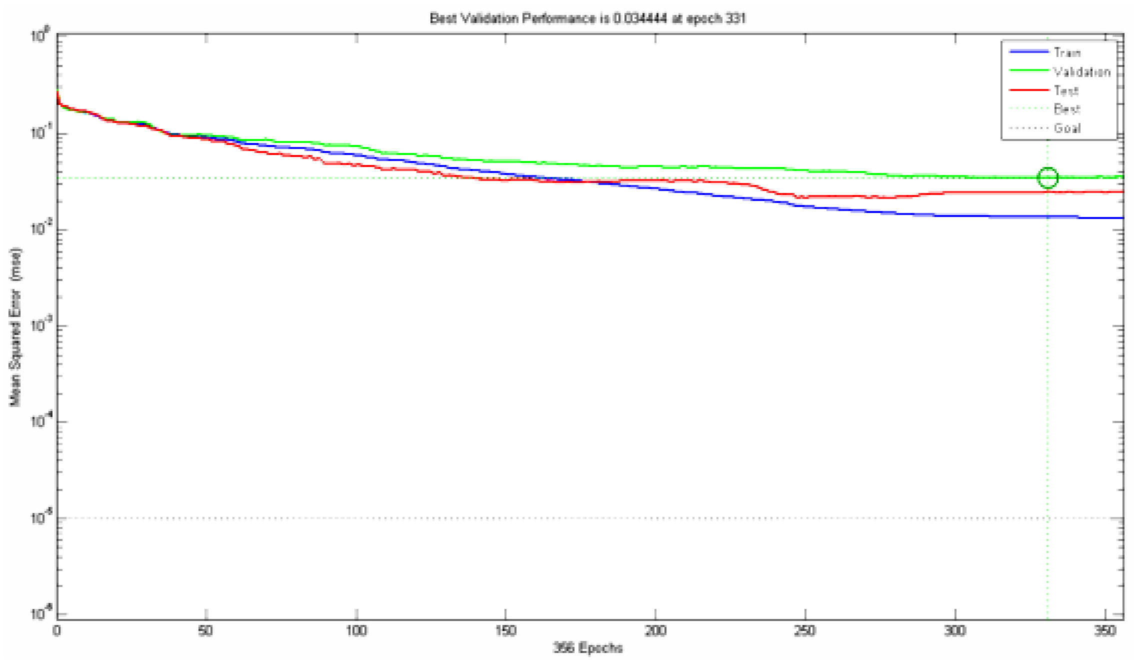


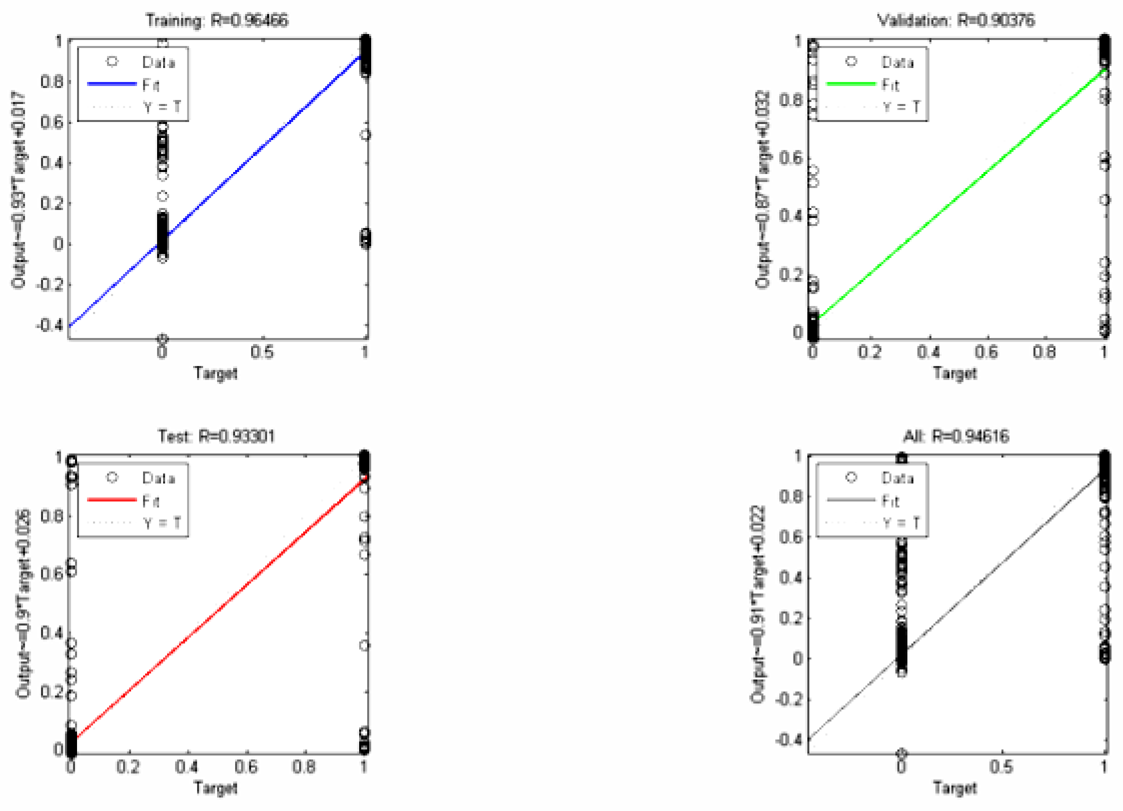
c) Resilient Backpropagation



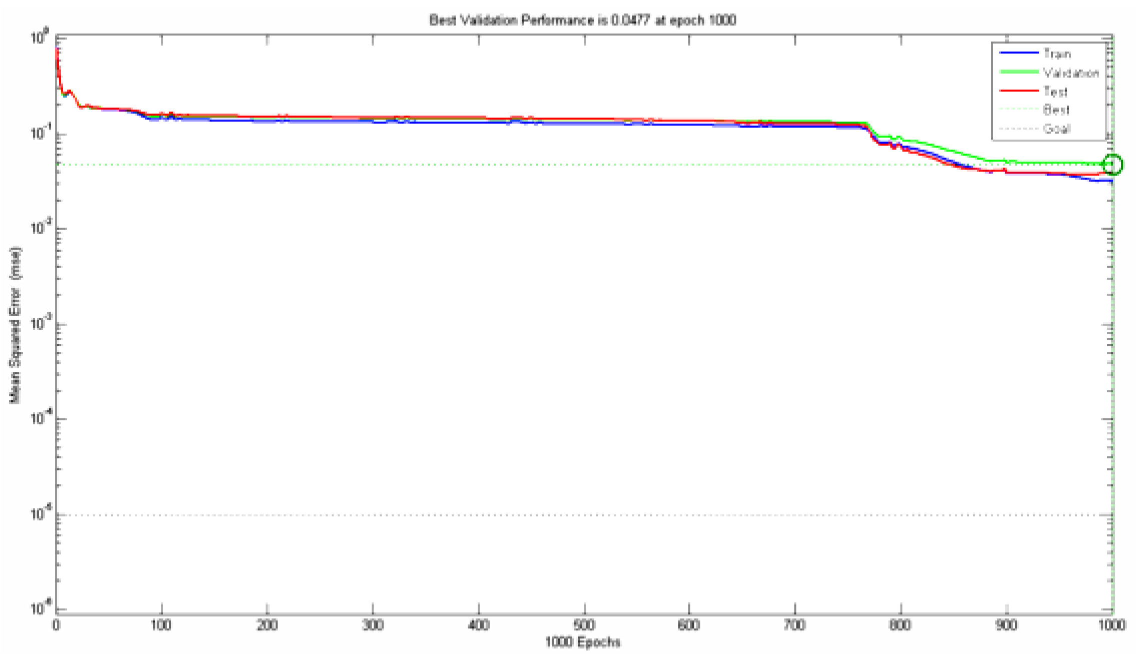


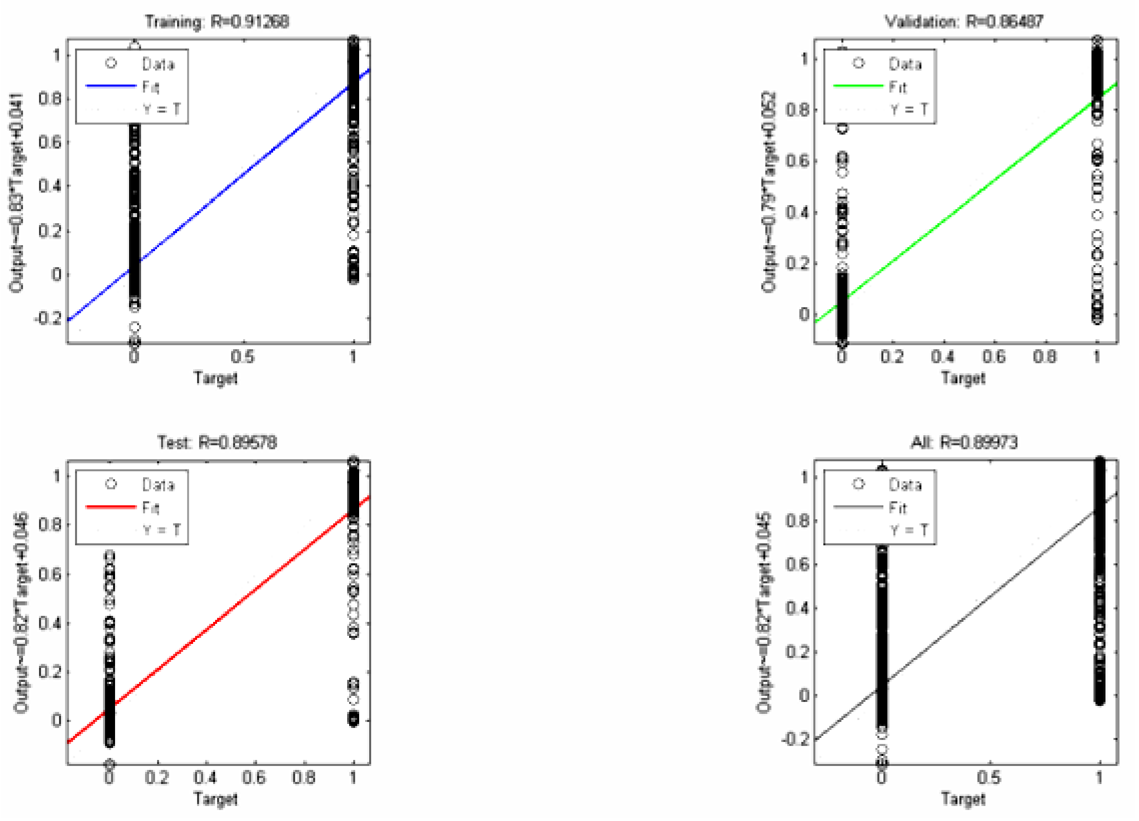
d) Scaled Conjugate Gradient



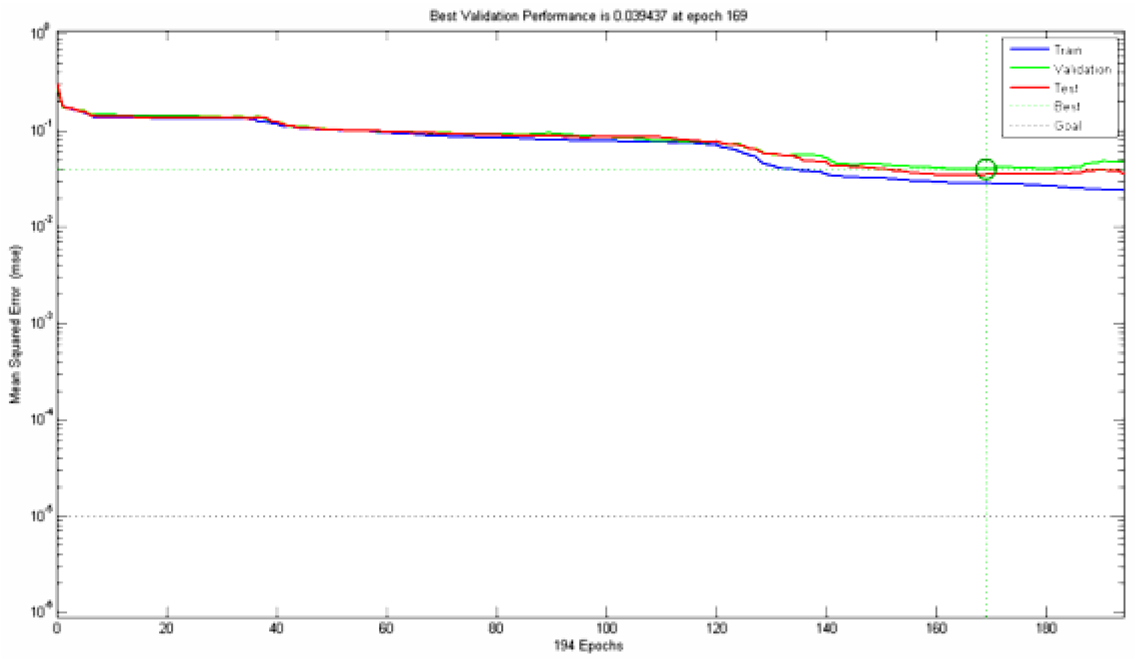


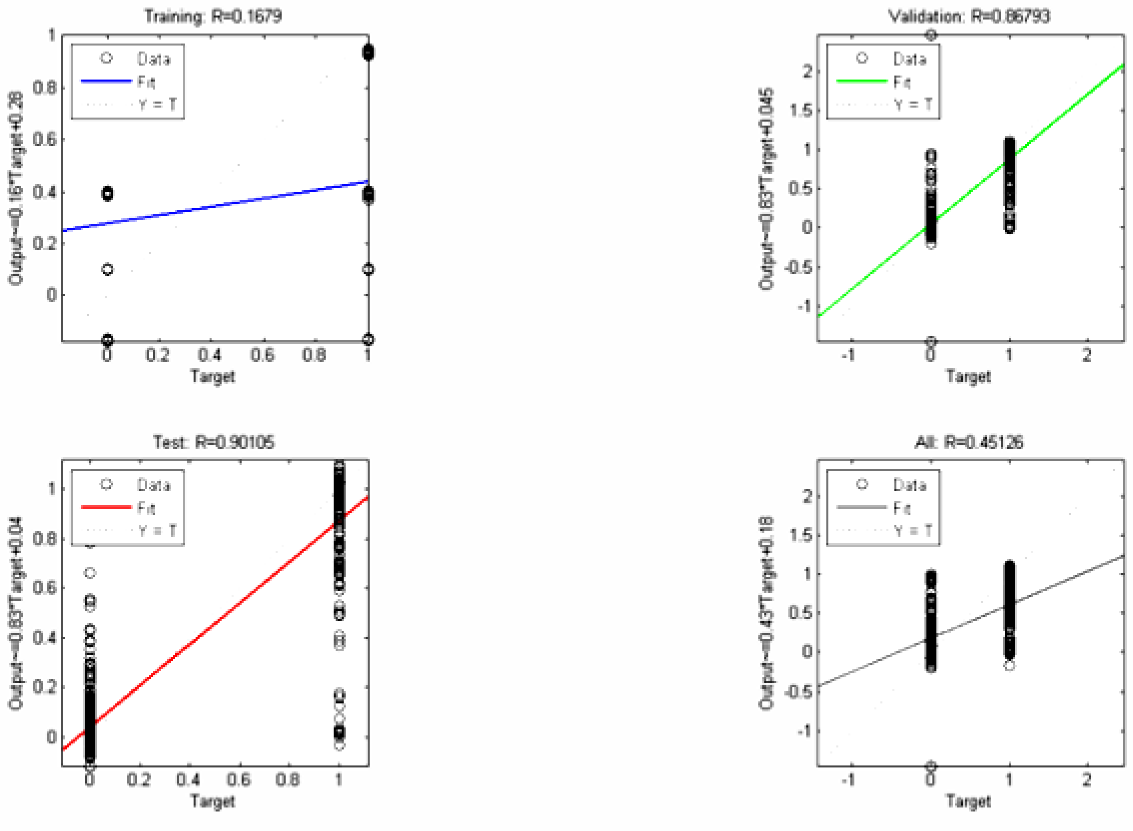
e) Variable Learning Rate Backpropagation





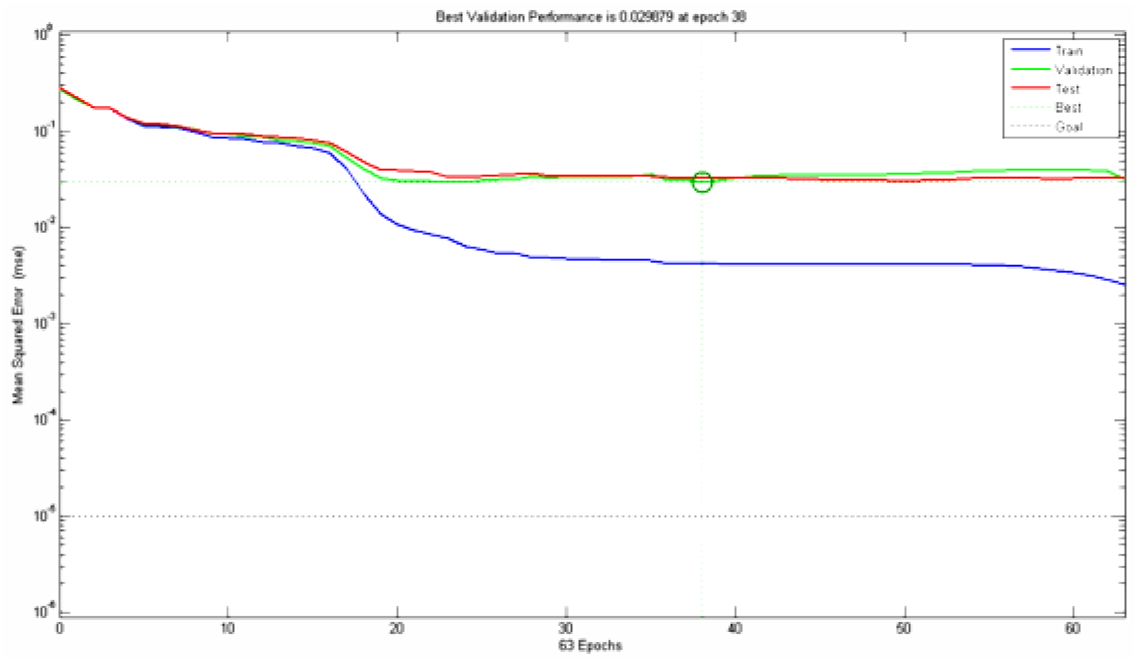
f) BFGS quasi–Newton Backpropagation

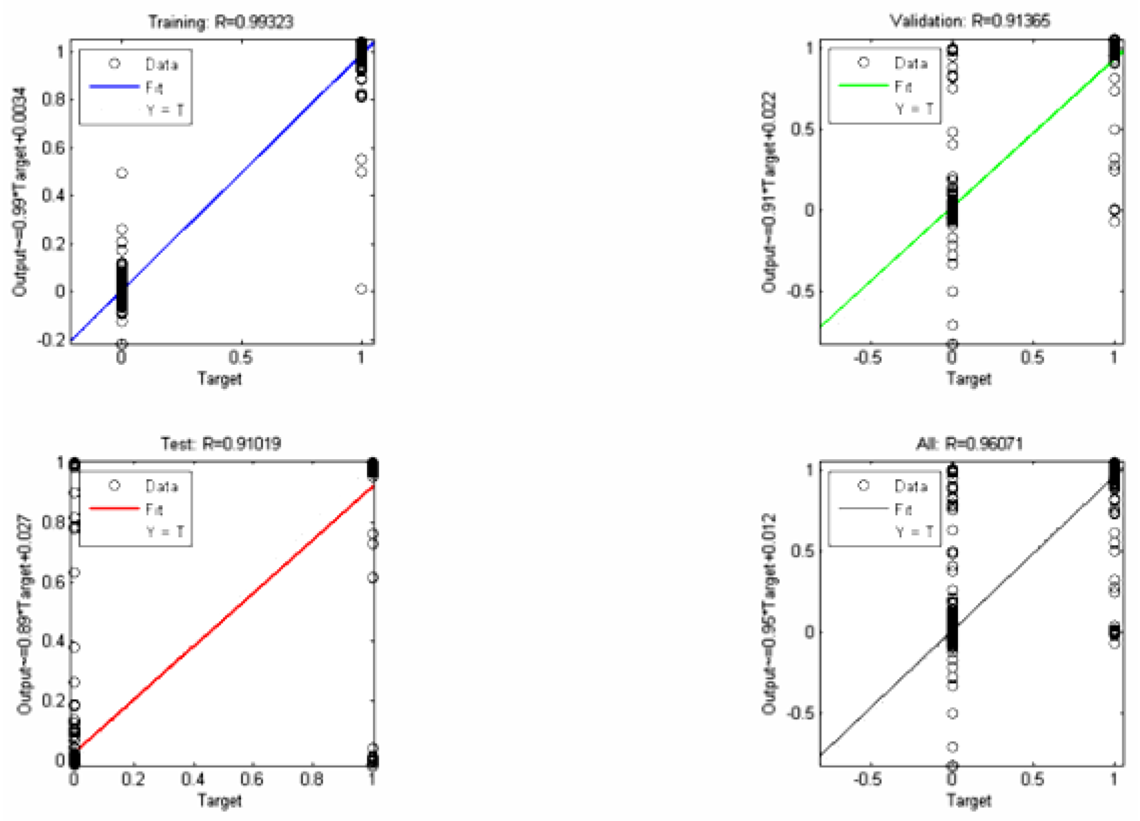




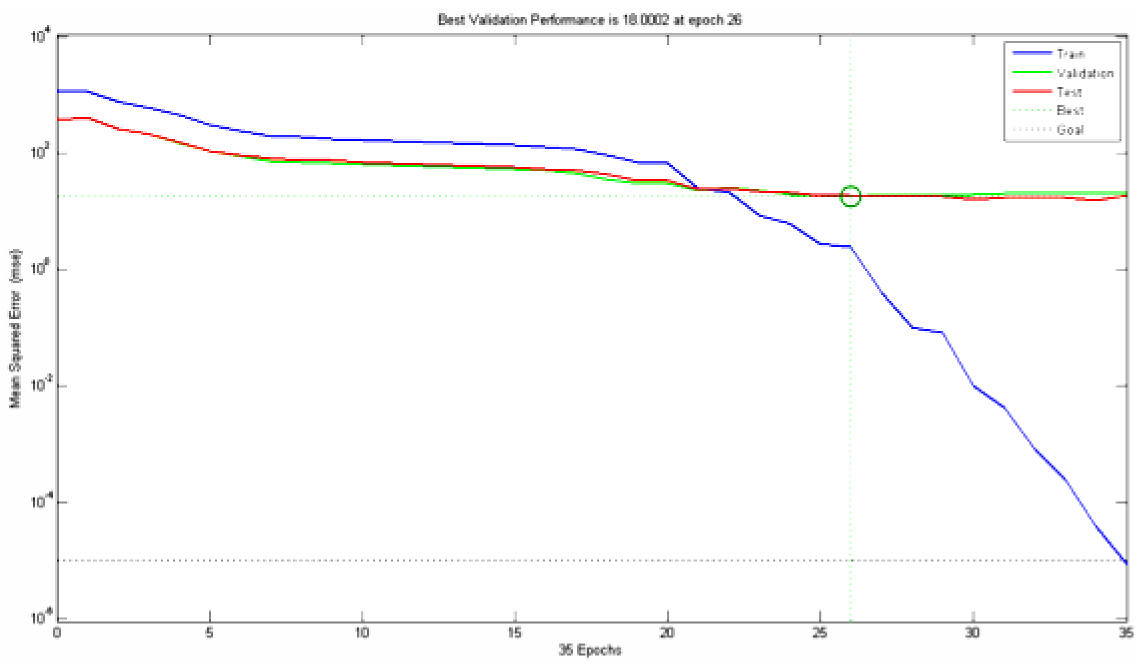
5) [10 8 4 3]

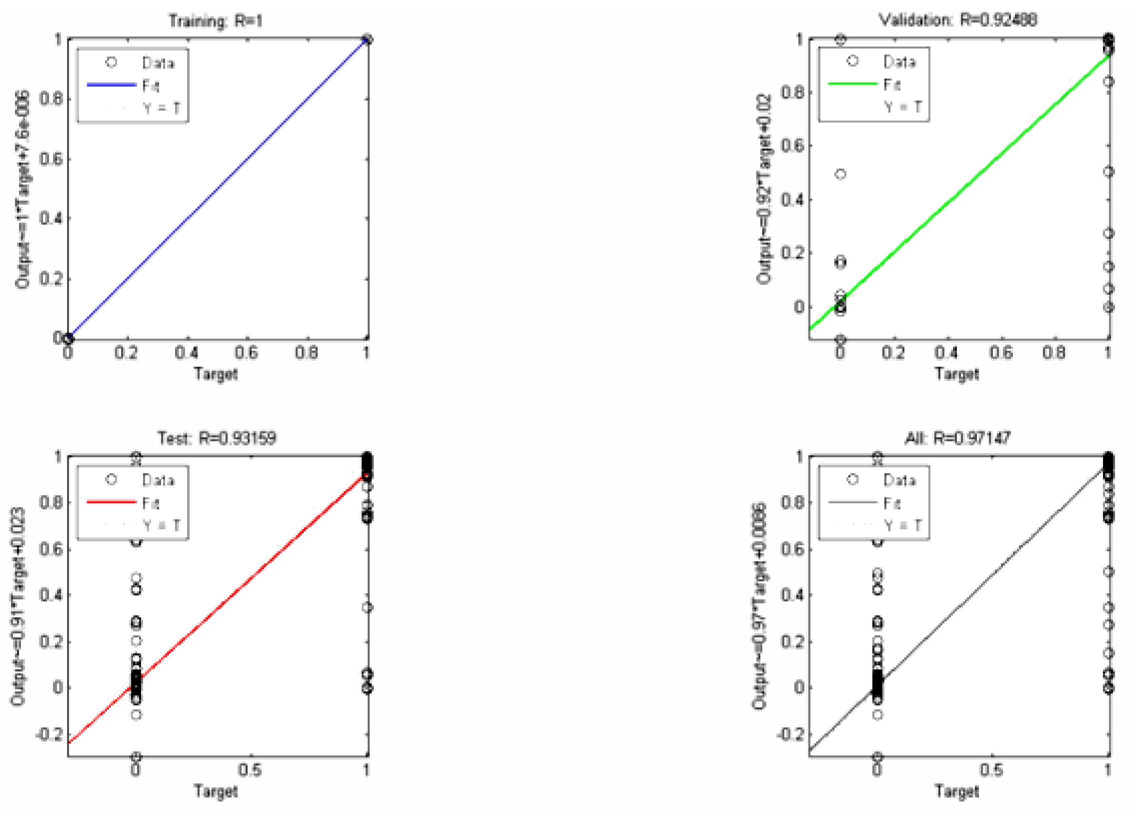
a) Levenberg–Marquardt



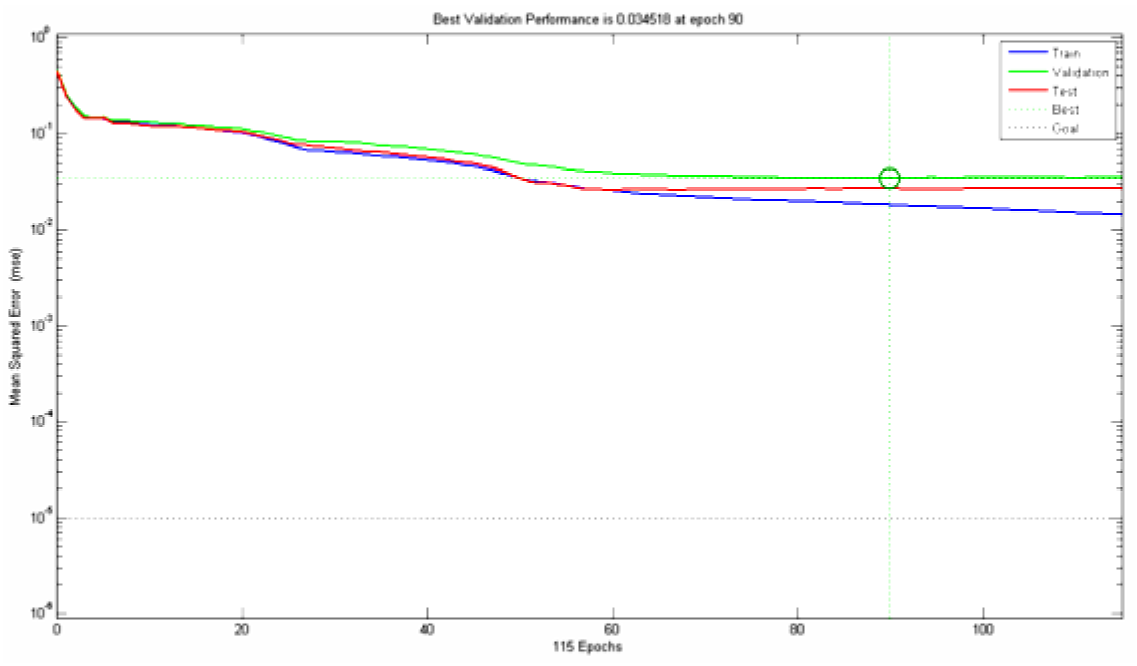


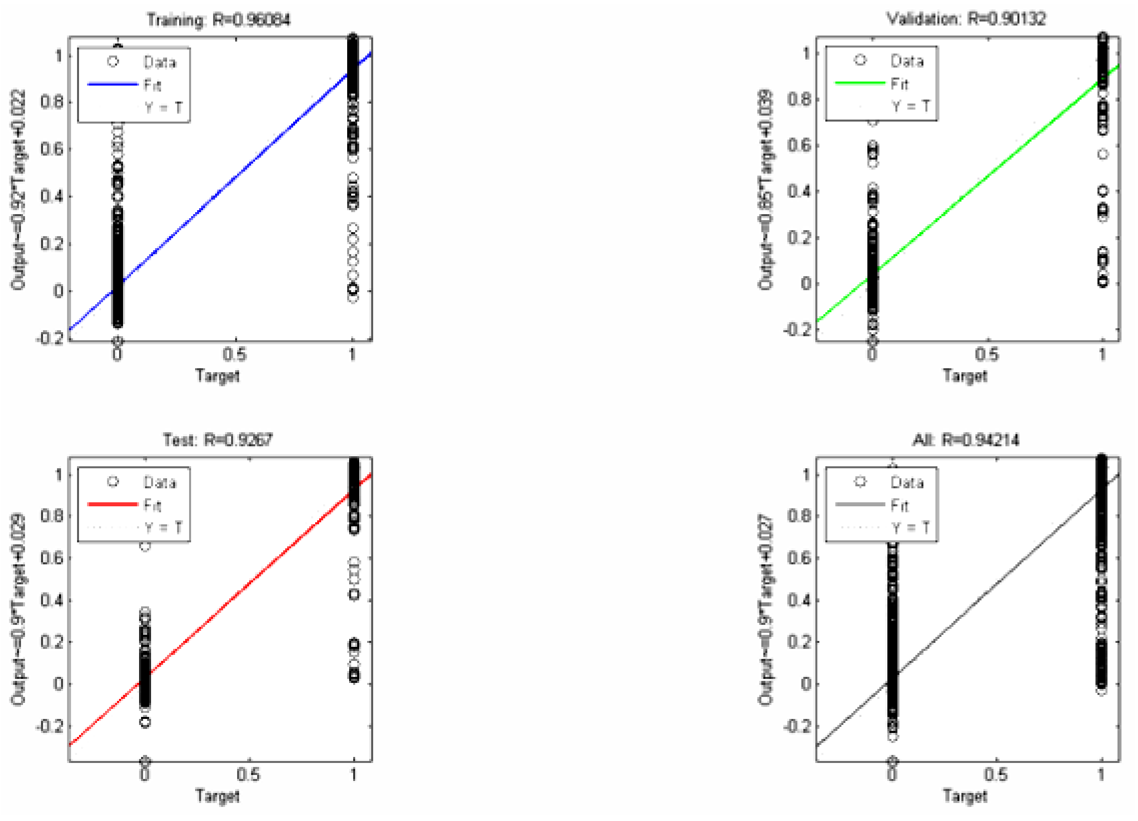
b) Bayesian Regularisation



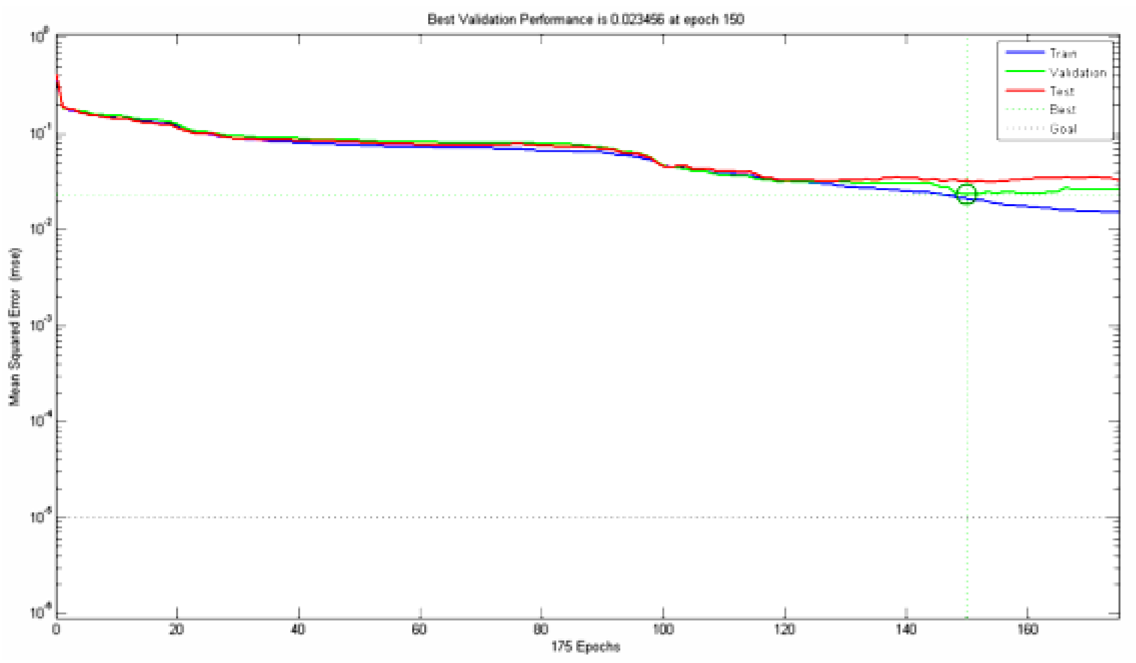


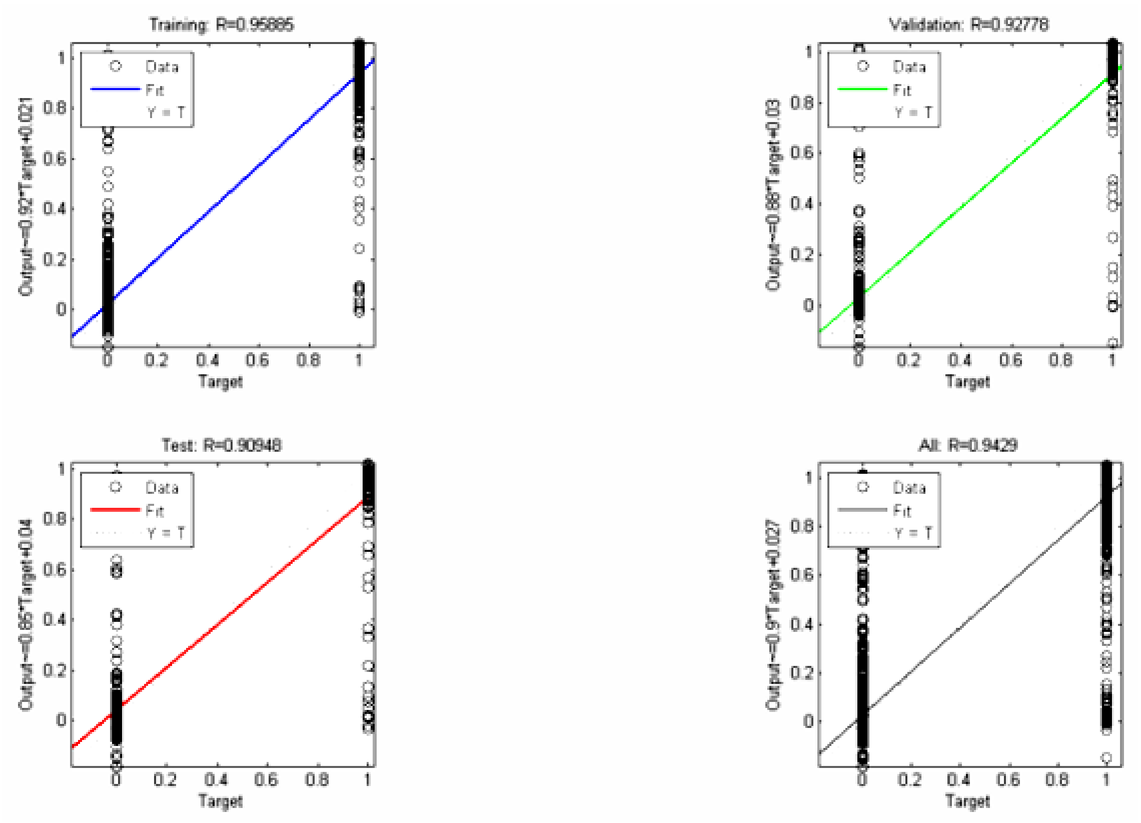
c) Resilient Backpropagation



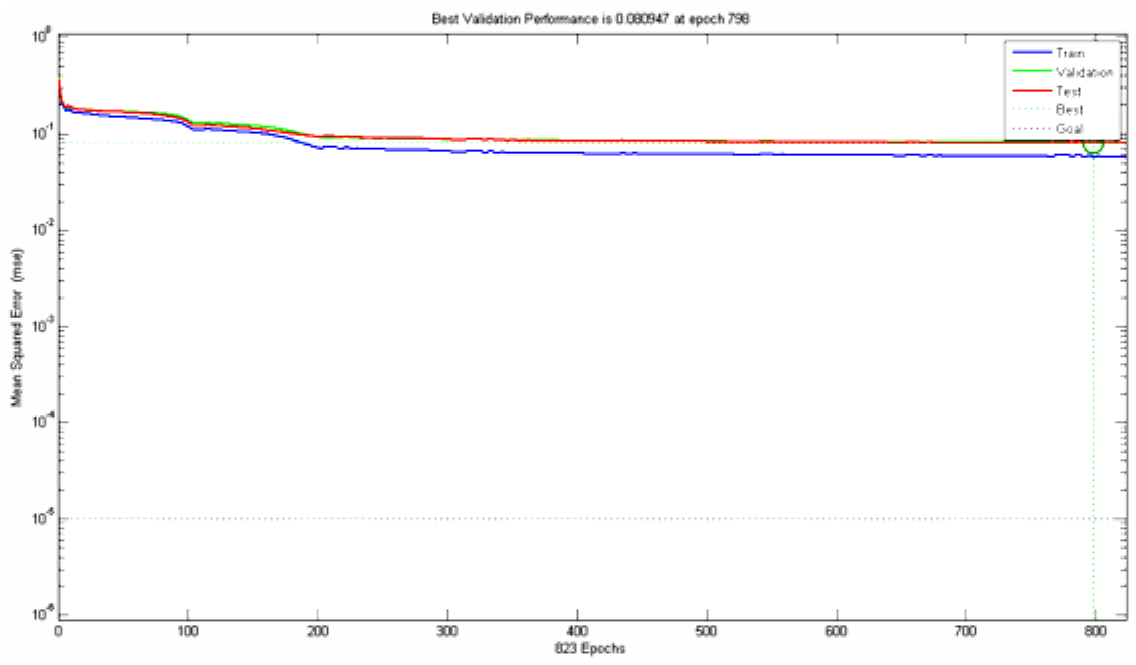


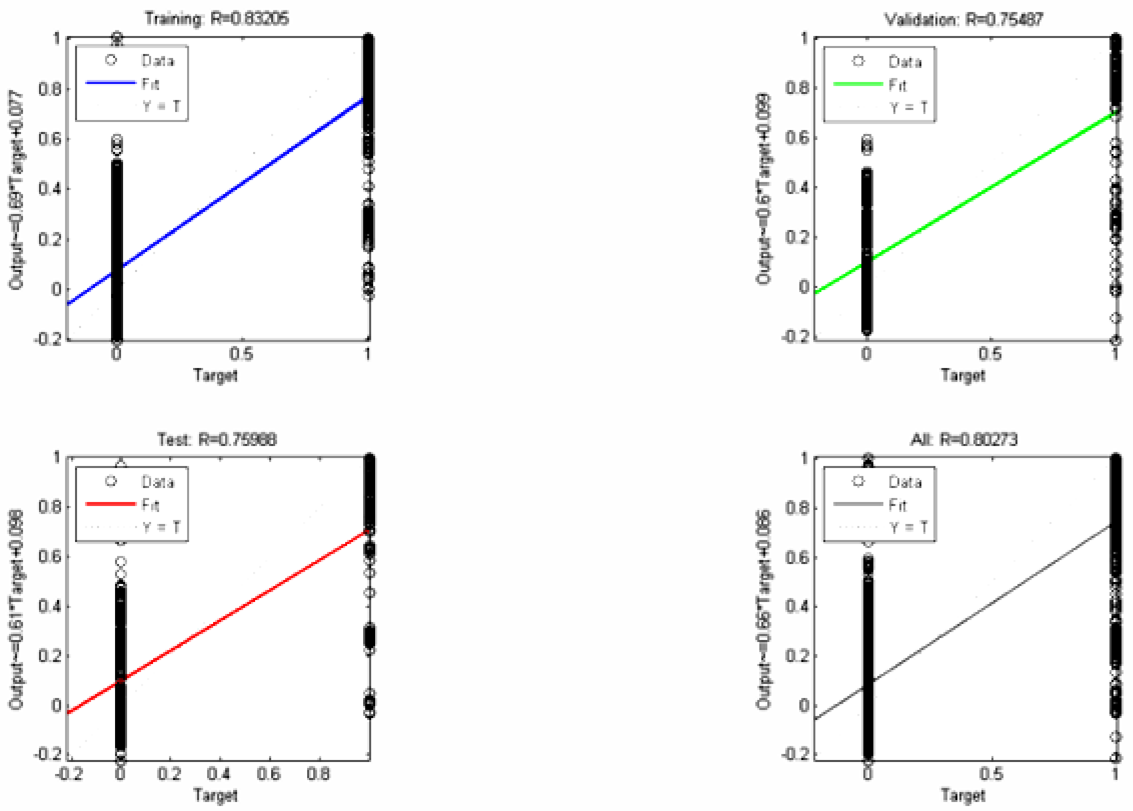
d) Scaled Conjugate Gradient



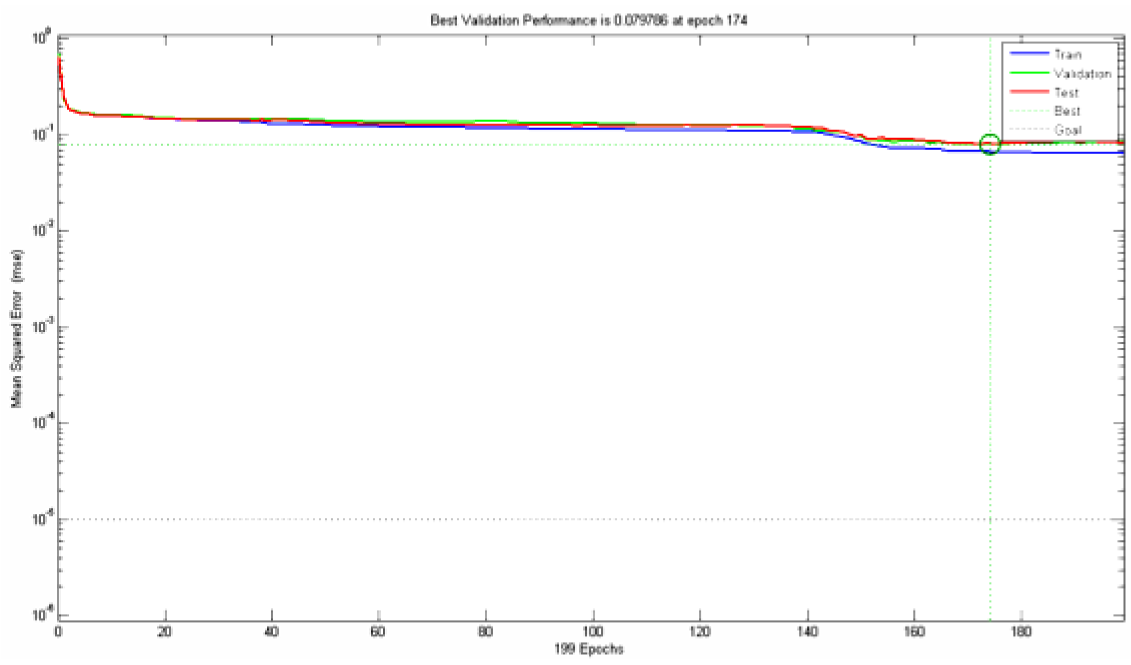


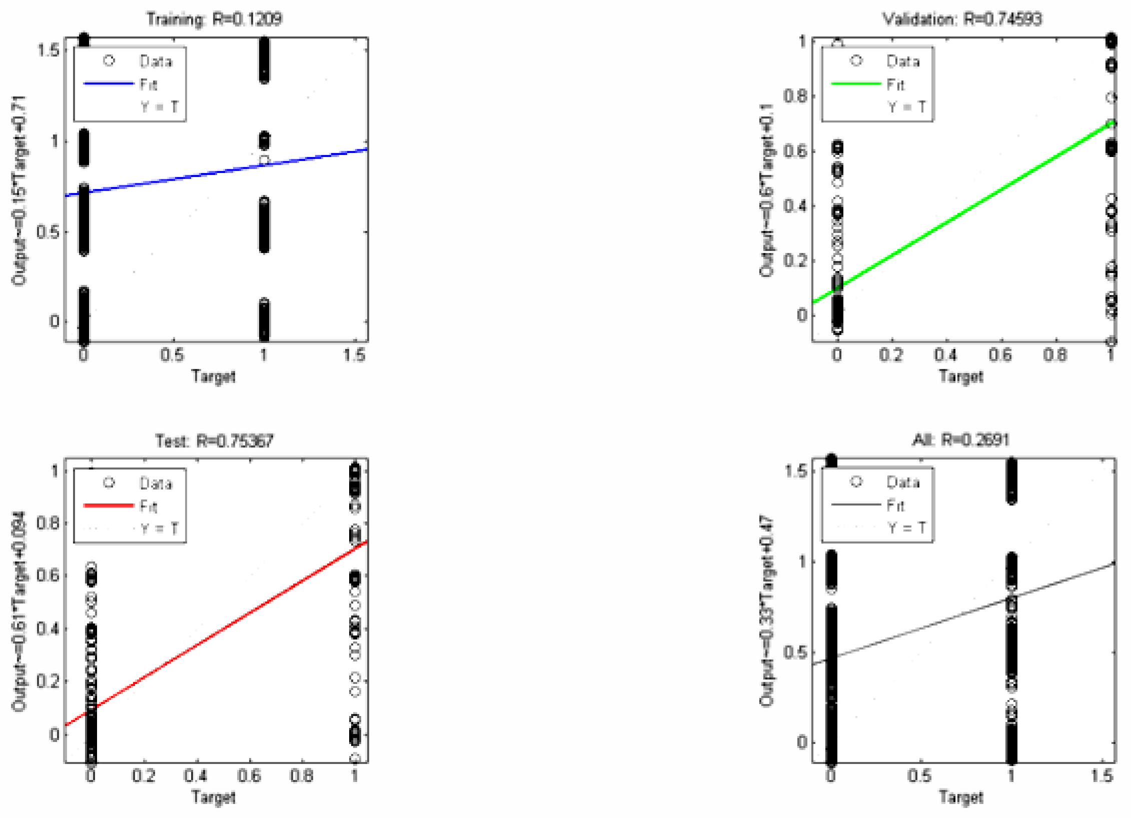
e) Variable Learning Rate Backpropagation





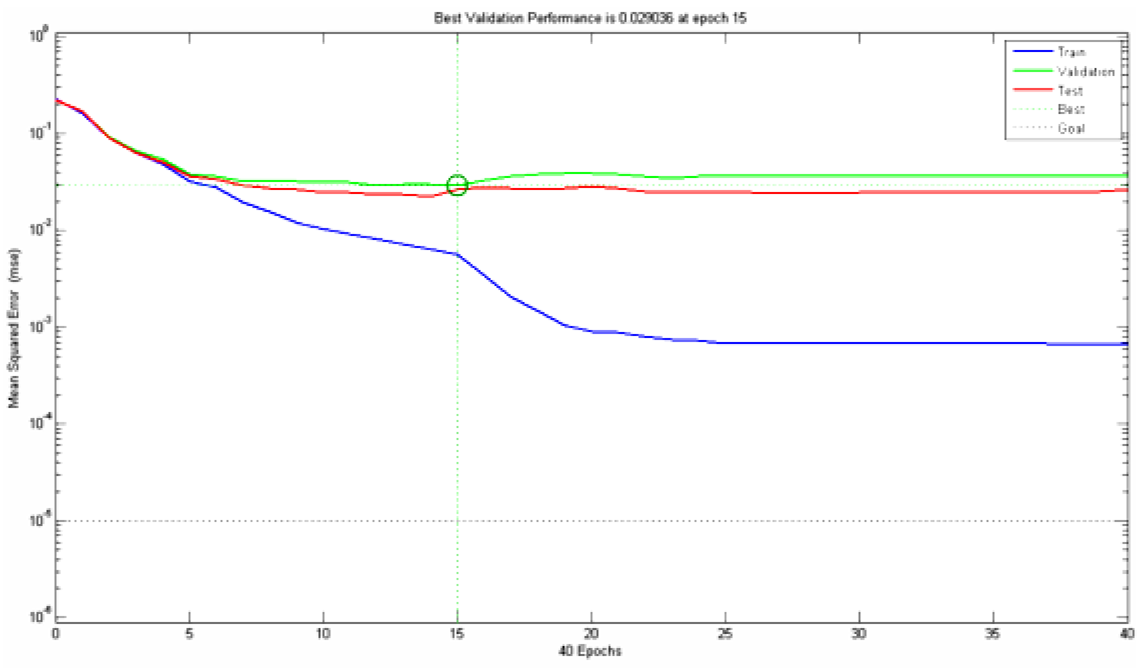
f) BFGS quasi–Newton Backpropagation

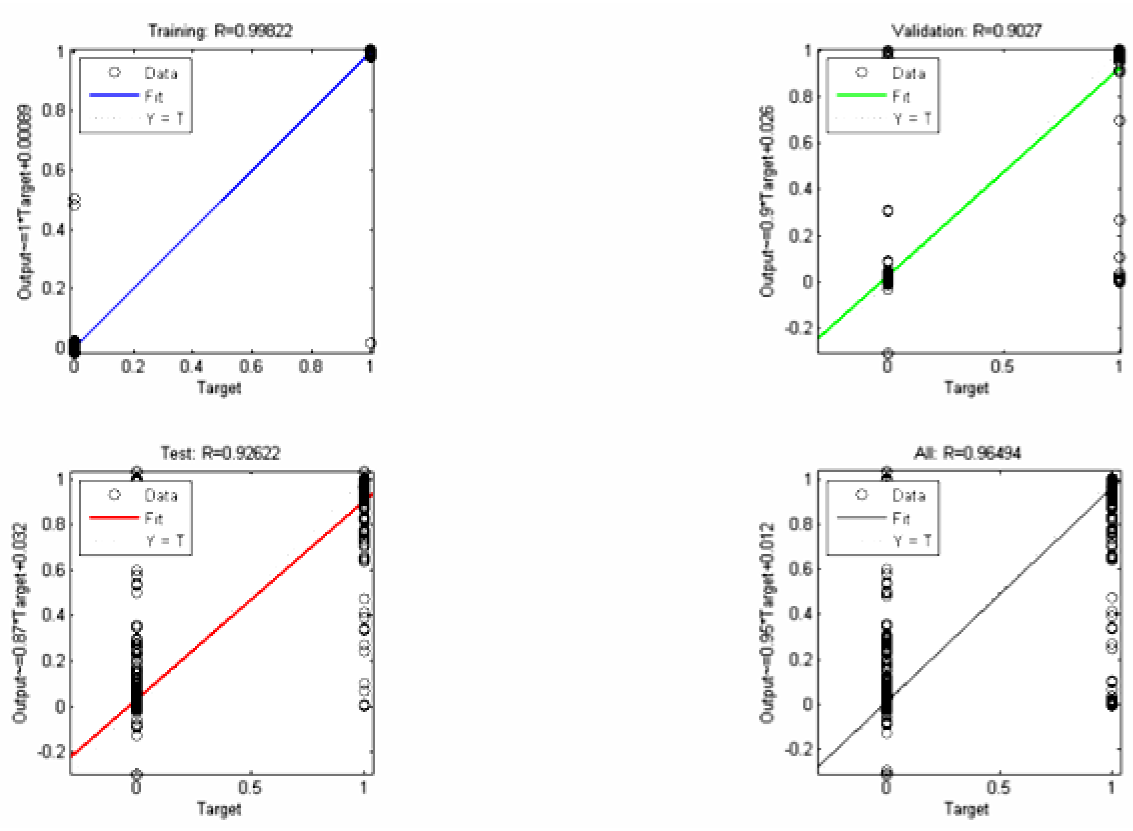




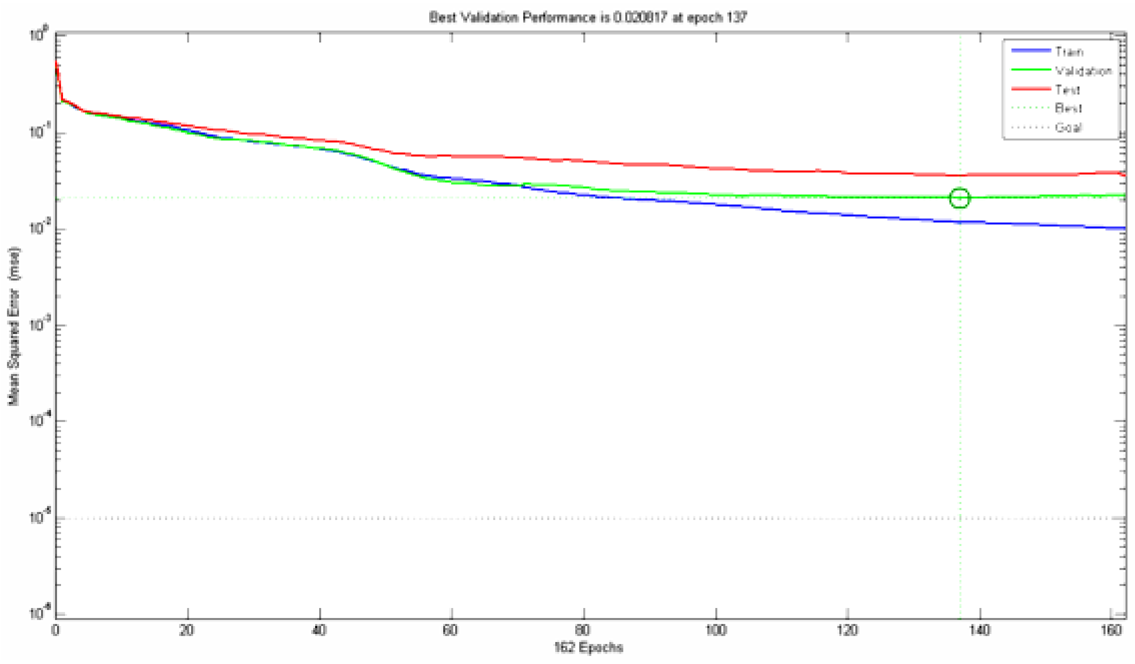
6) [10 8 5 4]

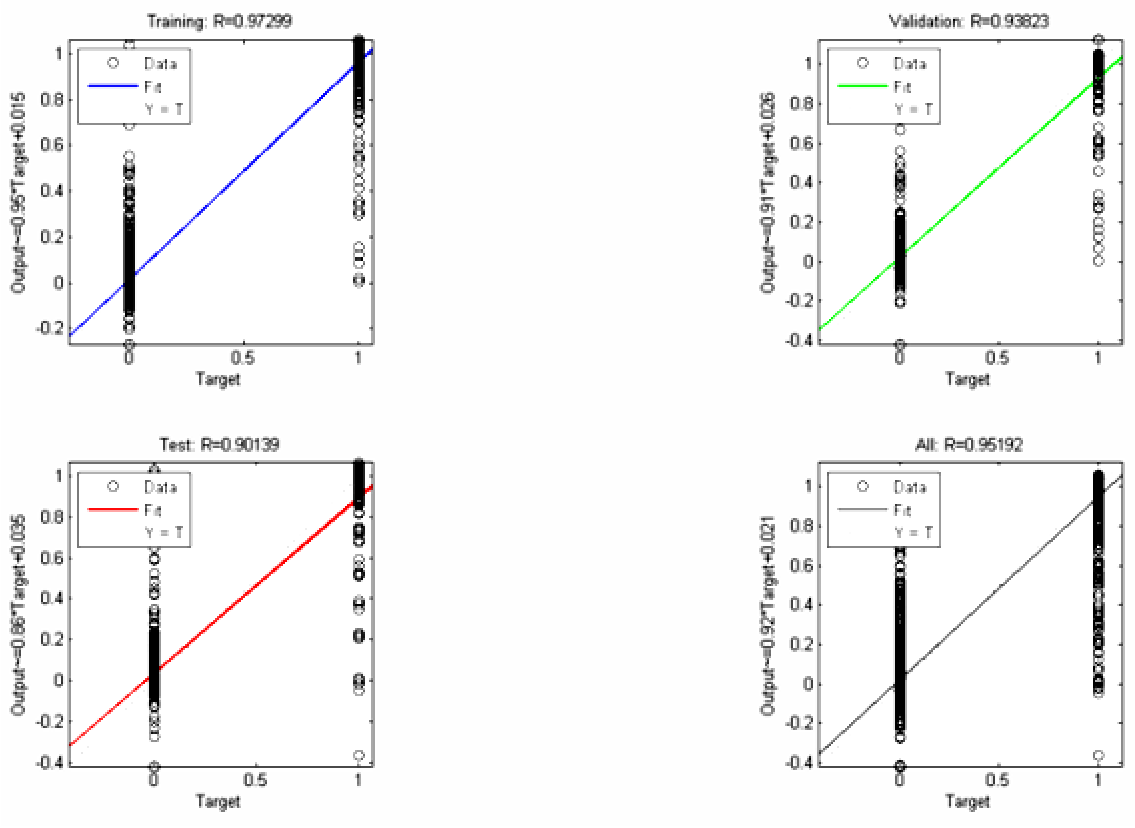
a) Levenberg–Marquardt



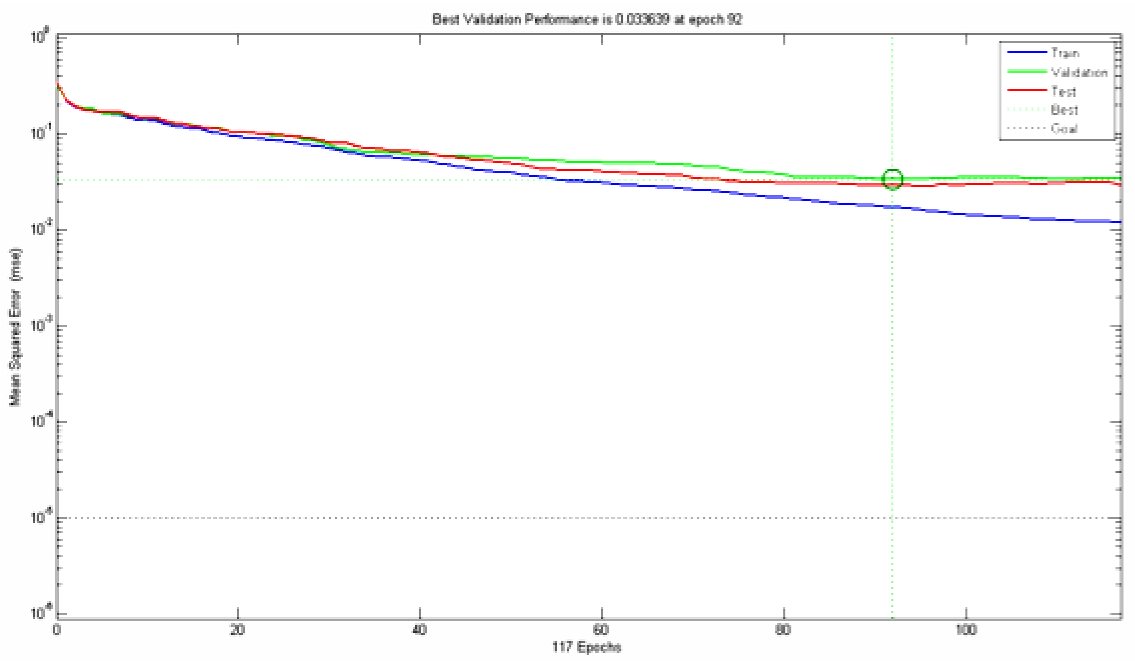


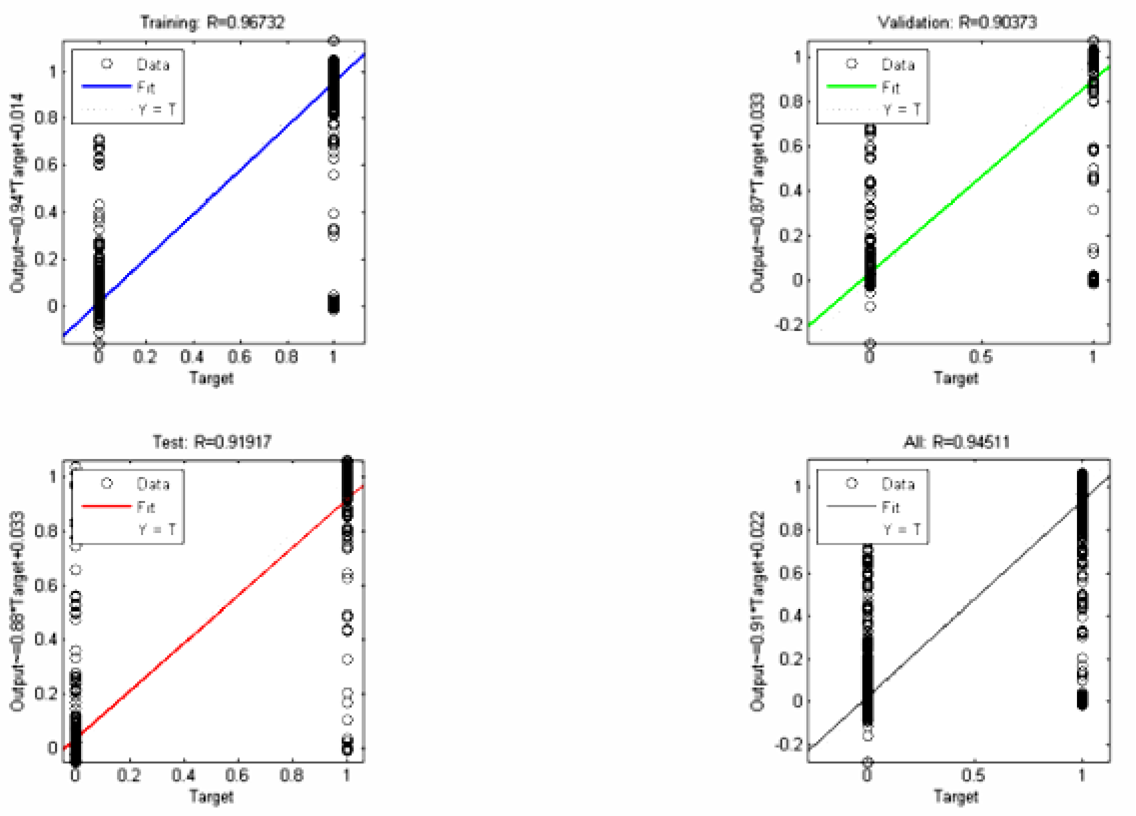
c) Resilient Backpropagation



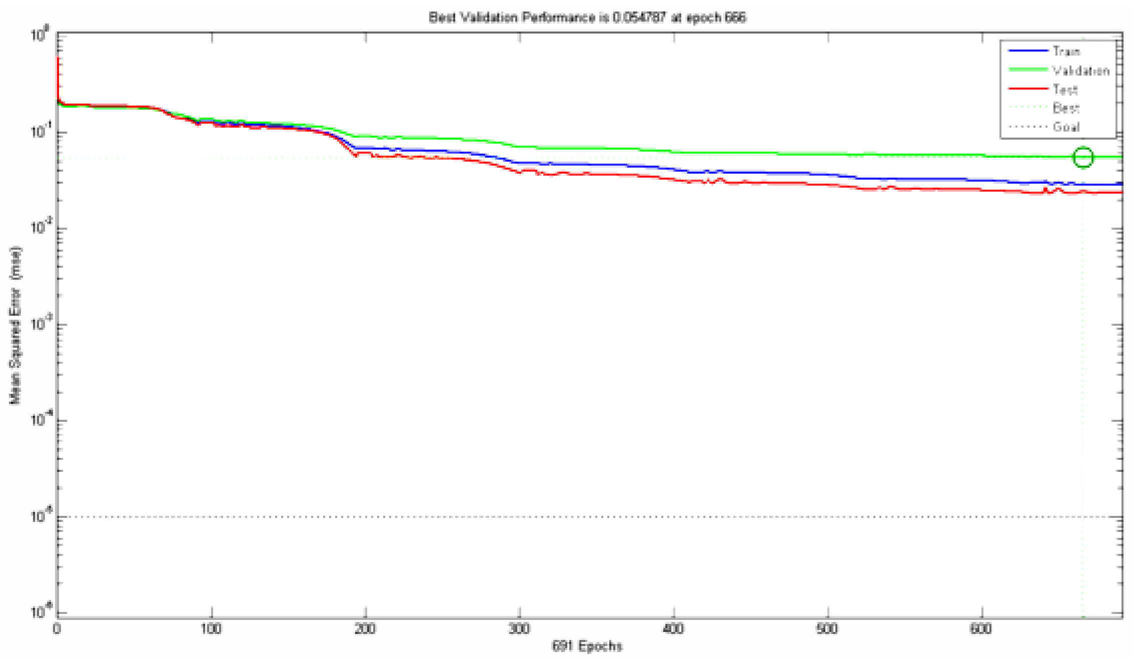


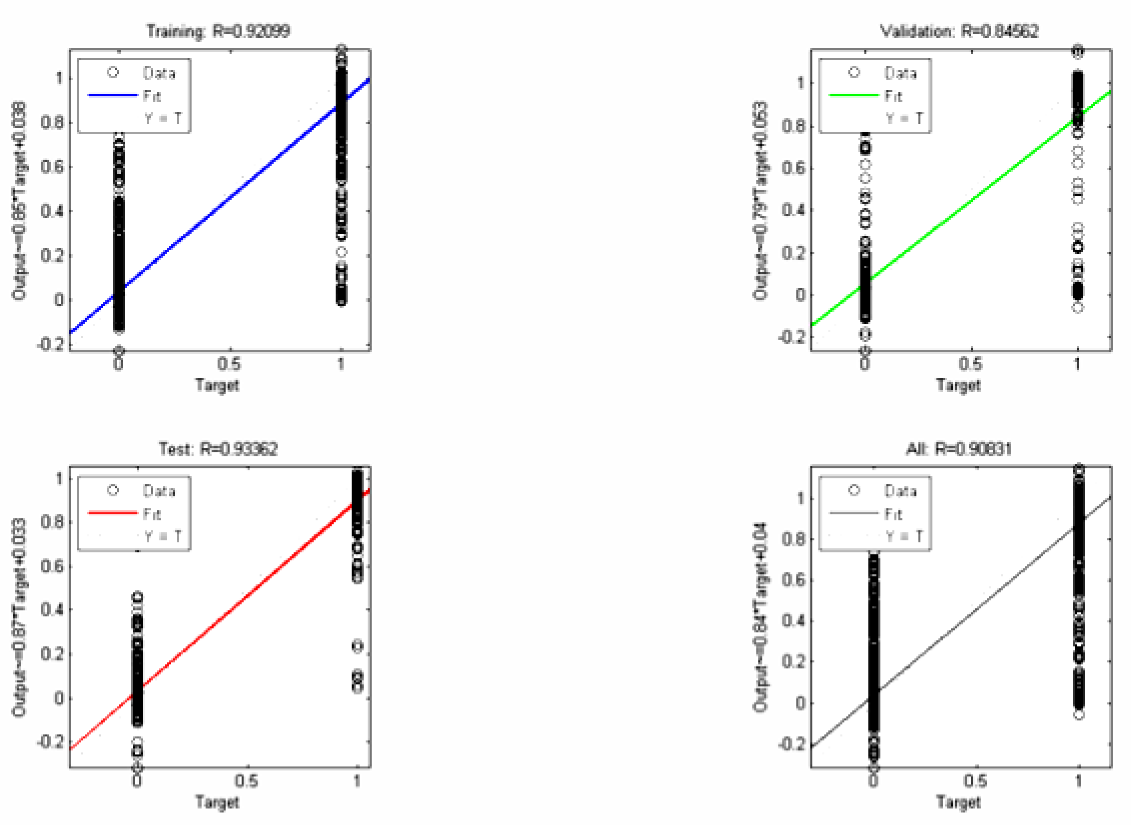
d) Scaled Conjugate Gradient



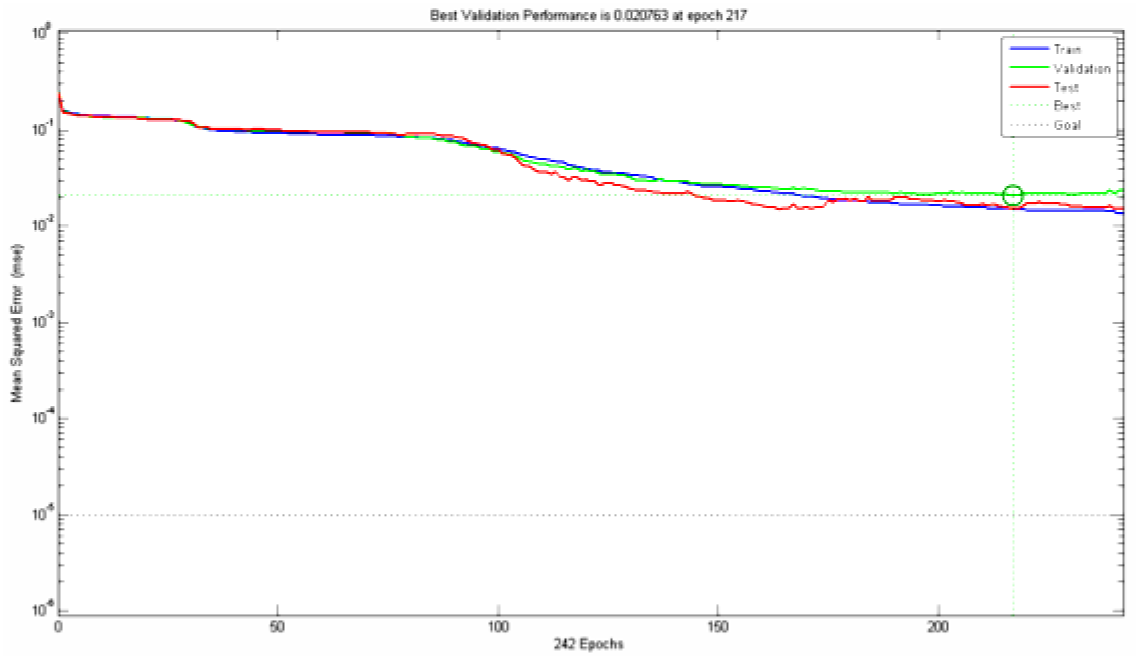


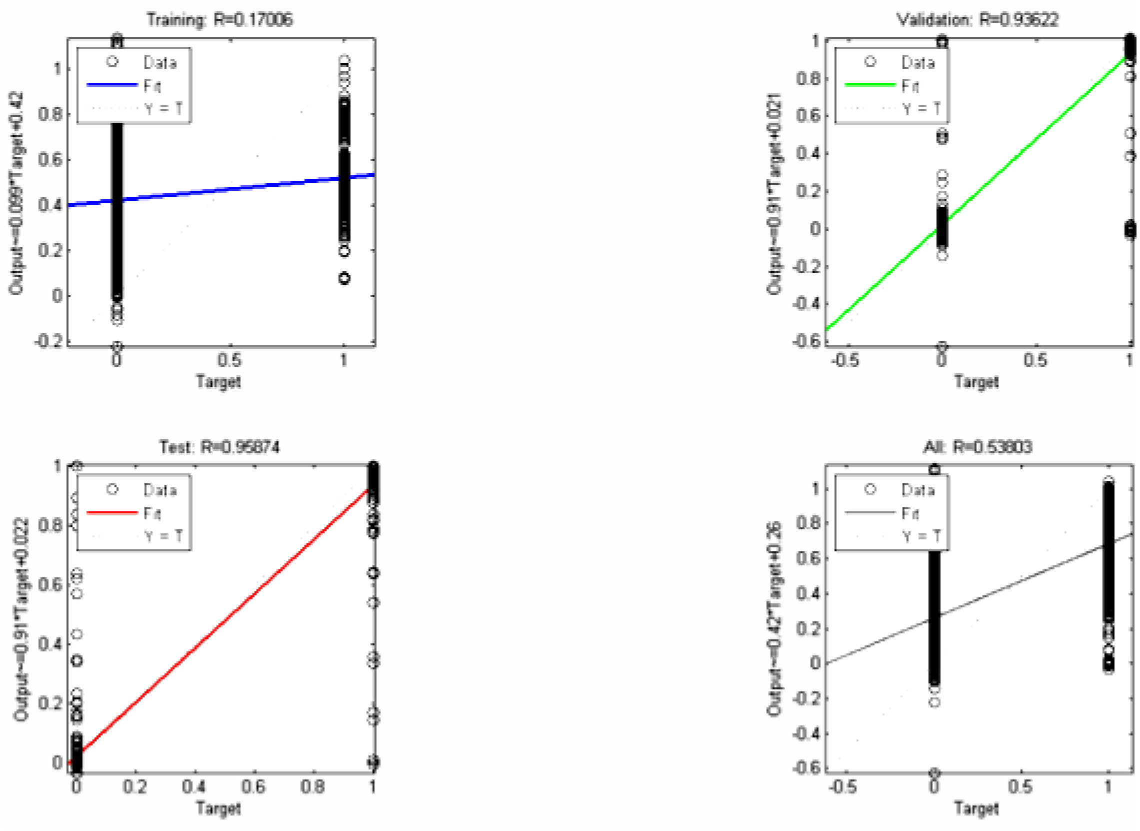
e) Variable Learning Rate Backpropagation

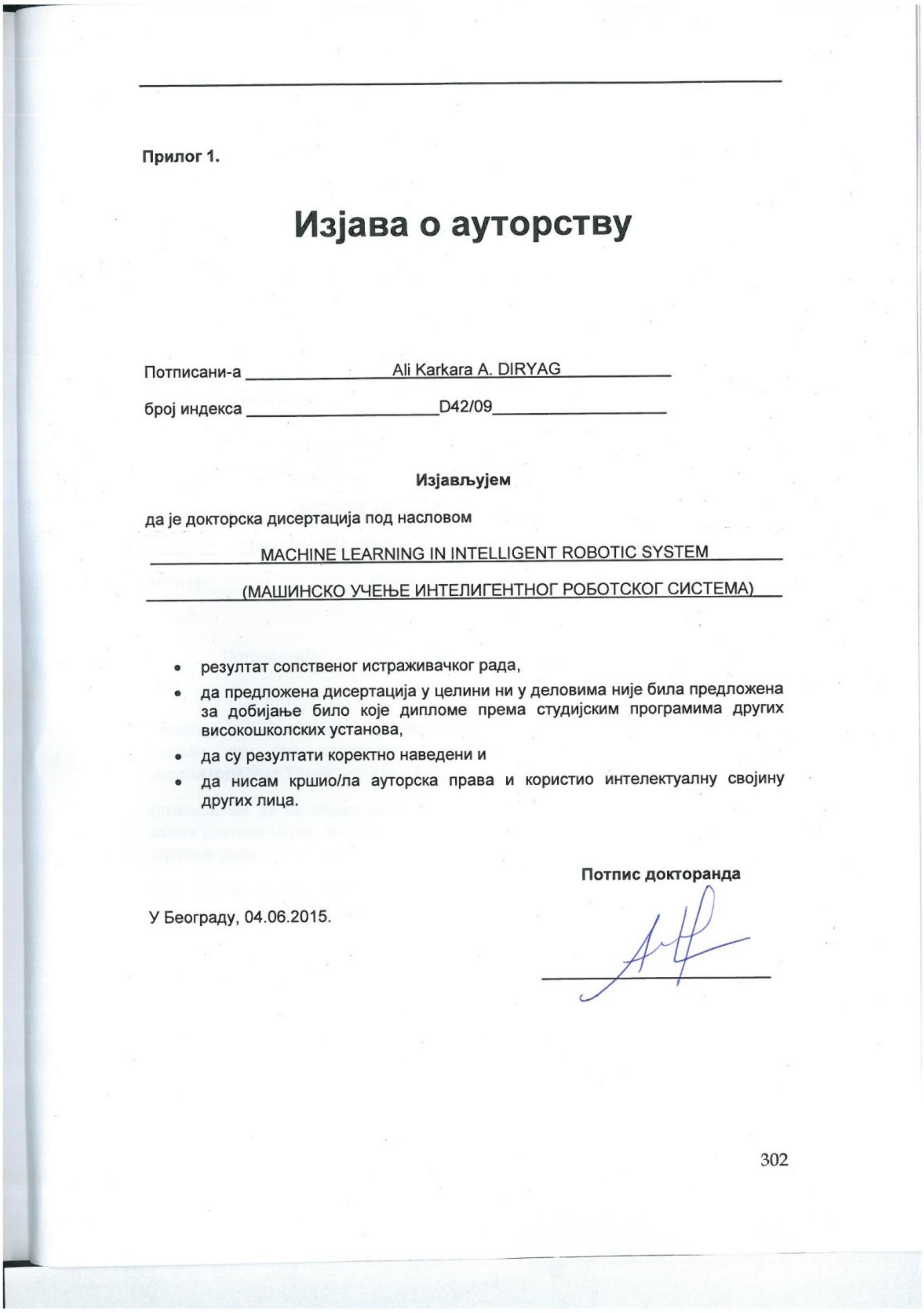




f) BFGS quasi–Newton Backpropagation







Прилог 1.

Изјава о ауторству

Потписани-а Ali Karkara A. DIRYAG

број индекса D42/09

Изјављујем

да је докторска дисертација под насловом

MACHINE LEARNING IN INTELLIGENT ROBOTIC SYSTEM

(МАШИНСКО УЧЕЊЕ ИНТЕЛИГЕНТНОГ РОБОТСКОГ СИСТЕМА)

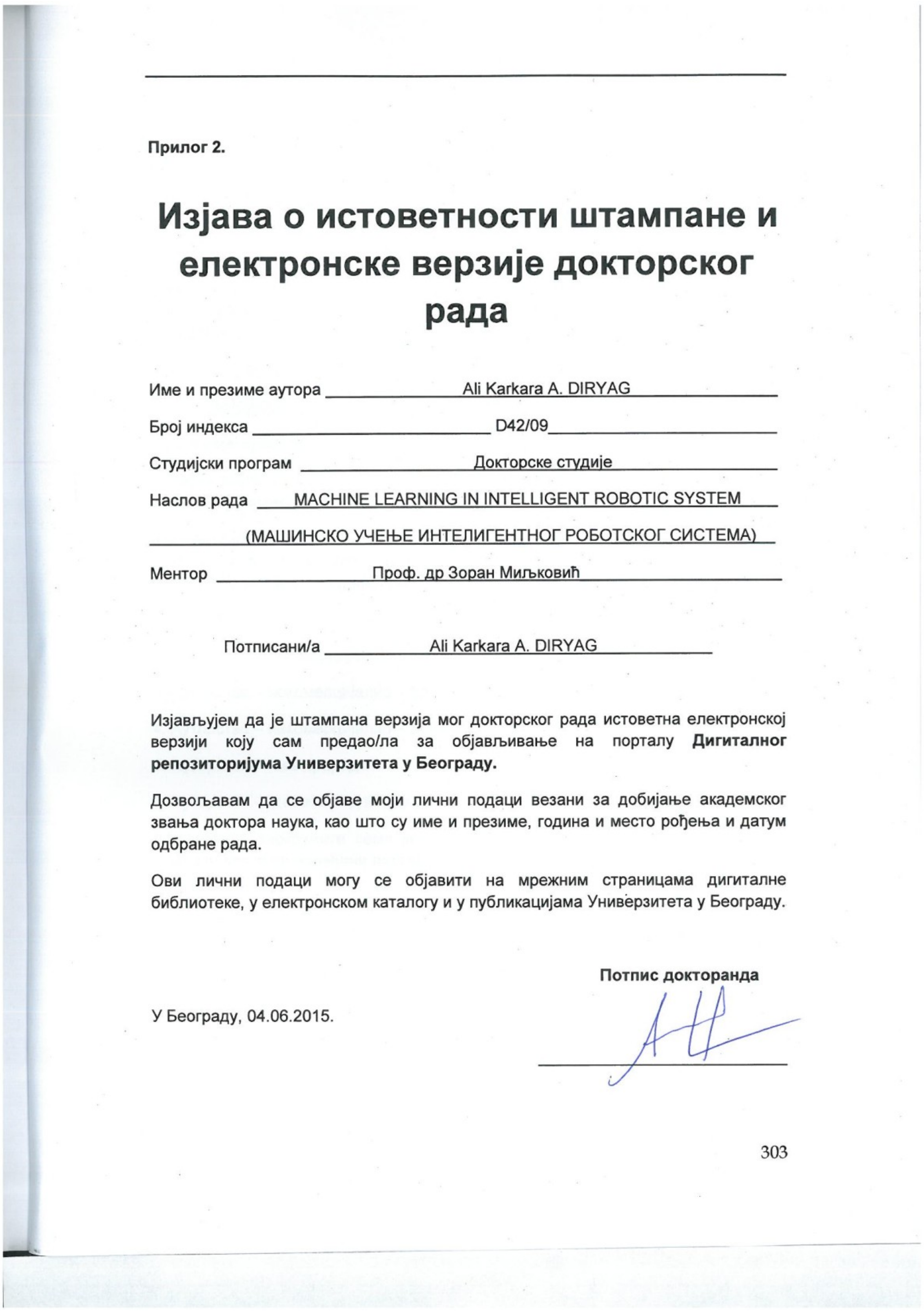
- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

[illegible]

У Београду, 04.06.2015.

302



Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора Ali Karkara A. DIRYAG

Број индекса D42/09

Студијски програм Докторске студије

Наслов рада MACHINE LEARNING IN INTELLIGENT ROBOTIC SYSTEM (МАШИНСКО УЧЕЊЕ ИНТЕЛИГЕНТНОГ РОБОТСКОГ СИСТЕМА)

Ментор Проф. др Зоран Миљковић

Потписани/а Ali Karkara A. DIRYAG

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис докторанда

У Београду, 04.06.2015.

[illegible]

303

Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић" да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

MACHINE LEARNING IN INTELLIGENT ROBOTIC SYSTEM

(МАШИНСКО УЧЕЊЕ ИНТЕЛИГЕНТНОГ РОБОТСКОГ СИСТЕМА)

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство

2. Ауторство - некомерцијално

[✓ 3. Ауторство – некомерцијално – без прераде]

4. Ауторство – некомерцијално – делити под истим условима

5. Ауторство – без прераде

6. Ауторство – делити под истим условима

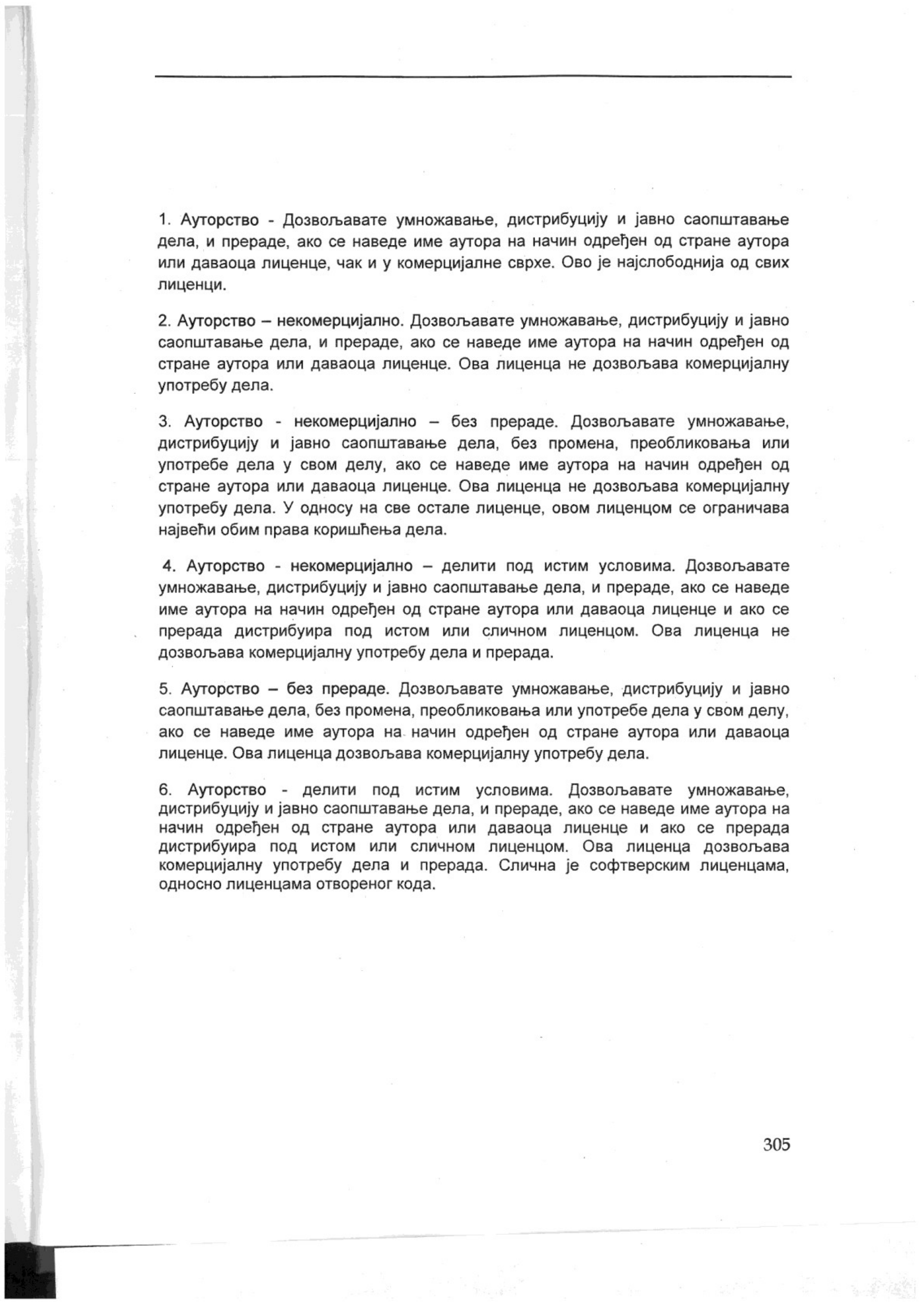
(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, 04.06.2015.

[illegible]

304



1. Ауторство - Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце, чак и у комерцијалне сврхе. Ово је најслободнија од свих лиценци.

2. Ауторство – некомерцијално. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела.

3. Ауторство - некомерцијално – без прераде. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела. У односу на све остале лиценце, овом лиценцом се ограничава највећи обим права коришћења дела.

4. Ауторство - некомерцијално – делити под истим условима. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца не дозвољава комерцијалну употребу дела и прерада.

5. Ауторство – без прераде. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца дозвољава комерцијалну употребу дела.

6. Ауторство - делити под истим условима. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца дозвољава комерцијалну употребу дела и прерада. Слична је софтверским лиценцама, односно лиценцама отвореног кода.